Automatic Discovery of Attributes in Relational Databases

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

In this work we design algorithms for clustering relational columns into *attributes*, i.e., for identifying strong relationships between columns based on the common properties and characteristics of the values they contain. For example, identifying whether a certain set of columns refers to *telephone* numbers versus *social security* numbers, or names of *customers* versus names of *nations*. Traditional relational database schema languages use very limited primitive data types and simple foreign key constraints to express relationships between columns. Object oriented schema languages allow the definition of custom data types; still, certain relationships between columns might be unknown at design time or they might appear only in a particular database instance. Nevertheless, these relationships are an invaluable tool for schema matching, and generally for better understanding and working with the data. Here, we introduce data oriented solutions (we do not consider solutions that assume the existence of any external knowledge) that use statistical measures to identify strong relationships between the values of a set of columns. Interpreting the database as a graph where nodes correspond to database columns and edges correspond to column relationships, we decompose the graph into connected components and cluster sets of columns into attributes. To test the quality of our solution, we also provide a comprehensive experimental evaluation using real and synthetic datasets.

Categories and Subject Descriptors

H.m [**Information Systems**]: Miscellaneous

General Terms

Algorithms

Keywords

Attribute discovery, Schema matching

1. INTRODUCTION

Relational databases are described using a strict formal language in the form of a relational schema. A relational schema specifies various properties of tables and columns within tables, the most important of which is the type of data contained in each column. There is a well defined set of possible primitive data types, ranging from numerical values and strings, to sets and large binary objects. The relational schema also allows one to define relationships between columns of different tables in the form of foreign key constraints. Even though the relational schema is a powerful description of the data, it has certain limitations in practice. In particular, it cannot accurately describe relationships between columns in the form of *attributes*, i.e., *strongly connected sets of values that appear to have the same or similar meaning within the context of a particular database instance*.

For example, consider a database instance that contains columns about telephone numbers and social security numbers. All such columns can be declared using the same primitive data type (e.g., decimal), but in reality there is never a case where these two types of columns need to be joined with each other: semantically, there is no reason why these columns should belong to the same type. Even though this fact might be known to users (or easy to deduce), it is nowhere explicitly specified within the schema. As another example, consider a database instance that contains a table of customer names and defines two views, one with European and one with Asian customers. Ostensibly, the customer name columns in the European and Asian views will not have any (or very few) values in common. Nevertheless, all three customer name columns belong to the same attribute. Moreover, suppose that there exists a fourth column that contains nation names. Clearly, nation names should not be classified in the same attribute as customer names even though these columns contain the same types of values (i.e., strings). Differentiating between these fundamentally different attributes can be an invaluable tool for data integration and schema matching applications, and, generally speaking, for better understanding and working with the data.

Existing schema matching techniques for identifying relationships between columns use simple statistics and string-based comparisons, e.g., prefix/suffix tests, edit distance, value ranges, min/max similarity, and mutual information based on q-gram distributions [11, 7, 8, 12, 13]. Other approaches use external information like thesauri, standard schemas, and past mappings. Our work on discovering attributes can be used as a valuable addition to all of the above, for designing automated schema mapping tools.

It is important to note here that attribute relationships are not always known in advance to database designers, so it is not always possible to encode them a priori (for example, by using constraints or object oriented schema languages). Certain relationships might hold solely for a particular database instance, others develop over time as the structure of the database evolves, yet others are obvious in hindsight only. Furthermore, there exists a large number of legacy databases (sometimes with sizes in the order of hundreds of tables and thousands of columns) for which schema definitions or

SIGMOD'11, June 12–16, 2011, Athens, Greece.

folklore knowledge of column meanings might have been lost. To make matters worse, in many practical situations users have access only to a keyhole view of the database (due to access privileges). In such cases users access the data through materialized views, without any information about the underlying schema, or even about the view definitions. In other words, as far as the user is concerned, all schema information has been lost.

Our approach for discovering attributes is purely data oriented. We do not examine solutions that depend on external knowledge about the data. We compute various statistical measures between all pairs of columns within the database, and derive positive and negative relationships between certain pairs of columns. Viewing the database instance as a graph where every column is a node and every positive/negative relationship is an edge, we decompose the graph into connected components. Then, we further decompose each component into a set of attributes.

In particular, in order to discern the type of relationship between any pair of columns, we use Earth Mover's Distance to find the similarity between the distributions of the values contained in the columns. We introduce two types of connections, one based on the *overall distribution* of values and one based on the *intersection distribution* (distribution with respect to the common values only). Low distribution similarity strongly suggests no attribute ties. High intersection distribution similarity suggests very strong attribute ties. We also propose the notion of a *witness column* for introducing relationships by indirect association (i.e., for columns that have no values in common directly, but share a lot of values with the witness column).

Our main contribution is to provide a robust, unsupervised solution that reports a clustering of columns into attributes. In addition, we perform a comprehensive empirical study using real and synthetic datasets to validate our solution, and show that it has very high precision in practice.

Section 2 gives some necessary background and definitions. Section 3 presents our solution. Section 4 discusses performance issues. Section 5 presents a comprehensive experimental evaluation of the proposed technique. Section 6 discusses related work. Finally, Section 7 concludes this paper.

2. DEFINITIONS AND BACKGROUND

Conventionally, in relational database terminology the term attribute is a synonym for a column. In this work, we use the term attribute to refer to a much stronger notion, based on the *actual meaning* of the values contained in a column. Formally:

DEFINITION 1 (ATTRIBUTE). *An* attribute *is a set of relational columns, such that columns in the same attribute are semantically equivalent to each other.*

In other words, an attribute is a logical notion based on common properties and characteristics of the values contained in the columns comprising that attribute.

For example, Figure 1 shows an excerpt of the schema of the TPC-H benchmark [16], which models a business environment and contains information about products, suppliers, customers, orders, etc. The figure shows three tables, CUSTOMER, NATION and ORDERS, and foreign-primary key relationships between some columns of these tables. A customer is associated with six columns in this example: CUSTKEY, NAME, ADDRESS, NATIONKEY, PHONE and COMMENT. Since CUSTOMER.NATIONKEY is a foreign key of NATION.NATIONKEY, the two NATIONKEY columns are by definition semantically equal and hence they belong to the same attribute. The same is true for ORDERS.CUSTKEY and CUSTOMER.CUSTKEY. Another example appears in Figure 2 which shows a slightly more

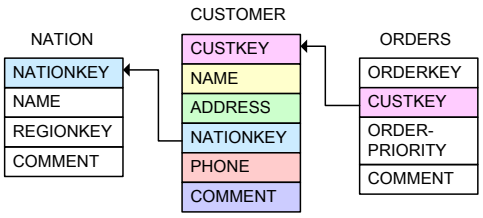


Figure 1: Excerpt of the TPC-H schema.

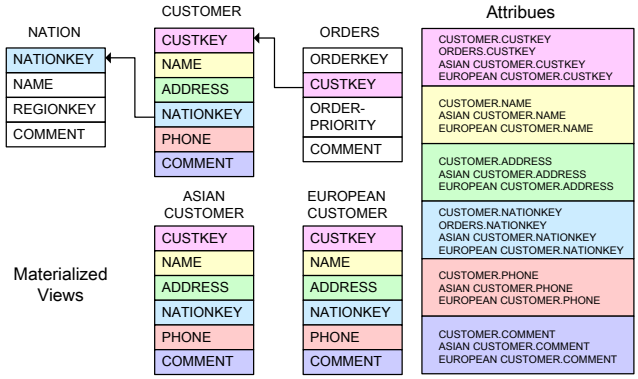


Figure 2: Attributes in TPC-H example, which contains three base tables and two materialized views of CUSTOMER table.

complex scenario that considers the existence of materialized views, i.e., ASIAN CUSTOMER and EUROPEAN CUSTOMER created from the CUSTOMER table based on NATIONKEY. The ideal clustering of the six columns contained in the CUSTOMER table is shown on the right side of the figure. Clearly, all columns from the three related tables belong to the same attribute, even if there is no direct association specified in the schema (e.g., in the form of primary/foreign keys) and despite the fact that, probably, EUROPEAN CUSTOMER and ASIAN CUSTOMER have no values in common.

We now formalize the problem of attribute discovery as follows:

DEFINITION 2 (ATTRIBUTE DISCOVERY). *Given a collection of relational tables, denoted* $\mathbf{T}$*, let* $\mathbf{C}$ *be the set of all columns in* $\mathbf{T}$. Attribute Discovery *is the process of partitioning* $\mathbf{C}$ *into* m *clusters* $\mathbf{A} = \{A_1, A_2, \ldots, A_m\}$ *such that each* $A_k = \{C_1^k, C_2^k, \ldots, C_{n_k}^k\}$ *is an attribute with respect to the set of tables* $\mathbf{T}$.

