On Schema Matching with Opaque Column Names and Data Values

Jaewoo Kang Jeffrey F. Naughton

Department of Computer Sciences
University of Wisconsin-Madison
1210 West Dayton Street
Madison, WI 53706, USA

{jaewoo, naughton}@cs.wisc.edu

ABSTRACT

Most previous solutions to the schema matching problem rely in some fashion upon identifying "similar" column names in the schemas to be matched, or by recognizing common domains in the data stored in the schemas. While each of these approaches is valuable in many cases, they are not infallible, and there exist instances of the schema matching problem for which they do not even apply. Such problem instances typically arise when the column names in the schemas and the data in the columns are "opaque" or very difficult to interpret. In this paper we propose a two-step technique that works even in the presence of opaque column names and data values. In the first step, we measure the pair-wise attribute correlations in the tables to be matched and construct a dependency graph using mutual information as a measure of the dependency between attributes. In the second stage, we find matching node pairs in the dependency graphs by running a graph matching algorithm. We validate our approach with an experimental study, the results of which suggest that such an approach can be a useful addition to a set of (semi) automatic schema matching techniques.

1. INTRODUCTION

The schema matching problem at the most basic level refers to the problem of mapping schema elements (for example, columns in a relational database schema) in one information repository to corresponding elements in a second repository. While schema matching has always been a problematic and interesting aspect of information integration, the problem is exacerbated as the number of information sources to be integrated, and hence the number of integration problems that must be solved, grows. Such schema matching problems arise both in "classical" scenarios such as company mergers, and in "new" scenarios such as the integration of diverse sets of queryable information sources over the web.

Purely manual solutions to the schema matching problem are too labor intensive to be scalable; as a result, there has been a great deal of research into automated techniques that can speed this process by either automatically discovering good mappings, or by proposing likely matches that are then verified by some human expert. In this paper we present such an automated technique that is designed to be of assistance in the particularly difficult cases in which the column names and data values are "opaque," and/or cases in which the column names are opaque and the data values in multiple columns are drawn from the same domain. Our approach works by computing the "mutual information" between pairs of columns within each schema, and then using this statistical characterization of pairs of columns in one schema to propose matching pairs of columns in the other schema.

To clarify our aims and provide some context, consider a classical schema mapping problem that arises in a corporate merger. To complete the merger, we have to integrate the databases of the two companies. How should we determine which attributes in one company's tables should be mapped to which attributes in the other's tables? First, one logical approach is to compare attribute names across the tables. Some of the attribute names will be clear candidates for matching, due to common names or common parts of names. Using the classification given in [20], such an approach is an example of *schema-based matching* [4][18]. However, for many columns schema-based matching will not be effective, because different institutions may use different terms or encodings for semantically identical attributes, or use similar names for semantically different attributes.

When schema-based matching fails, the next logical approach is to look at the data values stored in the schemas. Again referring to the classification from [20], this approach is called *instance-based matching* [6][12][13]. Instance-based matching also will work in many cases. For example, if we are deciding whether to match *Dept* in one schema to either *DeptName* or *DeptID* in the other, by looking at the column instances one may easily find the mapping, because *DeptName* and *DeptID* are likely to be drawn from different domains, e.g., names and alpha-numeric codes. Unfortunately, however, instance-based matching is also not always successful.

When instance-based mapping fails, it is often because of its inability to distinguish different columns over the same data domain and, similarly, its inability to find matching columns over different encodings of logically similar domains. For example, *EmployeeID* and *CustomerID* columns in a table are unlikely to be distinguished if both the columns are of numeric data types and the ranges of the *ID*s are identical. Similarly, if one company uses numeric values for the *EmployeeID* while the other company uses a formatted text for what is logically the same column, the

SIGMOD 2003, June 9-12, 2003, San Diego, CA.

Model	Color	Tire
XLE	White	P2R6
XG2.5	Silver	XR5
LE	Red	GM6

A	B	C
GL3.5	b1	c1
XGL	b2	c2
XE	b3	c3

Figure 1. Two tables from car part databases

traditional instance-based approach will fail to identify the correspondence between the two *EmployeeID* columns.

The technique we propose in this paper is also an instance-based technique. However, it applies in cases where previously proposed techniques do not apply because 1) it does not rely on any interpretation of data values, and 2) it considers correlations among the columns in each table. We emphasize that our claim is not that our technique dominates previously proposed techniques (it doesn't); rather, since it applies where previous techniques do not apply, it is a useful addition to a suite of automated schema mapping tools.

To gain insight into our approach, consider the example tables in Figure 1. Suppose these tables are from two automobile plants in different companies or divisions of a large company. Imagine that the column names of the second table and data instances in columns *B* and *C* are some plant specific codes that are incomprehensible to the schema matching tools. Conventional instance-based matchers may find correspondence between the columns *Model* and *A* due to their syntactic similarity. However, no further matches are likely to be found because the two columns *B* and *C* cannot be interpreted and they share exactly same statistical characteristics; that is, they have the same number of unique values, similar distributions, and so forth.

To make progress in such a difficult situation, our technique exploits dependency relationships between the attributes in each table. For instance, in the first table in Figure 1, there will exist some degree of dependency between *Model* and *Tire* if model partially determines the kinds of tires a car can use. On the other hand, perhaps *Model* and *Color* are likely to have very little interdependency. If we can measure the dependency between columns *A* and *B* and columns *A* and *C*, and compare them with the dependency measured from the first table, it may be possible to find the remaining correspondences.

As we can see, an advantage of using dependency relations in schema matching is that this approach does not require data interpretation; that is, even if the data sets in the schemas to be matched use different encodings, we can still measure the dependency relations. As a result, our proposed matching technique can be applied to multiple unrelated domains without retraining or customization. We refer to matching techniques that are not dependent of data interpretation as *un-interpreted matching*, and make this precise in the next definition.

Definition 1.1 Interpreted vs. Un-Interpreted Matching *Let* $M_1 = match(R(r_1, r_2, .., r_n), S(s_1, s_2, .., s_m))$ *and* $M_2 = match(R(r_1, r_2, .., r_n), S(f_1(s_1), f_2(s_2), .., f_m(s_m))$ *where* M_i *is a match result, match is a schema matching algorithm, R is a source schema of size n, S is a target schema of size m, and finally* f_i *is an arbitrary one-to-one function applied to the values of column i in the target schema. We call the given matching algorithm, match, an* un-interpreted matching *if and only if the two match results M1 and M2 are identical regardless of the function* f_i*. Conversely, it is called an* interpreted matching *if the two results are different.*

In the following it will also be useful to have the following definition, which captures the notion of whether the matching algorithm considers data elements in isolation or their relationship to other data elements.

Definition 1.2 Element vs. Structure Matching *Structure Matching algorithms utilize the relationship between columns in a table, while element matching algorithms only consider properties of single columns.*

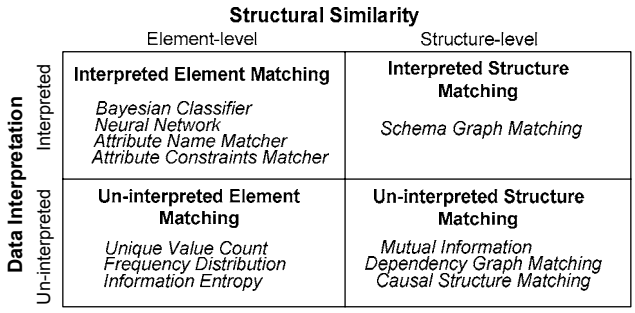


Figure 2. Schema matching technique classification

Figure 2 illustrates classification of schema matching techniques based on the use of data interpretation and structural similarity. While all four classes of techniques are valuable in different domains, we focus in this paper on *un-interpreted structure matching*. We propose a two-step technique that works in the presence of opaque attribute names and values. In the first stage, we measure the pair-wise attribute correlations in the tables to be matched and construct a dependency graph using mutual information [5] (a measure of the dependency between attributes.) In the second stage, we find matching node pairs in the dependency graphs by running a graph matching algorithm. In this paper we are making the following contributions:

- We introduce a new criterion, *data interpretation*, in classifying schema matching techniques. Along with *structural similarity* we classify schema matching techniques into four categories. Using this classification, we identify a new problem class that has not been addressed by existing techniques.
- We introduce a new two-step schema matching technique that takes into account the dependency relations among the attributes.
- We reduce a schema matching problem to a traditional graph matching problem by capturing hidden dependencies between attributes and structuring them as a labeled graph.
- We validate our approach with an experimental study, the results of which suggest that such an approach can be a useful addition to a set of (semi) automatic schema matching techniques. Our experiments also show by exploiting relationships between columns our techniques can do much better than a technique that only considers statistical properties of individual columns.

The rest of the paper is organized as follows: Section 2 describes the two-step un-interpreted structure matching technique. Section 3 validates the framework with an experimental study. Section 4 presents related work. Lastly, Section 5 concludes the paper and identifies future work.

2. UN-INTERPRETED MATCHING

In this section we describe in detail our un-interpreted structure matching technique. The algorithm takes two table instances as input and produces a set of matching node pairs. Our approach works in two main steps as shown below. The function Table2DepGraph() in the first step transforms an input table like the one shown in Figure 3(a) into a dependency graph shown in Figure 3(c). The function GraphMatch() in the second step takes as input the two dependency graphs generated in the first step and produces a mapping between corresponding nodes in the two graphs.

1. G1 = Table2DepGraph(S1);
   G2 = Table2DepGraph(S2);
2. {(G1(a), G2(b))} = GraphMatch(G1, G2);

where Si = input table, Gi = dependency graph,
(G1(a), G2(b)) = matching node pair.

The two steps are described in detail later in this section.

2.1 Preliminaries

To construct a dependency graph, we use mutual information and entropy, which are defined as follows (these definitions are from "Elements of Information Theory" by Cover and Thomas [5]):

Definition 2.1 Mutual Information *Let X and Y be two attributes with alphabets $\mathcal{X}$ and $\mathcal{Y}$, respectively. Consider some joint probability distribution p(x, y) and marginal probability distributions p(x) and p(y) over two attributes. We define the mutual information of X and Y as:*

$$MI(X;Y) = \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log\frac{p(x,y)}{p(x)p(y)}$$

Definition 2.2 Entropy *Let X be an attribute with alphabet $\mathcal{X}$, and consider some probability distribution p(x) of X. We define the entropy H(X) by:*

$$H(X) = -\sum_{x\in\mathcal{X}} p(x)\log p(x)$$

Note that both entropy and mutual information are functions of probability distributions and thus are independent of the actual values of attributes. This property allows them to be used in un-interpreted matching. One interesting question we explore in our performance section is whether we need to compute mutual information, or whether entropy alone suffices. Our results show that mutual information can substantially improve the matching algorithm in many cases.

Entropy describes the uncertainty of values in an attribute with a non-negative real number. Similarly, mutual information describes the correlation between the two attributes' probability distributions, also using a non-negative real number. In other words, it measures the amount of information captured in one attribute about the other. This becomes more intuitive when we consider the relationship between mutual information and entropy. To explain this relationship we need one more basic definition, that of conditional entropy [5].

Definition 2.3 Conditional Entropy *Let X and Y be two attributes with alphabets $\mathcal{X}$ and $\mathcal{Y}$, respectively. We define the conditional entropy of X and Y as:*

$$H(X\mid Y) = -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log p(x\mid y)$$

Conditional entropy $H(X|Y)$ measures the uncertainty of attribute X given knowledge of attribute Y. It is a non-negative real number and becomes zero when $X=Y$ or when there exists a functional dependency from Y to X, because in these cases, no uncertainty exists for attribute X. On the other hand, if the two attributes X and Y are independent, the conditional entropy $H(X|Y)$ equals $H(X)$. We can now redefine the mutual information formula using entropy and conditional entropy.

$$\begin{aligned} MI(X;Y) &= \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log\frac{p(x,y)}{p(x)p(y)} = MI(Y;X) \\ &= \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log\frac{p(x\mid y)}{p(x)} \\ &= \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log p(x\mid y) - \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log p(x) \\ &= H(X) - H(X\mid Y) = H(Y) - H(Y\mid X) \end{aligned}$$

As we can see in the equation, mutual information measures the reduction in uncertainty of one attribute due to the knowledge of the other attribute. In other words, it measures the amount of information that one attribute contains about the other. It is zero when two attributes are independent, and increases as the dependency between the two attributes grows. Note that mutual information of an attribute with itself (called self information), $MI(X; X)$, is equivalent to the entropy of X, i.e. $H(X)$.

2.2 Modeling Dependency Relation

Consider the example illustrated in Figure 3. Figure 3(a) and 3(b) show two four-column input tables and 3(c) and 3(d) show the corresponding dependency graphs. The Table2DepGraph() function produces such dependency graphs by calculating the pair-wise mutual information over all pairs of attributes in a table and structuring them in an undirected labeled graph. For instance, each edge in the dependency graph G1 (Figure 3(c)) has a label indicating mutual information between the two adjacent nodes; for example, the mutual information between nodes *A* and *B* is 1.5, and so on. The label on a node represents the entropy of the attribute, which is equivalent to its mutual information with itself or self information. Hence we can model our dependency graph in a simple symmetric square matrix of mutual information, which is defined as follows:

Definition 2.4 Dependency Graph *Let S be a schema instance with n attributes and a_i ($1\le i\le n$) be its ith attribute. We define dependency graph of schema S using square matrix M by:*

$$M = (m_{ij}), \quad \text{where} \quad m_{ij} = MI(a_i; a_j), \quad 1\le i, j\le n$$

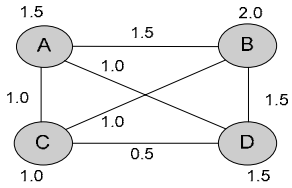
The intuition behind using mutual information as a dependency measure is twofold: 1) it is value independent; hence it can be

A	B	C	D
a1	b2	c1	d1
a3	b4	c2	d2
a1	b1	c1	d2
a4	b3	c2	d3

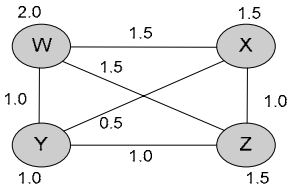
a) Example Table S1

W	X	Y	Z
w2	x1	y1	z2
w4	x2	y3	z3
w3	x3	y3	z1
w1	x2	y1	z2

b) Example Table S2



c) Dependency Graph (G1) of Table S1



d) Dependency Graph (G2) of Table S2

Figure 3. Two input table examples and their dependency graphs. A weight on an edge represents mutual information between the two adjacent attributes and a weight on a node represents entropy of the attribute (or equivalently, self-information $MI(A;A)$).

used in un-interpreted matching 2) it captures complex correlations between two probability distributions in single number, which simplifies the matching task in the second stage of our algorithm.

2.3 Matching Strategies

In this subsection, we focus on the second half of the schema matching process: GraphMatch(). Before we delve into the main discussion, let us first examine the types of cardinality constraints that we need to consider in schema matching. Let *A* and *B* be two input schemas that we are trying to match. We consider three types of cardinality constraints.

- One-to-one mapping ([1,1] – [1,1], in UML notation): Each attribute in *A* has a unique match in *B*, and vice versa. This corresponds to a case in which we know that the tables that we are trying to map have the same number of attributes, so the problem is just finding a correspondence between the attributes.
- Onto mapping ([0,1] – [1,1]): Each attribute in *A* has a unique match in *B* while each attribute in *B* either has a unique match in A or remains unmatched. This corresponds to a case in which we know that table *A*'s attributes are a subset of table *B*'s, so we have to discover this subset and then decide how to map attributes within this subset.
- Partial mapping ([0,1] – [0,1]): Each attribute in *A* either has a unique match in *B* or remains unmatched, and vice versa. This corresponds to the most general and difficult case in which we do not know which attributes of *A* map to *B*, nor do we even know how many attributes of *A* map to *B*. In this case we need to find the best subset of attributes of *A* to map to *B*, and also need to find how this subset of *A* should be mapped.