According to Definition 1, two columns C and C' are part of the same attribute if and only if semantically they behave the same. The semantics of two columns can be inferred by the type of relationship these columns have within a database instance. We define the following relationship types:

1. A primary/foreign key;
2. Two foreign keys referring to the same primary key;
3. A column in a view and the corresponding column in the base table;
4. Two columns in two views but from the same column in the base table;
5. No explicit relationship but semantically equivalent (e.g., non-key, customer name columns from different tables).

The first four relationship types are, by definition, indicators of strong attribute ties. The fifth relationship type encompasses all

columns that are semantically equivalent where this information cannot be inferred from the database schema, but only from the actual values contained in the columns. Only relationship type 1 has been studied in the past. To the best of our knowledge no previous work has studied relationship types 2-5. Nevertheless, existing work can easily be adapted to identify types 2-4. In what follows, we list a set of existing techniques that can be used to identify pairs of columns belonging to these relationship types. In each case, we point out why they are insufficient for identifying *all* relationship types - particularly type 5.

2.1 Name Similarity

It is natural to consider using the similarity of column names to infer column semantics, since, to a certain extent, names reflect the meaning of the values within a column. Indeed, previous work, especially in the area of schema matching, has applied this technique to identify associations between columns [14]. However, this is not always a robust solution for three reasons. First, a given database might not use strict naming conventions. Second, columns with no meaningful associations oftentimes have similar or even identical names. For instance, a column called NAME appears in both the NATION and CUSTOMER tables of TPC-H, even though the two columns refer to two semantically unrelated concepts. Third, two semantically related columns may happen to have very different names. For example, the columns in a view might have completely different names from the source columns in the base table. This happens when views are generated automatically, or when long, representative column names have to be abbreviated due to length constraints (e.g., the 30 characters limit in Oracle). Hence, simply relying on column name similarity can lead to both false positives and false negatives, for the purpose of discovering attributes.

2.2 Value Similarity

Another straightforward technique is to consider the similarity of the data values contained in a set of columns. The Jaccard coefficient $J(C_1, C_2) = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|}$ (or any other set similarity measure) can be used for this purpose, which can be efficiently estimated in practice [6]. However, this idea has its own drawbacks. For example, in our TPC-H database instance, column CUSTOMER.CUSTKEY contains values from 1 to 150,000, while column PART.PARTKEY contains all integers from 1 to 200,000. The overlap of the values in these two columns is very high: their Jaccard coefficient is 0.75. Nevertheless, the columns are not semantically related. Conversely, two columns can have a strong semantic relationship but no common values at all, e.g., EUROPEAN CUSTOMER and ASIAN CUSTOMER. Of course, one could argue that in this case the two columns belong to two different attributes. However, in our solution we would like to cluster these columns primarily as customer names (as opposed to, for example, nation names) and, optionally, also partition them into sub-attributes. In this case, using data value similarity alone would lead to a false dismissal.

2.3 Distribution Similarity

Data distribution similarity has been used before for finding associations between columns, for example, using Earth Mover's Distance (EMD) to discover meaningful primary/foreign key constraints [17]. Earth Mover's Distance is a measure of similarity between two data distributions on an underlying metric space. In the context of primary/foreign key relationships, it can be used based on the observation that, for the majority of cases, the distinct values in a foreign key column are selected at random from the values of the primary key column, and hence have similar underlying distributions. (This might not always be true, for example, when the foreign key column is produced in a way that some correlations are preserved, as is the case with locations and telephone number prefixes; a foreign key column containing telephone numbers from a specific geographical region would result in a well defined range of numbers and would not be random with respect to a primary key column containing numbers from all regions.) Notice that in the case of primary/foreign keys there is also an implicit *containment* assumption, i.e., that the values in the foreign key are a subset (or approximately a subset) of the values in the primary key.

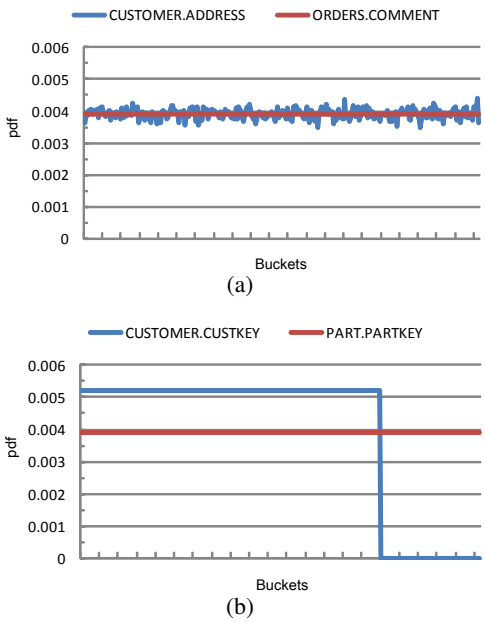


Figure 3: Data distribution histograms of two examples from TPC-H.

Earth Mover's Distance is defined as follows:

DEFINITION 3. *Given probability density functions (pdfs) C and C' on an underlying metric space, let a unit amount of work be equal to moving a unit of probability mass for a unit distance. Then, $EMD(C, C')$ is equal to the minimum amount of work needed to convert pdf C into pdf C'.*

The smaller the EMD value between two distributions is, the more similar these distributions are considered to be. In the context of sets of values, EMD is defined as the minimum amount of work needed to convert one set of values into another, where a unit of work is defined according to the *rank* of the values in the sorted order of the union of the two sets. For example, consider two sets of strings. First, we take the union of strings, sort them lexicographically and assign to each string its rank in the lexicographic order. Then, the amount of work needed to convert one string into another is equal to the distance of their ranks. For numerical values the only difference is that we sort the values in numeric instead of lexicographic order. See [17] for more details. Notice that we consider rank distributions for EMD computation purposes. Thus, the type of sorting affects the ranks of values and hence the distributions.

Computing distribution similarity is a necessary step in our setting, since we can use it to discover primary/foreign key constraints (one of the core relationship types we are interested in). However, it is not sufficient for our purposes. Consider for example the values in CUSTOMER.ADDRESS and ORDERS.COMMENT. If we sort them in lexicographic order for the purpose of computing EMD, they follow very similar distributions. The proportion of strings from one column that fall within a given range of strings from the

other column in lexicographic order is very similar. To illustrate this point, we plot the two distributions in Figure 3(a). The buckets are constructed using an equi-depth histogram based on the quantiles of column ORDERS.COMMENT. Then, we simply tally the number of strings from column CUSTOMER.ADDRESS that fall within each bucket. The plot clearly shows that the values in CUSTOMER.ADDRESS also follow a nearly uniform distribution across buckets. Indeed, the EMD between the two columns is only 0.0004. Still, computing distribution similarity does eliminate a large number of other column pairs: for example, the histograms for columns CUSTOMER.CUSTKEY and PART.PARTKEY, whose EMD is 0.125, are shown in Figure 3(b). We conclude that EMD values are a useful but insufficient filter. The main reason why EMD works for discovering primary/foreign key relationships, but not for discovering attributes, is the fact that for primary/foreign keys a containment relationship needs to hold: By definition, most, if not all, values from the foreign key must belong to the primary key. Thus, we would never consider ORDERS.COMMENT and CUSTOMER.ADDRESS as a valid primary/foreign key candidate in the first place, since they are not expected to have many strings in common. By contrast, for columns belonging to the same attribute no containment relationship needs to hold (e.g., EUROPEAN CUSTOMER and ASIAN CUSTOMER).

3. ATTRIBUTE DISCOVERY

It is clear that simply applying the aforementioned methods for discovering attributes will not yield accurate results. In this section we present a novel two-step approach. Intuitively, most of the time columns that belong to the same attribute tend to contain values that are drawn from the same underlying distribution. Conversely, if the values of two columns have different distributions, they more likely belong to different attributes. Therefore, here we advocate an algorithm that uses data distribution similarity, based on EMD, for partitioning columns into *distribution clusters*. This first step is used to separate columns into major categories, for example clusters that contain strings and clusters that contain only numerical columns. Furthermore, this step will also separate numerical columns with widely differing distributions, based for example on the range of values within the columns.