In the following, we will use distance metrics to evaluate the quality of matching. A distance is assigned to each instance of mapping between schema elements, and the goal is to find a mapping that optimize the distance, i.e., minimize it or maximize it, depending on how the distance metric is defined. One-to-one mappings and onto mappings both guarantee that all attributes in schema *A* will find matches in schema *B*, whereas partial mappings do not. Because of this, some distance metrics that work for one-to-one and onto mappings do not work for partial mappings. Let us formally define the class of such metrics:

Definition 2.5 Monotonic of Distance Metrics *Let A and B be two dependency graphs with sizes (#of nodes) n and m, respectively,* where $n \le m$. *Let $D_p(A,B)$ be the distance of best matching for two p node sub-graphs of A and B. The distance metric $D_p(A,B)$ is monotonic if and only if $D_p(A,B) \ge (or \le) D_{p+1}(A,B)$ for all graphs A and B, and for all p in $1 \le p \le n-1$.*

Monotonic metrics are not suitable for partial mapping because they reach their best score after either one attribute has been matched or all attributes have been fully matched depending on their direction of monotonicity, hence they will never produce a mapping in between (this problem doesn't arise with the one-to-one and onto mapping problems, because the problem statement enforces the number of columns to be matched.) To see this, suppose we are matching two schemas $R(r_1, r_2, .., r_n)$, $S(s_1, s_2, .., s_m)$. With a metric cost (or distance) of which increases monotonically as the size of matching grows, some pair of columns will be chosen first as being the best match; suppose this is r_i matched to s_j, and that the cost of this match is *c*. With such metric, we can never improve upon *c*, and the matching algorithm will just return that the "best" match is r_i and s_j, in effect not even considering matchings for additional columns. This is not appropriate for the partial mapping problem. Therefore, we need to be careful with metric selection in case of partial mapping. In this paper we consider two basic distance metrics, one monotonic, the other not monotonic. Clearly these are not the only possible metrics, and finding better metrics is an interesting area for future research. However, as we will see in the experimental section, these simple metrics perform surprisingly well. Consider the following basic distance metric:

Definition 2.6 Euclidean Distance Metric *Let A and B be two equal size dependency graphs and a_{ij} and b_{ij} be the mutual information between the node i and j in graph A and B, respectively. Let m be an index that maps a node in graph A to the matching node in graph B (i.e., m(node in A) = matching node in B). We define the Euclidean distance metric for graph A and B as:*

$$D_M^U(A,B) = \sqrt{\sum_{i,j}(a_{ij} - b_{m(i)m(j)})^2}$$

As we can see in the definition, the Euclidean distance metric is monotonic; that is, the distance between two input graphs increases monotonically as the number of matches increases. The minimum distance we get from the metric is always the distance of a single best matching node pair. Hence, we can not use the metric on partial mapping problems. As we pointed out, we need a non-monotonic distance metric for partial mapping. Here is one such metric:

Definition 2.7 Normal Distance Metric *Let α be some positive constant. Similarly, we can define the normal distance metric for graph A and B as:*

$$D_M^N(A,B) = \sum_{i,j}(1 - \alpha \frac{|a_{ij} - b_{m(i)m(j)}|}{a_{ij} + b_{m(i)m(j)}})$$

In the second term of the subtraction, we normalized the difference of two pairing mutual information values by dividing by the sum of the two values. The intuition behind this normalization is that, for example, mutual information values 8 and 9 are likely to indicate a better match than the pair 1 and 2 because the relative error in the latter is much greater than it is in the former. We refer to this normalized term, $|a_{ij} - b_{m(i)m(j)}| / (a_{ij} + b_{m(i)m(j)})$, as *normal distance*. The normal distance falls in the range of [0, 1] because the mutual information is non-negative real number. If we assume the mutual information values are uniformly distributed and we randomly choose two of them, the expected value of normal distance is 1/3. Now consider the control parameter α. In case of α =3, the expected value of whole distance metric becomes 0 with the normal distribution / random selection assumption. In such cases, the mapping of randomly chosen two attributes will not contribute to the distance metric. Conversely, if the two attributes map correctly the mapping will positively contribute to the distance metric.

By changing the parameter α, we can control the behavior of the distance metric. As we increase the α gradually from the original value, say 3, we will see the random mapping assignments start to contribute negatively to the distance metric. As a result, the matching returned from the normal distance metric with large α is likely to be more conservative than that with small α. That is, metric with large α returns smaller but high confidence candidate matches while the metric with small α returns larger but less confident candidates.

So far, we have discussed two distance metrics: one for monotonic and one for non-monotonic tasks. Let us now examine the search (or graph matching) algorithms we will use. In terms of complexity, the search for a one-to-one mapping is the easiest among the three cardinality types. Let *n* and *m* be the number of attributes in schemas *A* and *B*, respectively, and suppose that we are finding mappings from *A* to *B*. Then, the size of search space for one-to-one mapping is $O(n!)$. The search space of onto mapping is factor of mCn bigger than that of one-to-one mapping, which is $O(m!/(m-n)!)$. Finally, partial mapping is the one with the most flexible cardinality constraints and its complexity is asymptotically $\sum_{k=1}^{n} {}_nC_k \, m!/(m-k)!$. It is obvious that a naïve exhaustive search will be impractical for schemas with large numbers of attributes.

In practice, however, we can use instead an approximate search algorithm that trades off the accuracy of matching and the computational complexity. A large volume of literature has been devoted to finding such efficient, yet accurate graph matching approximations. In our experiments, however, we used a simple exhaustive search and did not explore the options of using approximations, because we wanted to measure the accuracy of un-interpreted matching precisely and the use of approximation might affect the measurement to some degree, due to the algorithm's own approximation error.

To improve upon a naïve exhaustive search and reduce the complexity at least to a tractable range (for example, hours not days), in our experiments in Section 4, we used simple heuristics to limit the search space. We set an upper bound for the number of match candidates that a search algorithm considers for each attribute. In our experiments, the match algorithm compares the entropies of attributes across the two input tables and chooses, for each source attribute, the closest *p* target attributes from the target table. We used three as the upper bound, *p*, in our experiments.

The match algorithm matches the two input graphs by finding the mapping that optimizes the distance metric. We can optimize different metrics in different ways. For example, if the Euclidean distance metric is used, the match algorithm must minimize the metric to find the best mapping. On the other hand, the match algorithm must maximize it if the normal distance metric is used.

Now, recall that one of our goals was to determine if mutual information matching is necessary, or whether entropy-only mapping was sufficient. To address this issue, we need an entropy-only version of the two distance metrics.

Definition 2.8 Entropy-only Euclidean Distance Metric *Let A and B be two tables with equal number of attributes and a_i and b_i be the entropies of attribute i in table A and B, respectively. Let m be an index that maps an attribute in table A to the matching attribute in table B. We define the entropy-only Euclidean distance metric for table A and B as:*

$$D_E^U(A,B) = \sqrt{\sum_i (a_i - b_{m(i)})^2}$$

Definition 2.9 Entropy-only Normal Distance Metric *Similarly, we can define the entropy-only normal distance metric for graph A and B as:*

$$D_E^N(A,B) = \sum_i (1 - \alpha \frac{|a_i - b_{m(i)}|}{a_i + b_{m(i)}})$$

The entropy-only matching works mainly in the same way as the mutual information based matching. It matches the attributes across the two input tables by finding the mapping that optimizes the entropy-only metric.

Let us now turn to a metric that we use to measure the accuracy of match result. We use *Precision* and *Recall*, which is a measure for answer quality widely used in the text retrieval community. Let *n* be the number of matches produced by a schema matching algorithm; *m* be the total number of true matches in two input schemas; and *c* be the number of correct matches in the produced match results. Given that, we can define Precision and Recall as follows:

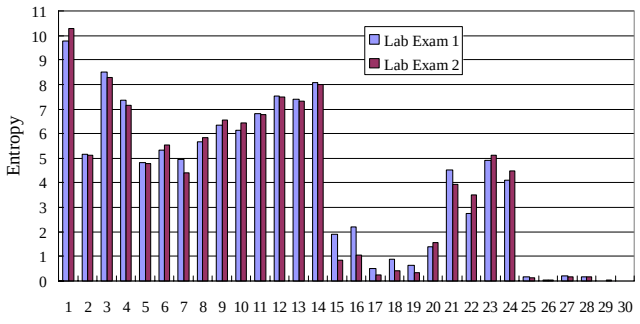
- Precision = c / n
- Recall = c / m

Note that Precision and Recall are identical if we are considering one-to-one mapping or onto mapping, because the number of produced matches and true matches are always same due to the cardinality constraints.

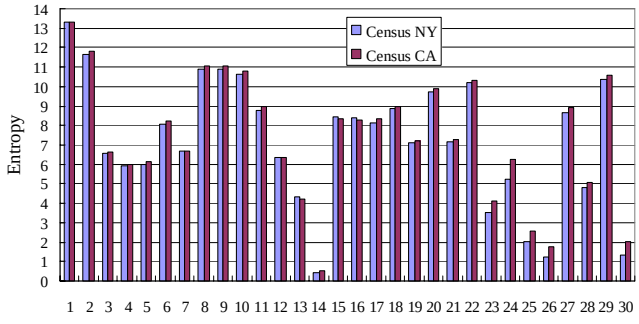
3. VALIDATING THE FRAMEWORK

In this section, we present the results of some schema matching experiments using our proposed approach. We ran experiments over two real-world data sets from different domains. We conducted several different types of experiments. For each type of cardinality constraint, we performed a set of experiments on different input sizes (or sizes of the overlap between two input schemas, in case of partial mapping) and different sample (data instance) sizes.

Testbed Implementation



a) Thrombosis lab exam 10K (#of tuples) samples



b) Census data 10K samples

1	2	3	4	5	6	7	8	9	10
97070	23	53	10	6.	4	14	0.5	23	10
97102	26	56	10	6.	5.	13	0.5	25	10
97122	25	48	90	6.	5.	15	0.6		
98012	26	57	10	7	4.	16	0.4	22	93
98021	34	52	98		5.	10	0.6	23	11
98031	35	54	95	6.	5.	13	0.5	24	11
98051	30	54	10	6.	5.	14	0.5	23	19
98063	26	55	10	6.	5	13	0.5	24	89
98082	26	22	20	6.	5.	9.8	0.6	20	18
98092	32	23	19	7.	5.	13.	0.6	27	15

c) First ten columns of Lab Exam 1 fragment

1	2	3	4	5	6	7	8	9	10
18091	1063	10	9	9	41	15	368	368	288
17511	3281	25	21	40	89	59	1211	1211	796
609	3424	29	13	15	148	26	1055	1055	861
3861	2884	18	7	4	114	11	670	670	568
18614	1478	12	10	15	40	16	630	630	459
3999	2414	29	16	27	87	21	967	967	753
5283	2385	42	17	39	46	40	968	968	622
21892	3053	28	17	16	99	33	1400	1400	1130
18554	14506	160	131	92	499	170	5084	5084	3965
12491	823	2	0	1	39	4	240	240	226

d) First ten columns of Census *CA* fragment

Figure 4. Attribute entropies of two data sets.

We implemented our two-step matching algorithm using Java 1.4. The first step of our algorithm uses a data loader and analyzer. The data loader/analyzer component loads data tables from text files; analyzes the loaded tables; and constructs dependency graphs. Then, the graph matching algorithm takes over and performs graph matching over the two dependency graphs. We implemented a naïve exhaustive search algorithm with simple filtering (considering in all cases only the *n* candidates with closest entropy values) to limit the search space. Although the filters we applied reduce the search space for the algorithms, the remaining search space is still very large. To run such expensive experiments, we divided the experiment runs to multiple sub-runs and executed them in parallel on multiple workstations. The full set of experiments with 50 iterations took approximately 5 hours to finish.

Data Sets

We used real-world data sets from two different data domains: medical data and census data. The medical data set we used in our experiments contains patients' lab exam results for diagnosing thrombosis [19]. Figure 4(a) shows the measured entropies of 30 randomly chosen attributes of the thrombosis lab exam data and Figure 4(c) shows a fragment of the first 10 (out of the 30) attributes' data values. The original table contains 12 years worth of patient exam records, which is approximately 50K tuples, and each tuple consists of 44 attributes representing test types. The column data types are mostly numeric, and a significant portion of the table is left blank (see attributes 15 – 30 in Figure 4(a).) Our basic experimental technique with the medical data set was to range partition the original table into two sub-tables based on exam dates (column 1) and to use these two sub-tables for experiments. We "pretended" that these sub-tables were two different tables that needed to have their schemas mapped. Obviously, we "knew" the correct answer for the mapping; but the mapping algorithm did not.

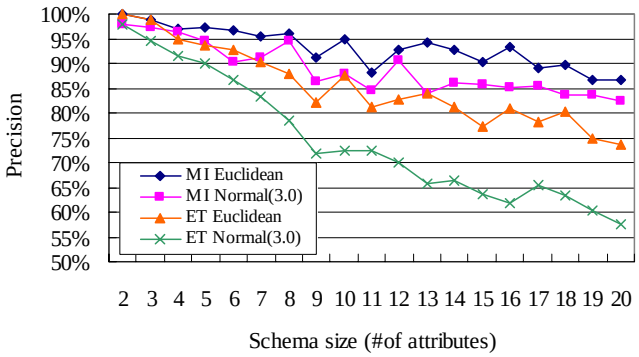
For our second data set, we used census data. Figure 4(b) and 4(d) show attribute entropies and a table fragment from the census data set, respectively. We used two state census data files, *CA* and *NY*, in our experiments [22]. Each table consists of 240 attributes. We ran the experiments over a randomly chosen set of 30 attributes.

Note that in Figure 4(d), attributes 8 and 9 are duplicated. The original census data files have some number of duplicate columns and two of them happened to be in the 30 attributes randomly chosen for our experiments. Evaluating the match results, we didn't count mappings like *NY*9 to *CA*8 a correct match; therefore the accuracy of matching was somewhat reduced degree by these duplicate columns.

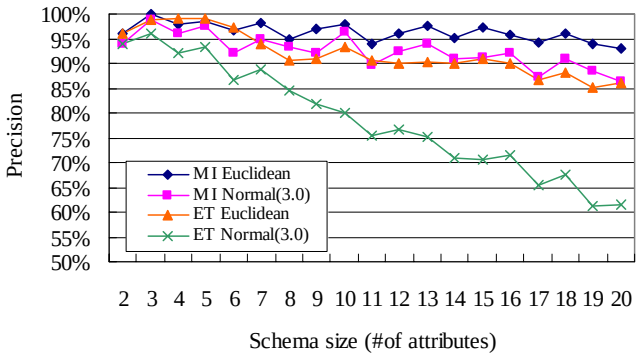
As we can see in Figure 4(a) and 4(c), even entropy-only matching gives a fair number of matches when the number of attributes to match is small and the attribute entropies differ by a substantial amount. However, the entropy-only approach fails when two or more attributes have close entropies. For example, in Figure 4(a), attributes 21 and 24 are likely to be cross-matched because the entropies of the two attributes are reversely ordered in the two tables. In our experiments below, we will continue to compare the entropy-only approach with our mutual information-based matching, to see if and when the additional complexity of considering mutual information is useful.