The use of EMD to create distribution clusters has some limitations. First and foremost, as can be seen from the example in Figure 3(a), not all the columns in one distribution cluster belong to the same attribute, especially when it comes to columns containing strings. String columns tend to have very similar distributions irrespective of the attribute these strings are derived from (e.g., addresses and comments). Second, using EMD might place some columns that belong to the same attribute into different clusters. This will happen for example if several views are defined on a table containing regions and telephone numbers, and the views select telephone numbers only from particular regions. Clearly, the telephone numbers in each view will start with a particular prefix. These correlations result in significantly different distribution of values between columns of the same attribute, rendering distribution based measures ineffective.

To solve the first problem, we use a subsequent, refinement phase that relies on computing the similarity of the distribution based on the intersection of two columns. We also use indirect associations through witness columns for cases where two columns have no values in common. We leave the solution of the second problem as future work, and identify this scenario as a limitation of our algorithm. From our experience, columns that have been generated based on indirect correlations do exist, but are rare in practice.

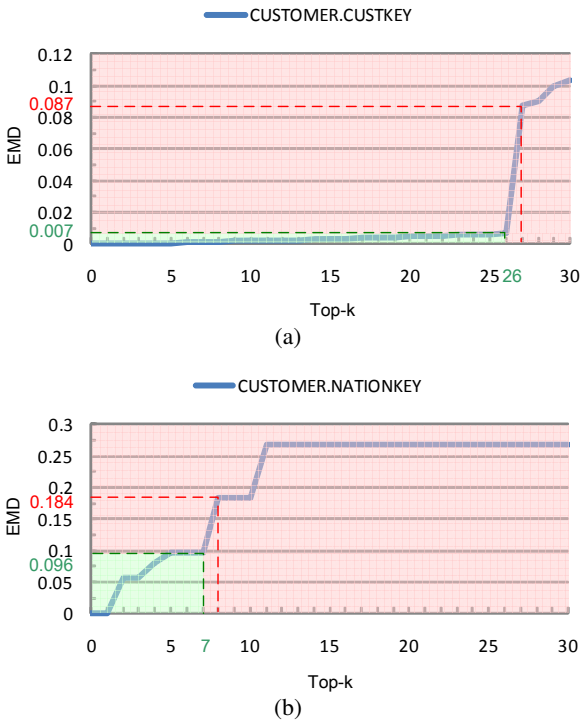


Figure 4: EMD plot of two examples in TPC-H.

3.1 Phase One: Computing Distribution Clusters

Given a set of columns $\mathbf{C}$, we form distribution clusters by computing the EMD between all pairs of columns in $\mathbf{C}$. Since EMD is a symmetric measure, this step requires $\frac{|\mathbf{C}|(|\mathbf{C}|-1)}{2}$ EMD computations. Then, given all pairwise EMDs we need to decide how to partition columns into clusters. The main idea here is to derive for every column a set of neighboring columns with small EMD, such that no two columns that intuitively belong to the same attribute are ultimately split into separate clusters. For every column C, we sort its EMD values to every other column in increasing order. Now we have to choose a cutoff EMD threshold that will determine the neighborhood $\mathbf{N}_C$ of C. After all neighborhoods have been determined, we will form a graph G_D, where nodes correspond to columns and undirected edges are created between a column C and all other columns in $\mathbf{N}_C$. Finally, the distribution clusters are formed by computing all connected components in G_D.

In practice, a small EMD value is not only subjective, but also attribute dependent. From our experience, different types of attributes exhibit significantly different behavior in terms of the similarity of the distribution of values across columns of that attribute, even when they are of the same primitive data type. We illustrate this with an example from TPC-H in Figure 4. For columns CUSTOMER.CUSTKEY and NATION.NATIONKEY, we sort their EMD values to all other columns in the database in increasing order of EMD, and plot the top-30 results. From careful, manual analysis of the data, we have determined that the columns belonging to the same attribute as CUSTKEY and NATIONKEY fall in the green region of the plots. We can clearly see that both the cutoff EMD value and the value of k that bound the green regions in the plots, differ significantly for these two attributes. For our purposes we would like to be able to identify the EMD cutoff threshold for each column automatically (and not by choosing a global EMD threshold or value k), and clearly this means that we have to resort to heuristics.

We observe an interesting property that holds for most columns, in all test cases that we have examined. Given a column C and all pairwise EMD values, it appears that there usually exists a big gap in the magnitude of EMD values, in the sorted order. For example, in both Figures 4(a) and 4(b) we observe a big gap in the sorted EMD order, after the cutoff region (the dotted red lines in the plots). Intuitively, the gap signifies a cutoff point below which other columns seem to have similar distributions with C, and above which columns seem to be completely unrelated to C (recall that small EMD means *similar* and large EMD means *different*). This is expected for most column types in realistic scenarios. For example, numerical columns are more similar to each other than string columns (hence, a big gap exists in the EMD values where numerical columns end and string columns begin); even among numerical columns, smaller gaps occur because of the different ranges of values in the columns (e.g., salaries vs. zip codes). Conservatively choosing this gap to be the cutoff EMD threshold that defines the neighborhood $\mathbf{N}_C$ guarantees that most false cluster splits are avoided in practice (in other words we do not end up with too many clusters). In addition, we can also use a conservative global EMD cutoff threshold θ (a value large enough to signal that two distributions are significantly different) to make sure that the opposite problem does not occur either, i.e., we do not end up with too few distribution clusters.

Algorithmically identifying the cutoff EMD threshold for a column C is straightforward. Let θ be the global threshold, and $L(C)$ be the sorted list of EMD values for C, i.e., $L(C)$ contains all values $e = EMD(C, C'), \forall C' \in \mathbf{C}$, in increasing order. We truncate $L(C)$ to values smaller than θ and identify the largest difference between two consecutive EMD values in the truncated list. The pseudo code appears in Algorithm 1. In the algorithm we also add θ in the list of EMD values to capture the special case were the largest gap between two values happens to involve θ.

Algorithm 1 ComputeCutoffThreshold $(L(C), \theta)$

```
1: L(C): list of EMD/column-name pairs (e, C'), e = EMD(C, C')
2: L = L ∪ (θ, ∅)
3: Sort L in increasing order of EMD values
4: φ_C = 0, i = 0, gap = 0
5: while L[i + 1] ≤ θ do
6:    if gap < L[i + 1].e − L[i].e then
7:       gap = L[i + 1].e − L[i].e
8:       φ_C = L[i].e
9:    i = i + 1
10: Return φ_C
```

Once the cutoff value ϕ_C for each column C has been computed we can define the neighborhood of a column as follows:

DEFINITION 4 (NEIGHBORHOOD). *The neighborhood $\mathbf{N}_C$ of column C consists of all columns C' with $EMD(C, C') \leq \phi_C$.*

Then, we build the *distribution graph*, which is defined as follows:

DEFINITION 5 (DISTRIBUTION GRAPH). *A* Distribution Graph *$G_D = (V_D, E_D)$ is an undirected graph where each column $C \in \mathbf{C}$ corresponds to a node $C \in V_D$, and an edge between nodes C and C' exists iff $C \in \mathbf{N}_{C'} \vee C' \in \mathbf{N}_C$.*

Alternatively, we can define the edges as $C \in \mathbf{N}_{C'} \wedge C' \in \mathbf{N}_C$, but our experimental evaluation shows that in practice this does not affect precision.

The distribution clusters are obtained by computing the connected components in the resulting distribution graph:

Algorithm 2 ComputeDistributionClusters $(\mathbf{C}, \theta)$

```
1: G_D = ∅
2: A is a hash table of lists of EMD/column-name pairs (e, C)
3: for i ← 1 to |C| do
4:    for j ← i + 1 to |C| do
5:       e = EMD(C_i, C_j)
6:       A[C_i].insert(e, C_j)
7:       A[C_j].insert(e, C_i)
8:    G_D.AddNode(C_i)
9: for i ← 1 to |C| do
10:   φ_{C_i} = ComputeCutoffThreshold(A[C_i], θ)
11:   for all C_j ∈ N_{C_i} do
12:      G_D.AddEdge(C_i, C_j)
13: Return connected components of G_D
```

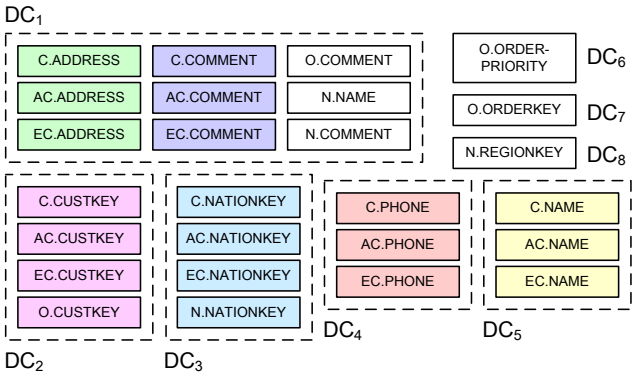


Figure 5: Distribution clusters of TPC-H example.