One-to-one mapping

Figure 5 presents the results of one-to-one schema matching. We ran the experiment while increasing the number of attributes in two input tables to be matched. For each table width, two to 20, we iterated the measurement 50 times with randomly chosen



a) Thrombosis lab exam 10K samples



b) Census data 10K samples

Figure 5. One-to-one mapping results.

subsets of attributes and averaged the results. Entropy-only matching results (labeled *ET*) are also presented to show the improvements obtained by taking into accounts of correlations between the attributes, which is given in the results of mutual information based matching (labeled *MI*). Furthermore, we tested both Euclidean and normal distance metrics in both entropy only and mutual information matching.

Figure 5(a) shows the precision of match results using thrombosis lab exam 10K tuple samples. As we see in Figure 5(a), match results obtained from narrow tables are better than that from wider tables. As the tables get wider, the precision of matching deteriorates. Comparing the two matching techniques, the entropy only matching combination shows much faster deterioration than mutual information matching. The best performer was the mutual information matching using the Euclidean distance metric, and the worst was entropy only matching using the normal distance metric. Comparing two metrics, the Euclidean distance metric works better than the normal distance metric in both the entropy only and mutual information matching. We used 3.0 for the normal distance metric's control parameter α. However, the value of the control parameter, α, has no effect in the match results in this case. As we mentioned in Section 3, the control parameter α balances the precision and recall of the match results. Both precision and recall are, however, always the same in one-to-one mapping and onto mapping.

Figure 5(b) shows the match results using the census data set 10K tuple samples. Although the overall precision is slightly better, the results look quite similar to those presented in Figure 5(a). Similarly, in Figure 5(b), mutual information matching yielded superior results to entropy only matching and the Euclidean distance metric performed better that the normal distance metric. Mutual information matching with the Euclidean metric produced a matching of approximately 93% accuracy when two 20 column tables were matched, in which on average, more than 18 attributes were correctly matched while only two mismatched. On the other hand, 85% accuracy was achieved by entropy only matching using the same metric. In Figure 5(a) with the lab exam data set, we had 86% and 74% accuracy for mutual information and entropy only matching, respectively, which can be interpreted as 3 misses and 5 misses out of 20 true matches.

The results from census data were slightly better than those of the lab exam data. One explanation for this can be found in the entropy signature of the two data sets shown in Figure 4. In Figure 4(a), we can see that the last six attributes, from 25 to 30, have very low entropy values. These are the columns in the original data that have mostly null values. Because of the lack of information in them, these columns do not contribute much to the match results. By contrast, in the census data, only one such attribute exists, which is attribute 14 in Figure 4(b).

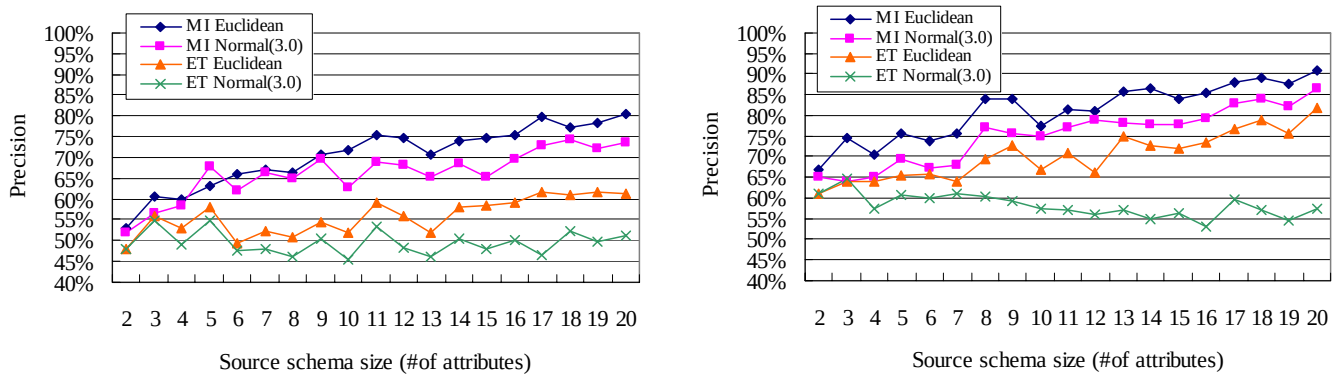
Turning to the issue of deterioration, one plausible explanation is that the search space (or the number of match candidates) grows super exponentially as the size of matching schema increases. For instance, matching two schemas of size two has only two match candidates; that is, for schemas *S1*(*a1*, *b1*) and *S2*(*a2*, *b2*), we can match either *a1*-*a2* and *b1*-*b2* or *a1*-*b2* and *b1*-*a2*. By contrast, for schemas of size 20, we have 20! candidates to search. Considering the super exponential growth of search space, the deterioration of mutual information matching precision is relatively small.

Onto mapping

Figure 6 illustrates the results of schema matching with the onto cardinality constraint. Figure 6(a) shows the results from thrombosis lab exam data and Figure 6(b) shows census data. In this experiment, we kept the target schema size constant at 22 attributes while increasing the source schema size from two to 20 attributes. In each step, as was done in our one-to-one mapping experiments, we iterated the measurement 50 times with randomly chosen subsets of attributes and averaged the results.

As was the case in the one-to-one mapping experiments, the census data match result is slightly better than that of the lab exam data. For example, with census data, the precision of the match result reached 81% when matching 11 attributes out of 22, while it was 75% with lab exam data. The performance gap between the two data sets widened as the source schema size increased from there; e.g., when the schema size reached 20, census data yielded 91% precision while lab exam data turned out only 80%. The performance gap phenomenon is consistent with what we observed in the one-to-one mapping results shown in Figure 5 and has a similar explanation.

In both data sets, mutual information matching outperformed entropy only matching. The precision of lab exam data matching was improved approximately 31% (from 61% in entropy only to 80% with mutual information) while precision in census data



a) Thrombosis lab exam 10K samples

b) Census data 10K samples

Figure 6. Onto mapping results. Target schema size is kept constant at 22 attributes while source schema size varying.

improved 12% (from 81% in entropy only to 91% for mutual information). We see that mutual information was more helpful for the lab exam data than it was for the census data. This is because in the lab exam data, more attributes had similar entropy, so that entropy-only mapping was more likely to get "confused." Turning now to compare our two metrics, Euclidean and normal, the Euclidean distance metric yielded better results overall in both data sets, which is consistent with our observations in the one-to-one mapping results.

To summarize the situation up to this point, we have considered the performance of two matching methods and two distance metrics, and the results have been consistent with those in the one-to-one mapping case.

However, there is a notable difference: the precision of matching in the onto case improves as the size of source schema increases, which is the opposite of what we saw in the one-to-one mapping case. We turn now to explain this phenomenon.

Let us consider the matching as two step process: selecting a subset of attributes from the target schema, and searching for the correct permutation of this selected subset. The reason that the onto experiments had better performance with a larger source schema is that the first step is harder than the second. If the first step were easy, the result of the onto mapping experiments should have looked similar to that of the one-to-one experiments. To illustrate this, let us consider an extreme case where the first step always returns the correct attribute subset. With this assumption, the onto mapping problem reduces to the one-to-one mapping problem. However, the result of the onto mapping is opposite to that of the one-to-one mapping; that is, as the schema size increases, the onto mapping precision improves while one-to-one mapping precision deteriorates.

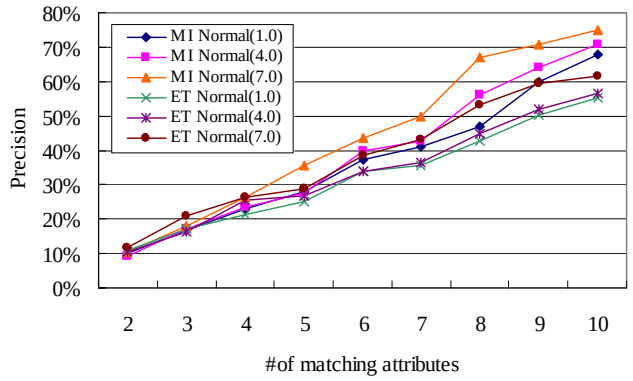
Now suppose that the second step always returned the correct permutation. Then the onto matching problem reduces to choosing the correct attribute subset from the target schema. In fact, this assumption is not too far from the real situation, because as shown in the one-to-one mapping results, the second step indeed produces almost perfect results, especially when the number of attributes is small. For example, consider the case of finding two attributes out of 22 attributes. The total number of possible selections is 231, and one of them is the correct selection and 40 others have only one correct attribute (50% precision). The remaining 190 selections yield no match (therefore 0% precision). Whereas, in case of finding 20 attributes out of 22, the maximum mismatch number is two; therefore, it will achieve 90% precision in the worst case. Considering this, it is easy to see why the precision improves in spite of the fast growing search space.