DEFINITION 6 (DISTRIBUTION CLUSTER). *Let $G_i = (V_D^i, E_D^i), V_D^i \subset V_D, E_D^i \subset E_D, 1 \leq i \leq n$ be the set of n connected components in distribution graph G_D. The set of columns corresponding to the nodes in V_D^i determines distribution cluster $\mathbf{DC}_i$.*

The pseudo code for computing distribution clusters is shown in Algorithm 2. We can compute the connected components of graph G_D (line 13 of Algorithm 2) using either depth-first or breadth-first search.

Figure 5 shows the distribution clusters of the TPC-H example in Figure 2, which contains three base tables and two materialized views (table names EUROPEAN CUSTOMER and ASIAN CUSTOMER are shortened to EC and AC). Using the distribution graph, twenty six columns are partitioned into eight clusters. Columns from distinctly different domains are immediately separated (e.g., numeric values and strings). The numeric columns with different ranges of values are also correctly clustered (e.g., key columns like CUSTKEY, NATIONKEY, ORDERKEY and REGIONKEY), as well as columns that contain specially formatted values (e.g., PHONE which contains numerals and dashes). However, distribution clusters cannot always differentiate between different string columns (e.g., ADDRESS and COMMENT).

3.2 Phase two: Computing Attributes

We now describe in detail how to further decompose distribution clusters into attributes, specifically for identifying columns that have very similar distributions overall but do not belong to the same attributes, as is the case for many string columns. We use an intersection distribution metric and witness columns, to construct one attribute graph per distribution cluster and then correlation clustering to decompose the cluster into attributes.

3.2.1 The Attribute Graph

In order to decompose clusters into attributes we create one *attribute graph* G_A per cluster. Given that all columns within the same distribution cluster have similar distributions of values, we need to differentiate between attributes by also taking into account the values these columns have in common. Clearly, columns with large intersection of values are highly related and should belong to the same attribute (this is similar to automatically identifying whether two columns have a primary/foreign key relationship, as in [17]). On the other hand, columns that have very few or no values in common, might or might not belong to the same attribute (e.g., as is the case of EUROPEAN CUSTOMER and ASIAN CUSTOMER, and conversely, ADDRESS and COMMENT).

We make here the following key observation. We can determine whether two columns with empty intersection come from the same attribute by using a *witness* column, i.e., a third column that is highly related to both columns. In other words, we introduce *relationships by indirect association*. For example, we know for a fact that both EUROPEAN CUSTOMER and ASIAN CUSTOMER have a large number of values in common with CUSTOMER, but not with each other. After identifying that CUSTOMER is related with EUROPEAN CUSTOMER and ASIAN CUSTOMER, we can deduce with high confidence that EUROPEAN CUSTOMER and ASIAN CUSTOMER are also related. Formally:

DEFINITION 7 (WITNESS COLUMN). *Consider three distinct columns C, C', C''. Column C'' is a witness for C and C' if and only if both conditions hold:*

1. *C'' and C are in the same attribute.*
2. *C'' and C' are in the same attribute.*

Clearly, if two columns belong to the same attribute, have no values in common, and no witness column, then we will not be able to identify these columns. This is one more limitation of our approach, but in practice, such cases might either be identifiable using orthogonal techniques (e.g., column name similarity), or in other cases might be hard to identify using any unsupervised solution.

Based on these observations, we create the *attribute graph* of each cluster $\mathbf{DC}$, similarly to the distribution graph of $\mathbf{C}$. Here, a node corresponds to a column of $\mathbf{DC}$ and an edge corresponds to an intersection relationship between two columns.

We also have to define a measure over these edges, which we call Intersection EMD. Intersection EMD measures the likelihood that a pair of columns are part of the same attribute, taking into account the distribution of the values within each column with respect to the *common* values. In general, for an edge (C, C'), $EMD(C, C \cap C') \neq EMD(C', C \cap C')$. Since the edge is undirected, we define its weight as the arithmetic mean of these two values. Formally:

DEFINITION 8 (INTERSECTION EMD).
Given columns C, C', the Intersection EMD *is defined as:*

$$EMD_{\cap}(C, C') = \frac{1}{2}(EMD(C, C \cap C') + EMD(C', C \cap C')).$$

Let $EMD_{\cap}(C, C') = \infty$, if $C \cap C' = \emptyset$.

Clearly, Intersection EMD can differentiate between columns like ADDRESS and COMMENT, since their intersection is empty. Even if they did have a small number of values in common, their Intersection EMD would be very large.

Similar to the case of computing distribution clusters, we need to decide whether the Intersection EMD between two clusters is small enough to place the columns into the same attribute. We are trying to balance the number of attributes to create (small thresholds will result in too many attributes and large thresholds in too few). For each individual column, we compute a cutoff threshold as before (see Algorithm 1), but instead of using EMD we use Intersection EMD. Similarly, we define the neighborhood $\mathbf{N}_C$ of a column C, this time with respect to Intersection EMD.

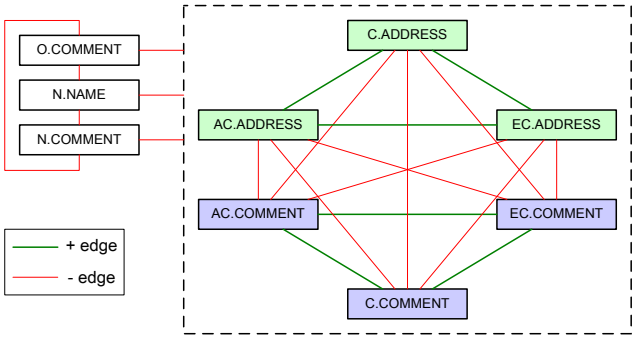


Figure 6: A possible attribute graph of distribution cluster $\mathbf{DC}_1$.

We now give the formal definition of the attribute graph corresponding to a distribution cluster:

DEFINITION 9 (ATTRIBUTE GRAPH).
The attribute graph $G_A = (V_A, E_A)$ of a distribution cluster $\mathbf{DC}$ *is a* complete graph *over the set of vertices of* $\mathbf{DC}$*, such that the weights of edges in E_A are either 1 (positive edges) or -1 (negative edges). Let E_A^+, E_A^- denote the set of positive, resp. negative, edges in G_A. To define them, consider an arbitrary pair of vertices $C, C' \in V_A$.*

1. *Neighborhood: If $C \in \mathbf{N}_{C'} \vee C' \in \mathbf{N}_C$, then $e_{CC'} \in E_{A1}^+$.*
2. *Witness: If $\exists C'' \in V_A$ s.t. $e_{CC''} \in E_{A1}^+ \wedge e_{C'C''} \in E_{A1}^+$, then $e_{CC'} \in E_{A2}^+$.*

We define $E_A^+ = E_{A1}^+ \cup E_{A2}^+$, and $E_A^- = E_A \setminus E_A^+$.

Figure 6 shows the attribute graph of distribution cluster $\mathbf{DC}_1$ from Figure 5. The green lines in the figure denote positive edges while the red lines are negative edges. The edges between the three nodes outside the dashed box to all other nodes are negative. Using Intersection EMD we are able to separate columns like ADDRESS and COMMENT, while AC.ADDRESS and EC.ADDRESS are connected through the witness column C.ADDRESS. The same holds for AC.COMMENT and EC.COMMENT.

The next step is to decompose the graph into attributes. Clearly, we could decompose the graph into connected components (by simply ignoring negative edges) similarly to phase one. Nevertheless, due to the nature of Intersection EMD and the fact that after phase one, attribute graphs consist of a small number of nodes, in practice attribute graphs tend to comprise of a single (or very few) connected components. A better approach here is to use the negative weights productively to find an optimal clustering of nodes into attributes that minimizes the number of conflicting nodes that end up into the same cluster and the number of related nodes that end up in different clusters. As it turns out, this is exactly the goal of *correlation clustering*.

3.2.2 Correlation Clustering

Let $G = (V, E)$ be an undirected graph with edge weights 1 or -1. Let E^+ be the set of positive edges, and E^- be the set of

negative edges; $E = E^+ \cup E^-$. Intuitively, edge $e_{uv} \in E^+$ if u and v are similar; and $e_{uv} \in E^-$ if u and v are dissimilar. The correlation clustering problem [3] on G is defined as follows:

DEFINITION 10 (CORRELATION CLUSTERING). *Compute disjoint clusters covering V, such that the following cost function is minimized:*

$$cost = |\{e_{uv} \in E^+ \mid Cl(u) \neq Cl(v)\}| + |\{e_{uv} \in E^- \mid Cl(u) = Cl(v)\}|,$$

where $Cl(v)$ denotes the cluster node v is assigned to.

This definition minimizes the total number of disagreements, i.e., the number of positive edges whose endpoints are in different clusters, plus the number of negative edges whose endpoints are in the same cluster. Alternatively, one can define the dual problem of maximizing the total agreement. More general versions of the problem exist, e.g., when weights are arbitrary real values. However, this version is sufficient for our purposes.

Correlation clustering can be written as an integer program, as follows. For any pair of vertices u and v, let $X_{uv} = 0$ if $Cl(u) = Cl(v)$, and 1 otherwise. The integer program is

Minimize:

$$\sum_{e_{uv} \in E^+} X_{uv} + \sum_{e_{uv} \in E^-} (1 - X_{uv})$$

s.t.

$$\forall u, v, w : X_{uw} \leq X_{uv} + X_{vw}$$

$$\forall u, v : X_{uv} \in \{0, 1\}$$

The condition $X_{uw} \leq X_{uv} + X_{vw}$ ensures that the following transitivity property is satisfied: if $X_{uv} = 0$ and $X_{vw} = 0$, then $X_{uw} = 0$ (note that this is equivalent to: if $Cl(u) = Cl(v)$ and $Cl(v) = Cl(w)$, then $Cl(u) = Cl(w)$). Therefore X defines an equivalence relationship, and the clusters are its equivalence classes. Correlation clustering is NP-Complete [3]. Nevertheless, the above integer program can be solved exactly by IP solvers (e.g., CPLEX [10]) for sufficiently small graphs. For larger graphs, one can use polynomial time approximation algorithms [2].

Going back to the example of Figure 6, correlation clustering on this graph will further decompose nine columns into five attributes, as shown in Figure 7. If all the edges in the attribute graph are correctly labeled, such as the simple example in Figure 6, then the graph results in perfect clustering, meaning that there are no disagreements. (When this is the case, simply removing all the negative edges and computing the connected components in the remaining graph also returns the correct attributes.) However, if a few edges are assigned conflicting labels, there is no perfect clustering. For example, Figure 8 shows a different attribute graph for distribution cluster DC_1, obtained by setting a higher threshold θ in Algorithm 1. The edge between AC.ADDRESS and AC.COMMENT is now labeled positive. In addition, two other positive edges are created, since AC.ADDRESS and AC.COMMENT act as witnesses for C.ADDRESS and C.COMMENT. As it turns out, in this case correlation clustering is still able to separate the columns correctly, by finding a partition that agrees as much as possible with the edge labels. Of course, in some cases correlation clustering will result in mistakes, but in the end our solution will decompose the graph into attributes that can be manually inspected much more easily than having to look at the complete distribution graph.

We now summarize phase two. The pseudo code is shown in Algorithm 3. For each distribution cluster computed in phase one, compute the Intersection EMD between each pair of columns in the

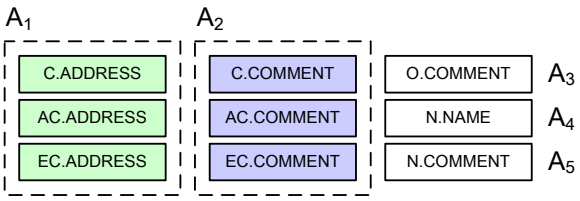


Figure 7: Attributes discovered in the attribute graph of distribution cluster $\mathbf{DC}_1$.

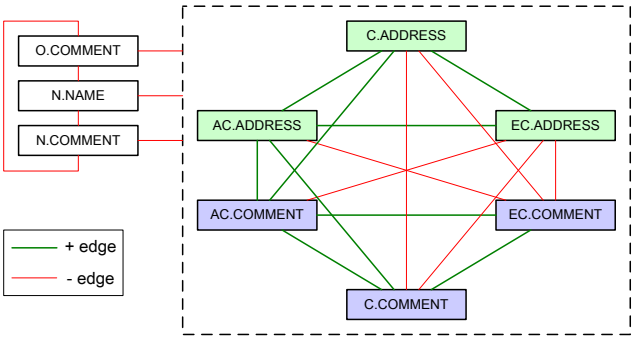


Figure 8: Another possible attribute graph of distribution cluster $\mathbf{DC}_1$.

cluster and store the resulting values in a hash table I in increasing order of Intersection EMD. Then compute the cutoff threshold for each column. Construct the attribute graph G_A according to Definition 9. Creating positive edges based on witness columns is implemented by creating positive edges between nodes with path of length two. This is accomplished by first computing the adjacency matrix E of graph $G_{A1} = (V_A, E^+_{A1})$, where $E[C_i][C_j] = 1$ means the edge between node C_i and C_j is positive. The adjacency matrix of graph $G_A = (V_A, E^+_{A2})$ can be computed by multiplying E by itself. The sum of the two matrices is the final adjacency matrix M of attribute graph $G_A = (V_A, E_A)$. Finally, we compute attributes using correlation clustering on graph G_A.

Algorithm 3 ComputeAttributes ($\mathbf{DC}$, θ)

```
1: G_A = ∅, E[][] = 0, M[][] = 0
2: I is a hash table of lists of Intersection-EMD/column-name
   pairs (e, C)
3: for i ← 1 to |DC| do
4:    for j ← i + 1 to |DC| do
5:       e = EMD∩(C_i, C_j)
6:       I[C_i].insert(e, C_j)
7:       I[C_j].insert(e, C_i)
8:    φ_{C_i} = ComputeCutoffThreshold(I[C_i], θ)
9:    for all C_j ∈ N_{C_i} do
10:      E[C_i][C_j] = 1
11:   G_A.AddNode(C_i)
12: M = E + E × E
13: for i ← 1 to |DC| do
14:    for j ← 1 to |DC| do
15:       if M[i][j] == 0 then
16:          G_A.AddNegativeEdge(C_i, C_j)
17:       else
18:          G_A.AddPositiveEdge(C_i, C_j)
19: Return correlation clustering of G_A
```

4. PERFORMANCE CONSIDERATIONS

Clearly, due to the difficult and subjective nature of this problem, no unsupervised solution will lead to 100% precision 100% of the time. The solution provided here can be used in conjunction with other techniques for improving quality. Notice that the two phases of our algorithm are quite similar. We create a graph based on some similarity measure and decompose the graph based on connected components or correlation clustering. The heuristic nature of the algorithm raises a number of questions about possible alternative strategies. For example, we could reverse the two steps, or use Intersection EMD instead of EMD first. We can also use correlation clustering in the first phase of the algorithm.

We use EMD first simply because it is a weaker notion of similarity than Intersection EMD. EMD acts upon all values of two columns, while Intersection EMD acts upon the common values. EMD is used to roughly decompose the instance graph into smaller problems, by separating columns that clearly belong to different attributes; strings from numerical columns and columns with significantly different ranges and distributions of values. The graph based on EMD edges alone (without any Intersection EMD edges) is sparse and easy to partition into smaller instances. Of course, we could combine both phases into one by creating a graph with EMD, Intersection EMD and witness edges, but we use a two phase approach here for efficiency. For the same reason we do not use correlation clustering in the first phase. Running correlation clustering on the distribution graph G_D can be very expensive due to the large number of nodes. On the other hand, running correlation clustering independently on the much smaller connected components is manageable.

Notice that the cost of computing EMD and Intersection EMD depends on the size of the columns involved. Clearly, for columns containing a very large number of distinct values the cost of computing all pairwise EMD and Intersection EMD values can be prohibitive.

For that reason we can approximate both measures by using a technique introduced in [17], which is based on quantiles. Essentially the technique computes a fixed number of quantiles from all columns (e.g., 256 quantiles) and then computes EMD between two columns by using the quantile histograms. In particular, given two columns C and C', $EMD(C, C')$ is computed by taking the quantile histogram of C and performing a linear scan of C' to find the number of values in C' falling into each bucket of the histogram of C (notice that we cannot compute EMD between two quantile histograms directly, since the bucket boundaries might not coincide, in which case EMD is undefined). Then $EMD(C, C')$ is approximated as the EMD between the two resulting histograms. The intuition here is that quantile summaries are a good approximation of the distribution of values in the first place, hence the EMD between two quantile summaries is a good approximation of the EMD between the actual columns. Moreover, this approach has a proven bounded error of approximation, as shown in [17].