Partial mapping

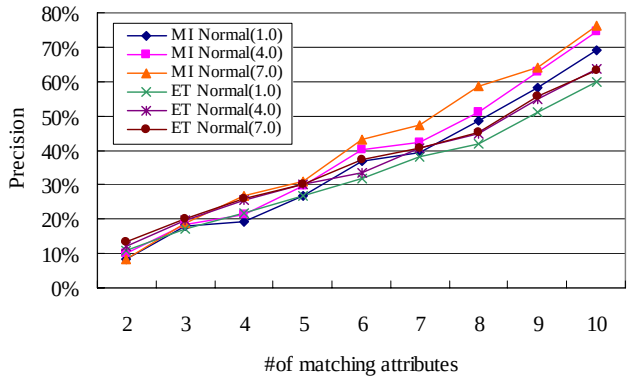
Figure 7 illustrates the results of schema matching with the partial mapping cardinality constraint. Figure 7(a) and 7(c) show the precision and recall of the thrombosis lab exam data results and Figure 7(b) and 7(d) shows the precision and recall of the census data results, respectively. In this experiment, we keep the size of both source and target schema constant at 12 attributes while varying the number of correct matches from two to 10 attributes. In each step, as was done in the previous experiments, we iterated the measurement 50 times with randomly chosen subsets of attributes and averaged the results. Unlike previous two cases, partial mapping requires a non-monotonic distance metric because in this case, the size of source schema and the number of correct matches are not necessarily same. For the same reason, both precision and recall should be examined.

In this experiment, we used the normal distance metric with three different control parameter values: 1, 4, and 7. In Figure 7(a), *MI Normal*(*1.0*) represents mutual information matching using normal distance metric with control parameter α=1, and similarly others. Unlike the previous two cases, it is not easy to tell which approach dominates from the experiments. In fact, the choice of α is dependent on the application semantics. If an application prefers a small number of candidates with high confidence, then a larger α is going to be more suitable. In contrast, if the application is willing to accept relatively low confidence in match candidates but wants as many probable matches retrieved as possible, then smaller α should work better.

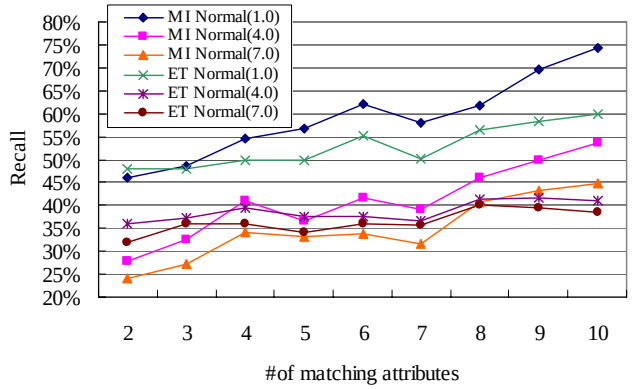
For instance, in Figure 7(a), *MI Normal*(*7.0*) achieved 75% precision where the two input schemas contain ten true matches, while the same metric turned out only 45% recall at the same point in Figure 7(c). In other words, it produced candidate matches, 75% of which were correct, and the number of correct matches in the candidates was 45% of the number of total true matches. On the other hand, *MI Normal*(*1.0*) achieved approximately 67% precision, while it turned out 75% recall at the same point in the graph. It is intuitively clear that the normal distance metric with α=1.0 returned a larger number of candidates than the metric with α=7.0. In fact, we can calculate the average



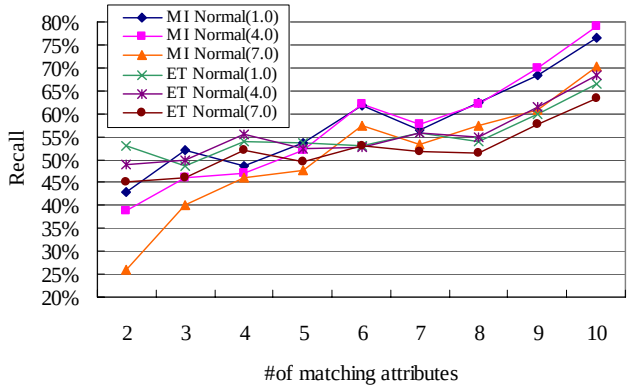
a) Precision of lab exam mapping results (10K samples)



b) Precision of census data mapping results (10K samples)



c) Recall of lab exam mapping results



d) Recall of census data mapping results

Figure 7. Partial mapping results. The size of both source schema and target schema is set to 12 attributes, while the number of correct matches varies.

number of candidates returned by the two metrics using the definition of precision and recall given in Section 3. The normal distance metric with α=7.0 (where #of matching attributes=10) returned on average six candidates while the metric with α=1.0 returned more than 11 candidates.

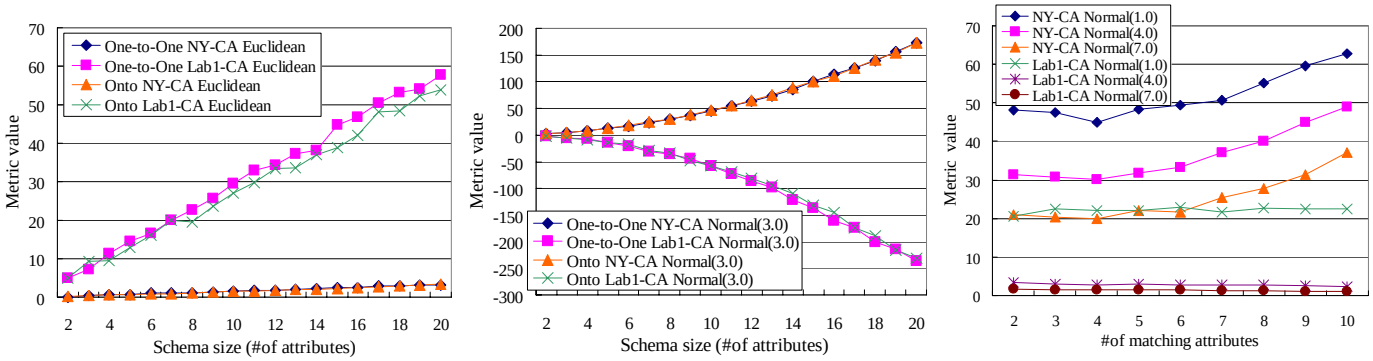
As was the case in the previous two scenarios, the performance on the census data set is slightly better than that of lab exam data. In Figure 7(b) and 7(d), *MI Normal*(*4.0*) achieved approximately 75% precision and 79% recall, where the number of matching attributes is ten. Comparing two matching methods, in the lab exam data set, mutual information matching improved entropy only matching results by approximately 24% in both precision and recall, where the number of matching attributes was ten and α was 1.0. In case of census data set, the improvement was 19% and 16% for precision and recall, respectively, at the same data point in Figure 7(b) and 7(d) (i.e., #of matching attributes=10) using α = 4.0.

Unlike previous two scenarios, the search space for partial mapping remains same throughout the experiments with varying numbers of matching attributes. The search space in our experiment in Figure 7 (i.e., 12 attribute to 12 attribute schema matching) was over 53 billion possible matches. Although the search space did not change, the accuracy of results improved as the number of matching attributes increased. The explanation we gave for the onto mapping scenario applies here as well. It is easier to find more matches when the number of true matches is greater. Compared to the onto mapping experiments, however, the accuracy of results is significantly dampened when the number of true matches is small.