Computing Intersection EMD entails computing the intersection between all pairs of columns, which of course is a very expensive operation, especially if no index exists on either column. In order to improve performance we build Bloom filters [4] and use the Bloom filters to compute an approximate intersection between sets. Given two columns C and C', we first perform a linear scan of column C, probe the Bloom filter of C', and if the answer is positive, use the existing quantile histograms of C and C' to (approximately) compute both $EMD(C, C \cap C')$ and $EMD(C', C \cap C')$. Optionally, we can improve the approximation of the intersection by also scanning column C' and probing the Bloom filter of C. Given that Bloom filters introduce false positives, this approach can result in cases where two columns have an empty intersection, but their approximate Intersection EMD is finite. Nevertheless, for columns of very large cardinality (especially in the absence of indexes), using Bloom filters results in significant performance improvement. One can further balance accuracy and performance, by using exact intersection computations for small columns, and Bloom filters for large columns.

As discussed above, correlation clustering is NP-Complete. Nevertheless, we solve it exactly, by running CPLEX [10] on its corresponding integer program. In our experiments, CPLEX was able to find solutions for large graph instances very fast. The largest graph instance we tried contained 170 nodes, 14365 variables, and 2.5 million constraints and took 62 seconds to complete using four Intel IA-64 1.5GHz CPUs and four threads. Alternatively, one can use polynomial time approximation algorithms [2] if faster solutions are required for larger graphs.

5. EXPERIMENTS

We conducted extensive experiments to evaluate our approach on three datasets based on the TPC-H[1] benchmark (with scale factor 1), and the IMDb[2] and DBLP[3] databases. For each dataset, we created a large set of materialized views to simulate a more complex scenario. The detailed statistics of all datasets are given in Table 1. The views are created from a selection of rows based on the values of a specific column (mostly columns that contain categorical data) and each of the views represents a semantically meaningful subset of the data in the base table. Table 2 summarizes the views generated in each dataset. The experiments were run on an Intel Core2 Duo 2.33 GHz CPU Windows XP box and CPLEX was run on a four Intel IA-64 1.5GHz CPU Linux box.

	Base tables	Views	Columns	Rows
TPC-H	8	110	785	12,680,058
IMDb	9	118	254	12,048,155
DBLP	6	66	285	8,647,713

Table 1: Datasets statistics.

We use two standard metrics, *precision* and *recall*, to measure the quality of discovered attributes. The gold standard was manually identified from a careful analysis of each dataset. Given the nature of the problem, we define precision as a measure of purity of a discovered attribute (how similar is the set of columns contained in the attribute with respect to the gold standard), and recall as a measure of completeness. Let the set of discovered attributes be $\mathbf{A} = \{A_1, A_2, \ldots, A_m\}$ and the gold standard be $\mathbf{T} = \{T_1, T_2, \ldots, T_{m'}\}$. We first define the precision and recall of a discovered attribute A_i. Each column in A_i belongs to an attribute in $\mathbf{T}$. Let A_i correspond to T_j if and only if the majority of columns in A_i also belong in T_j. Then, the precision and recall of A_i are defined as:

$$Precision(A_i) = \frac{|A_i \cap T_j|}{|A_i|}, Recall(A_i) = \frac{|A_i \cap T_j|}{|T_j|}.$$

We then define the precision and recall of the final result $\mathbf{A}$ as the average over all attributes:

$$Precision(\mathbf{A}) = \frac{\sum_{i=1}^{m} Precision(A_i)}{m},$$

$$Recall(\mathbf{A}) = \frac{\sum_{i=1}^{m} Recall(A_i)}{m}.$$

[1] http://www.tpc.org/tpch
[2] http://www.imdb.com/interfaces
[3] http://dblp.uni-trier.de/xml/

Dataset	Base Table	View No.	Selection	Description
TPC-H	CUSTOMER	1-2	ACCTBAL	Customers with positive/negative account balance.
		3-7	MKTSEGMENT	Customers in each market segment.
		8-37	NATIONKEY	Customers from each nation/region.
	NATION	38-42	REGIONKEY	Nations in each region.
	PART	43-67	BRAND	Parts of each brand.
		68-72	MFGR	Parts by each manufacture.
	SUPPLIER	73-102	NATIONKEY	Suppliers from each nation/region.
	ORDERS	103-105	ORDERSTATUS	Orders in each status.
		106-110	ORDERPRIORITY	Orders with each priority.
IMDb	MOVIE	1-28	COUNTRY	Movies released in each country.
		29-118	YEAR	Movies released in each year.
DBLP	ARTICLES	1-20	YEAR	Journal papers published in each year.
	INPROCEEDINGS	21-38	YEAR	Conference papers published in each year.
	BOOKS	39-66	YEAR	Books published in each year.

Table 2: Description of materialized views.

5.1 Distribution Similarity

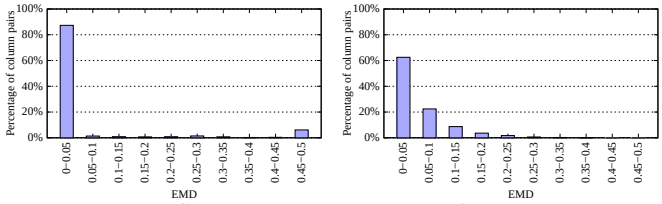
As already discussed, in most cases columns that belong to the same attribute tend to have similar distributions, and columns that have different distributions more likely belong to different attributes. First, we run experiments to verify this intuition. For each dataset, we examine the EMD values between all pairs of columns in the same attribute, based on the gold standard, and plot the distribution histograms (for efficiency we approximate all EMD computations using quantile histograms). Figure 9 shows the results for TPC-H and DBLP. For TPC-H 87.3% EMD values between columns of the same attribute are smaller than 0.05. For DBLP 62.5% are below 0.05 and only 2.8% are above 0.2. This verifies our intuition that EMD is a robust measure for phase one of the algorithm.

Notice that a few pairs of columns in TPC-H have very large EMD. This is caused by the four attributes shown in Table 3. View1 and View2 select customers with positive and negative balance (see Table 2), which results in a horizontal partition of the base table and very different distributions in each partition. The same happens for attributes phone number and order date. Since View8-37 and View73-102 select customers and suppliers from a particular nation/region, the phone numbers in each view start with the same prefixes. View103-105 are the orders in each particular status and order status is correlated to the date when the order is placed. Distribution similarity fails to discover the associations between columns if such correlations exist, and this is a limitation of our approach. However, we can see here that distribution similarity between columns of the same attribute holds for the large majority of columns. After removing the horizontal partitions mentioned above from TPC-H (65 columns in total), the EMD values between all pairs of columns within the same attribute are below 0.2 and for up to 99.5% of the pairs, EMD is below 0.05.

Attribute	Columns
Customer account balance	ACCTBAL in CUSTOMER and View1-2
Customer phone number	PHONE in CUSTOMER and View8-37
Supplier phone number	PHONE in SUPPLIER and View73-102
Order date	DATE in ORDERS and View103-105

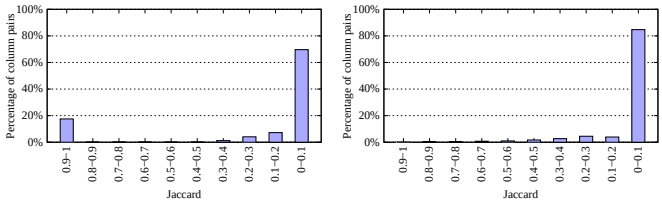
Table 3: Attributes that contain horizontally partitioned columns in TPC-H.

To illustrate the point that columns of the same attribute do not necessarily have too many values in common, in Figure 10 we plot a histogram of the pairwise Jaccard similarity of columns within the same attribute, based on the golden standard. Recall that a high



(a) TPC-H (b) DBLP

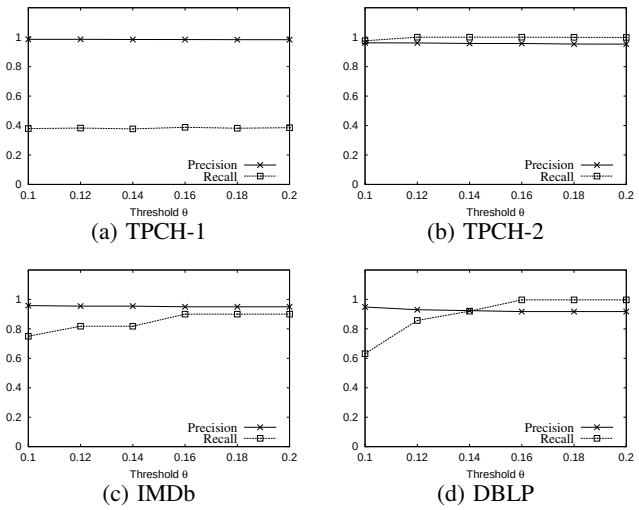
Figure 9: Distribution histograms of EMD values between all pairs of columns in the same attribute for TPC-H and DBLP.



(a) TPC-H (b) DBLP

Figure 10: Distribution histograms of Jaccard values between all pairs of columns in the same attribute for TPC-H and DBLP.

Jaccard value implies a large number of common values and vice versa. We observe that for TPC-H 70% of column pairs have Jaccard similarity smaller than 0.1, and only 19% have Jaccard above 0.9. The results for DBLP are even more striking, with more than 80% of column pairs having Jaccard similarity smaller than 0.1. It is thus clear that Jaccard similarity is a poor measure for clustering columns into attributes. In particular, a naive approach for discovering attributes would be to create a column similarity graph with edges weighted according to pairwise Jaccard similarity, then remove edges with Jaccard similarity smaller than some threshold, and finally compute the connected components. Figure 10 clearly shows that dropping edges with reasonably small Jaccard similarity would result in a very sparse graph, separating columns into atomic attributes. On the other hand, retaining all edges would tend to separate columns into two attributes, one for numerical attributes and one for strings.



(a) TPCH-1 (b) TPCH-2

(c) IMDb (d) DBLP

Figure 11: Accuracy results on TPC-H, IMDb and DBLP for varying thresholds θ.

5.2 Attribute Discovery

Here we measure the accuracy of our technique for discovering attributes. We use a single global threshold θ for computing the distribution clusters in Phase one as well as building the attribute graph in Phase two. Furthermore, we use Bloom filter for all columns, across the board, to approximate Intersection EMD. In the experiments, we vary θ from 0.1 to 0.2. Table 4 shows the accuracy results for all datasets. For TPC-H, we report two sets of results. TPCH-1 refers to the results with respect to the original dataset. TPCH-2 refers to the results with respect to a reduced dataset, obtained by removing the horizontally partitioned columns. For readability, we also plot the precision and recall in Figure 11. We can see that for large ranges of thresholds θ (0.16-0.2) we achieve high precision and recall for all datasets, which makes our approach easy to use in practice.

Threshold θ		0.1	0.12	0.14	0.16	0.18	0.2
TPCH-1	m'	46					
	m	108	107	106	103	102	101
	P	0.986	0.985	0.984	0.984	0.983	0.983
	R	0.379	0.383	0.377	0.388	0.382	0.385
TPCH-2	m'	45					
	m	41	40	39	39	38	38
	P	0.962	0.961	0.957	0.957	0.953	0.953
	R	0.976	1	1	1	0.999	0.998
IMDb	m'	10					
	m	12	11	11	10	10	10
	P	0.958	0.955	0.955	0.95	0.95	0.95
	R	0.75	0.818	0.818	0.9	0.9	0.9
DBLP	m'	14					
	m	19	14	13	12	12	12
	P	0.949	0.93	0.924	0.918	0.918	0.918
	R	0.632	0.857	0.92	0.997	0.997	0.997

Table 4: Accuracy results on TPC-H, IMDb and DBLP for different thresholds θ; m' true number of attributes; m attributes in our solution; P is precision; R is recall.

For the TPC-H dataset, as already explained, 65 columns (belonging to only 4 attributes out of 45) are from horizontal partitions of the base tables due to indirect correlations between columns. Here, distribution similarity fails to cluster such columns together, more precisely each view becomes its own cluster, resulting in more than 100 attributes overall. However, as shown in TPCH-2, by drilling down we can see that our approach achieves high precision and recall for discovering attributes in the remaining set of columns. It should be noted here that columns that form unit clusters can be singled out and treated separately during post-processing. For future work, we are investigating whether it is possible to identify if unit clusters belonging to horizontally partitioned columns can be concatenated as a subsequent step.

Our approach discovers fewer attributes than the gold standard. Table 5 shows the attributes that cause the errors for $\theta = 0.12$. As shown, nine distinct attributes are clustered into four attributes by our algorithm (one attribute per row), and that accounts for the five missing attributes. Here, 9997 out of 10000 values in SUPPLIER.ADDRESS are identical to the values in CUSTOMER.ADDRESS (due to the way addresses are generated in TPC-H), making these two columns indistinguishable; DISCOUNT contains all values in TAX; the same is true for SUPPKEY and AVAILQTY; 12.8% of values in SUPPLIER.ACCTBAL appear in CUSTOMER.ACCTBAL. Finally, LINEITEM.COMMENT has no intersection with ADDRESS, but the false clustering here occurs due to false positives produced by the Bloom filter in phase two.

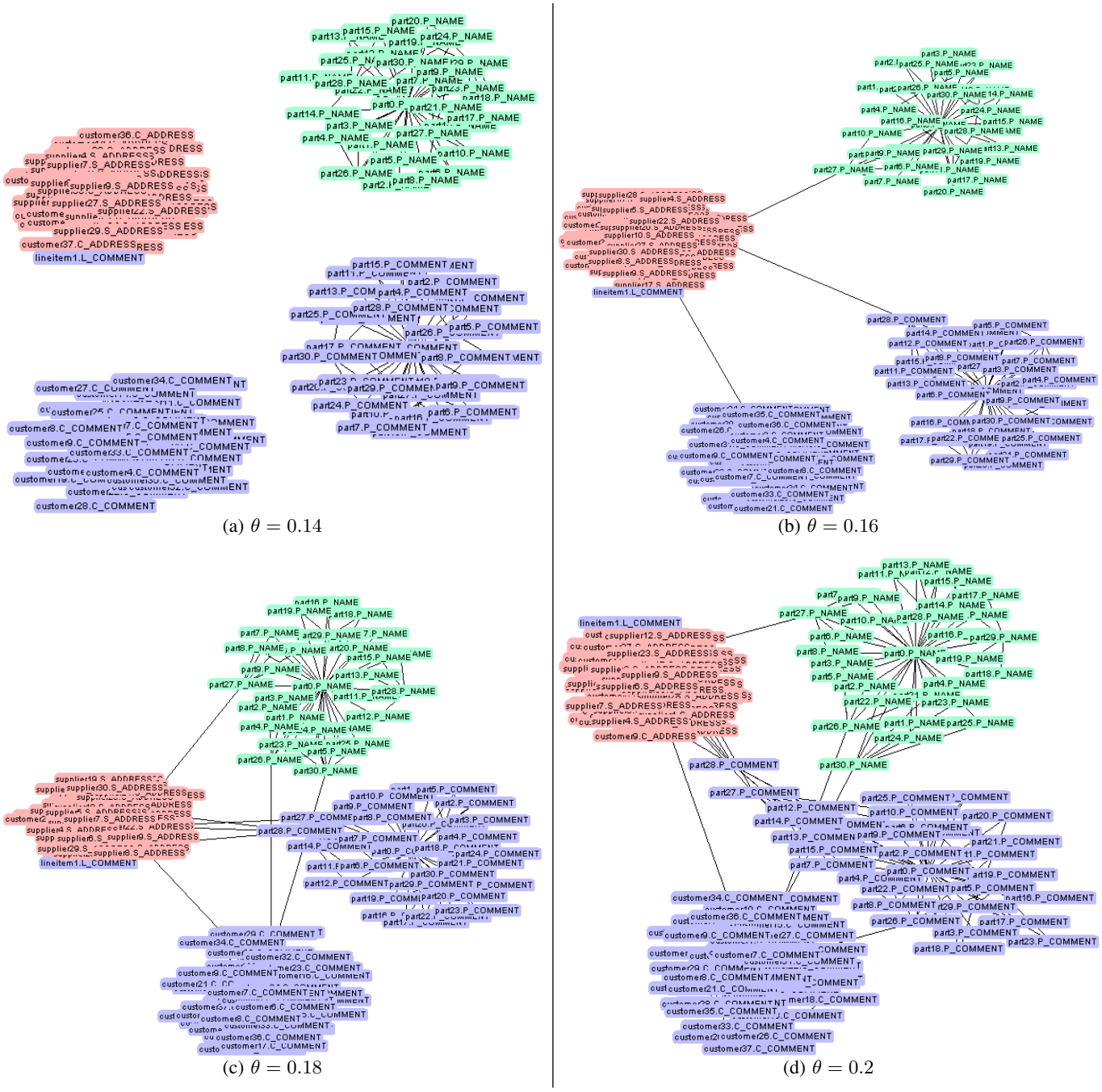
Correlation clustering proves to be a robust way of separating attributes in the attribute graph. Figure 12 shows an attribute subgraph of TPC-H, for varying θ from 0.14 to 0.2 (negative edges are removed for readability). For $\theta = 0.14$, the four clusters are totally disconnected (there are no positive edges between different attributes). Correlation clustering (or even connected components) in this case would separate the graph into four attributes. Our method is able to separate CUSTOMER.COMMENT views from PART.COMMENT views, while, for example, the method of comparing column names will fail in this case. On the other hand, CUSTOMER.ADDRESS and SUPPLIER.ADDRESS are clustered together, but this is clearly because 9997 out of 10000 addresses are the same. As θ increases, the number of positive edges across attributes increases as well. This is evident in Figure 12(d). However, after running correlation clustering we are still able to separate the graph into four attributes once again, with very few errors. For $\theta = 0.16$, the result is exactly the same as for $\theta = 0.14$. For $\theta = 0.18$, correlation clustering will place one view of PART.NAME in the attribute of CUSTOMER.COMMENT. For $\theta = 0.2$, one view of PART.COMMENT will be placed in the attribute for ADDRESS.