For example, in Figure 7(a) and 7(c) where only two true matches exist, the precision of the results was around 10% with the recall of 46% (using α = 1.0). In experiments with the onto mapping, the precision was over 50% with the same data set. One reason for this discrepancy is that the search space for partial mapping is significantly larger than that for the onto mapping, and the difference is more striking in cases with a small number of matches. Another possible reason is that we were using normal distance metrics in the partial mapping scenario, which was shown to be inferior to the Euclidean distance metrics in the previous two matching scenarios. As pointed out earlier, the Euclidean metric is monotonic and therefore is not applicable to the partial mapping scenarios.

On the Result of Unrelated Schema Matching

So far we have examined the cases of matching where we know matches exist. In this subsection, we consider the matching of unrelated schema instances. We examine how our matching algorithm reacts to the matching of such schema pairs. The ability of identifying relevant/irrelevant schema instances from many others is an important aspect of schema matching technique. Consider a scenario of Web-source integration. The number of sources on the Web grows daily, and, to make matters worse, sources are added and removed in an uncontrolled fashion.



a) Euclidean distance metric values of one-to-one and onto mapping results

b) Normal distance metric values of one-to-one and onto mapping results

c) Normal distance metric values of partial mapping results

Figure 8. Distance metric values of matching results. Matching results of two schema instance pairs are compared: *NY-CA* and *Lab1-CA*.

Without automated tool support, the task of building and maintaining Web-source information integration systems will be very human-labor intensive. Considering this, at the very least, the ability of classifying or clustering schemas is a desirable feature for a (semi) automatic schema matching tool. Therefore, as a first step, we would like to investigate if our matching techniques can distinguish "good" or "bad" candidates for matchings of schemas.

Turning to our schema matching algorithm, ideally, we will see that the algorithm turns out much worse metric values in "bad" cases (where the two tables being considered are really distinct) than it does on the "good" cases (where the two tables logically should be integrated). One such example is shown in Figure 8(a). We tried matching the Census data for California to the Lab exam1 (henceforth *Lab1-CA*) data set and compared the results to the case of a correct matching (matching New York's census data to California's census data.) Figure 8(a) shows the Euclidean metric values of one-to-one and onto mapping results. As we expected, *NY-CA*'s Euclidean distance grows in much lower rate than *Lab1-CA*'s distance as the size of source schema increases. Note that with the Euclidean distance metric, the smaller the distance between two schemas, the closer the matching tool thinks the schemas are. On the other hand, with the normal distance metric, larger metric values mean closer matching. Figure 8(b) shows the normal distance metric values of the same set of test shown in Figure 8(a). Unlike the previous case, *Lab1-CA*'s normal metric value declines while *NY-CA*'s normal metric grows, for both one-to-one and onto mapping.

Figure 8(c) shows the results of partial mapping using normal metric with three different control parameters (α): 1.0, 4.0, and 7.0. In all three cases, *NY-CA*'s normal metric (Figure 8(c)) grows in a similar fashion as the normal metric of one-to-one and onto mapping cases (i.e., Figure 8(b) *NY-CA*.) On the other hand, *Lab1-CA*'s normal metric values are virtually unchanged. The reason for this is that in partial mapping, there is no cardinality constraint and because of that, the mapping always turns out the smallest possible matches (in case with α of 4.0 and 7.0) for all range in x-axis of the graph because there is no true match between *Lab1* and *CA*. In case of $\alpha \leq 1.0$, the normal distance metric becomes a monotonic metric because the subtraction inside of summation is always greater than or equal to zero; as a result, the metric always turns out maximum matches. Because of this, the *Lab1-CA*'s *Normal*(*1.0*) shows relatively high values compared to the metric with two other α values. Despite that, the metric values are virtually unchanged throughout the range as no matches can be found.

Considering the results in Figure 8, our technique shows promise in that it clearly distinguishes the case of being applied to two tables that should be integrated and being applied to two tables that are logically disjoint.

Summary of Experimental Results

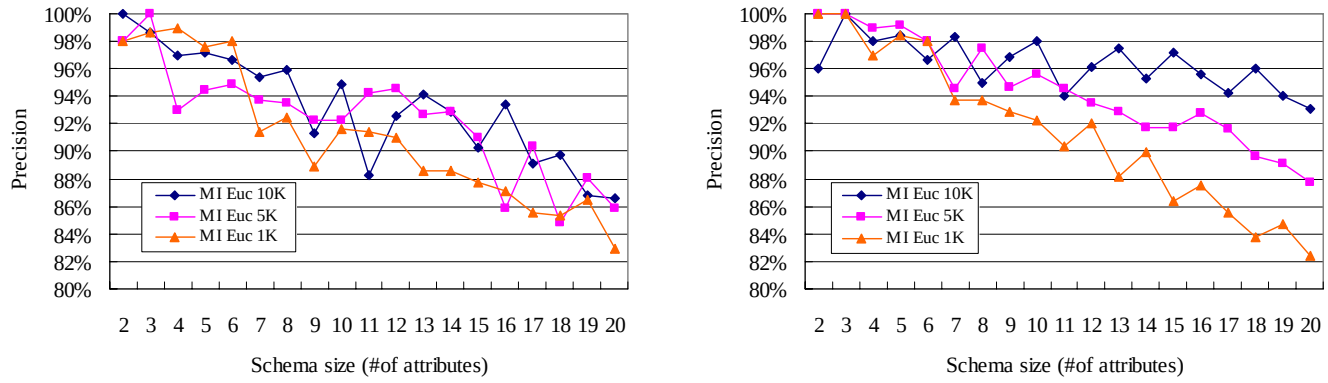
In summary, the accuracy of matching results is generally better when the two schemas to be matched have a larger overlap (in relative terms; i.e., five out of ten is better than five out of 20). Accuracy in the one-to-one mapping scenario (100% overlap) was in the range of 85% to 95%; in the onto mapping scenario (approx. 90% overlap, where source schema size=20) this dropped to 80% to 90% accuracy; and in the partial mapping scenario (83% overlap, where #of matching attributes=10) achieved around 70% precision and 75% recall on both data sets.

One of the reasons for worse results in smaller overlap cases is that at least in our experiments, the task of finding a correct attribute subset to which to map is much more difficult than the task of finding the right permutation once the subset has been identified.

For both data sets, lab exam and census data, we used 10K tuple samples throughout the experiments. Figure 9 shows the effects of sample size in schema matching accuracy. Figure 9(a) presents the results of lab exam data and Figure 9(b), that of census data. As we expected, the larger sample produced the better accuracy in both data sets. However, the effect of sample size was much greater in census data set. One reason for this is that census data is denser than the lab exam data in that census data has no nulls whereas lab data has large number of empty fields and therefore, the contribution to the matching results per tuple units is much greater with census data than with lab exam data.

4. RELATED WORK

Most previously proposed work on schema matching has focused on developing *interpreted* matching techniques (see [20] for survey). Such techniques are largely dependent on identifying similarity in schema element names, common data representation formats, or common data domains. Because our technique is



a) Thrombosis lab exam one-to-one matching

b) Census data one-to-one matching

Figure 9. Effects of sample size in match results. One-to-one mapping precisions with 1K, 5K and 10K samples are presented.

based on un-interpreted matching, it can complement existing techniques and can be combined with traditional schema matching systems.

Some proposed techniques employ machine learning. Li and Clifton proposed a neural network-based schema matching prototype called SemInt [13][14]. Berlin and Motro proposed Automatch, a technique based on machine learning with feature selection [2]. Another machine learning approach, LSD, was proposed by Doan et al. [6][7] LSD employs multi-strategy learning with three-level architecture. Although these systems are flexible, they need to be re-trained before being able to be used in a new application domain. More importantly, learners rely on data interpretation and therefore it is not applicable to our problem domain.

Other work has considered rule-based schema matching. These include TranScm [18] and ARTEMIS [4]. Both TranScm and ARTEMIS are schema-based matching techniques and our un-interpreted instance-based technique can be combined with them to improve the accuracy of matching. Some other techniques represent a schema in a graph format and perform matching based on the structural similarity of the two graph representations. Cupid [15] and Similarity Flooding [16] fall into this category. Unlike our scheme, both Cupid and Similarity Flooding rely on schema-based structural similarity and therefore it is not applicable to our problem domain.

Meanwhile, though they are not targeted to schema matching, many generic graph matching algorithms have been developed in the theoretical computer science literature. One example is the Graduated Assignment Algorithm developed by Gold and Rangarajan [9]. These algorithms can be tuned to match our dependency graphs and replace the exhaustive search algorithm used in our experiments.

At another end of the spectrum, the schema mapping system called Clio [12][17][23] creates a mapping between two input schemas in an interactive fashion using user feedback. It produces as a mapping a view definition (mapping query) over the target schema so that a meta query engine can execute the mapping query and as a result, translate the data from the original schema into the target schema. Our un-interpreted matching and Clio can be combined because Clio focuses on finding correspondences between data instances while un-interpreted matching focuses on finding mappings between schema elements.

In our work, we used mutual information and entropy to represent interaction between attributes. These concepts are popular in the information theory community and have been well accepted in other domains as well [5]. Although we found that mutual information is an effective tool for capturing dependencies in an un-interpreted manner, there exist other ways that this could be accomplished. One interesting approach would be to use Bayesian network structure learning [8][10][11]. Bayesian networks capture dependency (or sometimes causal) relations between attributes in the form of conditional probability distributions. Among many others, [8] and [10] caught our attention because they use mutual information to limit potentially intractable search spaces of possible structures.

Finally, Bernstein et al. presented model management scenarios in their vision paper [3]. They proposed a unified framework for applications to access underlying models using high level operators such as *Match*, *Merge*, *ApplyFunction* and *Compose*. The schema matching technique reported in this paper works as a *Match* operator in model management. Developing remaining operators using our un-interpreted method would be an interesting area for future work.

5. CONCLUSION

In this paper we investigated schema matching techniques that work in the presence of opaque column names and data values. We proposed a two-step technique that does not rely on the interpretation of data elements or schema elements. To our knowledge our paper is the first to introduce an *un-interpreted matching* technique utilizing inter-attribute dependency relations. We have shown that while a single column un-interpreted matching such as entropy-only matching can be somewhat effective alone, further improvement were possible by exploiting inter-attribute correlations. The improvement obtained was in the range of 9% to 31% depending on the cardinality constraints for the mapping problem and the data sets used in the experiments.

A good deal of room for future work exists. In our work, we have only tested two simple distance metrics, Euclidean and normal. It is possible that more sophisticated distance metrics could produce better results. An interesting and important direction would be to

search for metrics that specialize in the task of finding correct subset of attributes from target schemas. Another potentially interesting direction would be to find an accurate yet computationally efficient approximation algorithm for the instances of the graph matching problem generated by our approach to the schema mapping problem. It would also be interesting to evaluate other dependency models using different un-interpreted methods. In this work, we focused only on matching flat tables. Extending the technique to nested structures (for example, XML or object-oriented schemas) would be another interesting direction for future work. Furthermore, in our experiments, we only tested our matching techniques against the tables that are produced by the same organization. It would be interesting to apply our algorithm to tables generated independently by two organizations, and to explore when data sets are likely to exhibit properties that make them amenable to our un-interpreted instance-based approach.

Finally, the schema matching problem we considered in this paper is only one small subpart of a larger problem. A more complete solution to the data integration problem requires the additional steps (at the very least) of identifying which tables are candidates for matching and handling cases in which the tables in the two schemas do not have a one-to-one correspondence. It is our hope that the un-interpreted matching approach we present here can be useful as a piece of the solution to this larger problem. We are encouraged by the results we present in this paper, since they show that our technique, which does not rely on any interpretation of the schemas or data instances in the matching problem, can distinguish schemas that "make sense" to integrate from those that do not, and can propose good matchings for schemas that do make sense to integrate.

Acknowledgements

We would like to thank anonymous reviewers for their valuable comments. This research was supported by NSF grants CSA-9623632 and ITR 0086002.

REFERENCES

[1] Domenico Beneventano, Sonia Bergamaschi, Silvana Castano, Alberto Corni, R. Guidetti, G. Malvezzi, Michele Melchiori, Maurizio Vincini: Information Integration: The MOMIS Project Demonstration. VLDB 2000: 611-614

[2] Jacob Berlin, Amihai Motro: Database Schema Matching Using Machine Learning with Feature Selection. CAiSE 2002: 452-466

[3] Philip A. Bernstein, Alon Y. Halevy, Rachel Pottinger: A Vision of Management of Complex Models. SIGMOD Record 29(4) 2000

[4] Silvana Castano, Valeria De Antonellis, Sabrina De Capitani di Vimercati: Global Viewing of Heterogeneous Data Sources. TKDE 13(2): 277-297 (2001)

[5] Thomas M. Cover and Joy A. Thomas. Elements of Information Theory. John Wiley & Sons, Inc., New York, 1991.

[6] AnHai Doan, Pedro Domingos, Alon Y. Halevy: Reconciling Schemas of Disparate Data Sources: A Machine-Learning Approach. SIGMOD Conference 2001

[7] AnHai Doan, Pedro Domingos, Alon Y. Levy: Learning Source Description for Data Integration. WebDB (Informal Proceedings) 2000: 81-86

[8] Nir Friedman, Iftach Nachman, Dana Peer: Learning Bayesian Network Structure from Massive Datasets: The "Sparse Candidate" Algorithm. UAI 1999: 206-215

[9] Steve Gold and Anand Rangarajan. A graduated assignment algorithm for graph matching. IEEE Transactions on Pattern Analysis and Machine Intelligence, 18(4), April 1996.

[10] Lise Getoor, Benjamin Taskar, Daphne Koller: Selectivity Estimation using Probabilistic Models. SIGMOD Conference 2001

[11] D. Heckerman. A tutorial on learning with Bayesian networks. Technical Report MSR-TR-95-06, Microsoft Research, March, 1995 (revised November, 1996)

[12] Mauricio A. Hernández, Renée J. Miller, Laura M. Haas: Clio: A Semi-Automatic Tool For Schema Mapping. SIGMOD Conference 2001

[13] Wen-Syan Li, Chris Clifton: SEMINT: A tool for identifying attribute correspondences in heterogeneous databases using neural networks. DKE 33(1): 49-84 (2000)

[14] Wen-Syan Li, Chris Clifton: Semantic Integration in Heterogeneous Databases Using Neural Networks. VLDB 1994: 1-12

[15] Jayant Madhavan, Philip A. Bernstein, Erhard Rahm: Generic Schema Matching with Cupid. VLDB 2001: 49-58

[16] Sergey Melnik, Hector Garcia-Molina, Erhard Rahm: Similarity Flooding: A Versatile Graph Matching Algorithm. ICDE 2002

[17] Renée J. Miller, Laura M. Haas, Mauricio A. Hernández: Schema Mapping as Query Discovery. VLDB 2000: 77-88

[18] Tova Milo, Sagit Zohar: Using Schema Matching to Simplify Heterogeneous Data Translation. VLDB 1998: 122-133

[19] PKDD 2001 Discovery Challenge on Thrombosis Data. http://lisp.vse.cz/challenge/pkdd2001/

[20] Erhard Rahm, Philip A. Bernstein: A survey of approaches to automatic schema matching. VLDB Journal 10(4) (2001)

[21] Triada, Ltd. http://www.triada.com/

[22] U.S. Census Bureau. Census data file ftp site. ftp://ftp2.census.gov/census_2000/datasets/

[23] Ling-Ling Yan, Renée J. Miller, Laura M. Haas, Ronald Fagin: Data-Driven Understanding and Refinement of Schema Mappings. SIGMOD Conference 2000