For IMDb we achieve 0.95 precision and 0.9 recall for θ ranging from 0.16 to 0.2. In our result, ACTOR.NAME and DIRECTOR.NAME are clustered together due to very large overlap of values. Since most directors are also actors, in this case choosing whether directors and actors should be treated differently depends on application context. In this case of course, a simple solution based on column names can provide an answer. Another problem is column MEXICO.MOVIENAME which is not included in the movie names attribute. Some simple data analysis here shows that 14.0% of movie names in MEXICO.MOVIENAME start with *la/las/los* and 11.5% names start with *el*, making the distribution of this column significantly different from movie names in other views. Decreasing θ to 0.14 and 0.12 results in splitting SPAIN.MOVIENAME out as well, for the same reason. When using threshold $\theta = 0.1$, HONGKONG.MOVIENAME and TAIWAN.MOVIENAME are also separated. This is not surprising, since both mainly contain names in Chinese and have small overlap with the movie names in other views.

Finally, for the DBLP dataset we also achieve precision above 0.9 and recall above 0.997, for θ ranging from 0.16 to 0.2. The errors occur in four attributes. Here, AUTHOR.NAME and EDITOR.NAME are clustered together given that 596 out of 621 editors

1	DISCOUNT	TAX	
2	SUPPKEY	AVAILQTY	
3	CUSTOMER.ACCTBAL	SUPPLIER.ACCTBAL	
4	CUSTOMER.ADDRESS	SUPPLIER.ADDRESS	LINEITEM.COMMENT

Table 5: Attributes that are incorrectly clustered together in TPC-H for $\theta = 0.12$.



(a) $\theta = 0.14$ (b) $\theta = 0.16$

(c) $\theta = 0.18$ (d) $\theta = 0.2$

Figure 12: An attribute sub-graph of TPC-H for varying thresholds θ.

appear in the AUTHOR table as well. The same is true for ARTICLES.TITLE and INPROCEEDINGS.TITLE, since it seems that the majority of papers submitted to journal publications have the exact same title as the conference versions of the papers.

Overall, clearly it is in some cases difficult even for humans to decide what constitutes an attribute, without additional application dependent context. Our technique is able to separate major attributes very well, and make only minor mistakes that can either be corrected by supervision and simple statistical analysis, or by using orthogonal approaches (e.g., column name matching, if meaningful schema information exists).

6. RELATED WORK

The work of Zhang et al. [17] was the main impetus behind our approach, since they introduced the usage of EMD for correlating columns based on the distribution of their values. Zhang et al. introduced the quantile approximation of EMD that we also use here and laid the foundation of using EMD to express primary/foreign key relationships that are one of our main relationship types. Nevertheless, here we extend this work by building distribution and attribute graphs based on EMD and Intersection EMD, which is a novel statistical measure of the relationship between two columns.

In the context of relational databases there has been little work that concentrates on classifying columns into attributes. The only previous work that we are aware of is by Ahmadi et al. [1], that utilizes q-gram based signatures to capture column type information based on formatting characteristics of data values (for example the presence of '@' in email addresses). The technique builds signatures based on the most popular q-grams within each column and clusters columns according to basic data types, like email addresses, telephone numbers, etc. In that respect, the goal of this work is orthogonal to ours: It tries to categorize columns into generic data types. Our work takes into account the distribution of values, the distribution with respect to common values, and indirect associations through witnesses, as opposed to only using the distribution of q-grams.

Related work from the field of schema matching has concentrated on three major themes. The first is semantic schema matching that uses information provided only by the schema and not from particular data instances. The second is syntactic schema matching that uses the actual data instances. The third uses external information, like thesauri, standard schemas, and past mappings. Most solutions use hybrid approaches that cover all three themes. Our work falls purely under the second category. Existing data driven approaches have not used any distributional information to discover relationships between columns, apart from simple statistics. Current approaches use string-based comparisons (prefix/suffix tests, edit distance, etc.), value ranges, min/max similarity, and mutual information based on q-gram distributions [11, 7, 8, 12, 13]. Rahm and Bernstein [15] conducted a survey on schema matching techniques.

Previous work tangentially related to discovering attributes is that on discovering columns that contain similar values. A number of statistical summaries have been developed for that purpose, including min-hash signatures [5], and locality sensitive hashing [9]. These techniques cannot be used for discovering attributes, since they only capture intersection relationships between columns and no distributional information.

7. CONCLUSION

We argued that discovering attributes in relational databases is an important step in better understanding and working with the data. Toward this goal, we proposed an efficient solution, based on statistical measures between pairs of columns, to identify such attributes given a database instance. Our solution was able to correctly identify attributes in real and synthetic databases with very high accuracy. In the future, we plan to investigate whether the techniques proposed here can be extended to discover multi-column attributes (for example when customer names are expressed as separate first/last name columns) and also explore whether information theoretic techniques can be used to solve the problem of horizontally partitioned attributes (like telephone numbers based on locations).

8. ACKNOWLEDGEMENT

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

9. REFERENCES

[1] B. Ahmadi, M. Hadjieleftheriou, T. Seidl, D. Srivastava, and S. Venkatasubramanian. Type-based categorization of relational attributes. In *EDBT*, pages 84–95, 2009.

[2] N. Ailon, M. Charikar, and A. Newman. Aggregating inconsistent information: Ranking and clustering. *Journal of the ACM*, 55(5), 2008.

[3] N. Bansal, A. Blum, and S. Chawla. Correlation clustering. *Machine Learning*, 56:89–113, June 2004.

[4] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *CACM*, 13(7):422–426, 1970.

[5] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. *Journal of Computer and System Sciences*, 60(3):630–659, 2000.

[6] S. Chaudhuri, V. Ganti, and R. Kaushik. A primitive operator for similarity joins in data cleaning. In *ICDE*, page 5, 2006.

[7] R. Dhamankar, Y. Lee, A. Doan, A. Y. Halevy, and P. Domingos. iMAP: Discovering complex mappings between database schemas. In *SIGMOD*, pages 383–394, 2004.

[8] H. H. Do and E. Rahm. COMA - a system for flexible combination of schema matching approaches. In *VLDB*, pages 610–621, 2002.

[9] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *VLDB*, pages 518–529, 1999.

[10] IBM. CPLEX optimizer. http://www.ibm.com/software/integration/optimization/cplex-optimizer.

[11] J. Kang and J. F. Naughton. On schema matching with opaque column names and data values. In *SIGMOD*, pages 205–216, 2003.

[12] J. Madhavan, P. A. Bernstein, and E. Rahm. Generic schema matching with cupid. In *VLDB*, pages 49–58, 2001.

[13] S. Melnik, E. Rahm, and P. A. Bernstein. Rondo: A programming platform for generic model management. In *SIGMOD*, pages 193–204, 2003.

[14] T. Milo and S. Zohar. Using schema matching to simplify heterogeneous data translation. In *VLDB*, pages 122–133, 1998.

[15] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350, 2001.

[16] T. P. P. C. (TPC). TPC benchmarks. http://www.tpc.org/.

[17] M. Zhang, M. Hadjieleftheriou, B. C. Ooi, C. M. Procopiuc, and D. Srivastava. On multi-column foreign key discovery. *PVLDB*, 3(1):805–814, 2010.