Establishing Value Mappings Using Statistical Models and User Feedback

Jaewoo Kang Tae Sik Han
North Carolina State University
Raleigh, NC, 27695, U.S.A.
{kang,tshan}@csc.ncsu.edu

Dongwon Lee Prasenjit Mitra
The Pennsylvania State University
University Park, PA, 16802, U.S.A.
{dlee,pmitra}@ist.psu.edu

ABSTRACT

In this paper, we present a "value mapping" algorithm that does not rely on syntactic similarity or semantic interpretation of the values. The algorithm first constructs a statistical model (e.g., co-occurrence frequency or entropy vector) that captures the unique characteristics of values and their co-occurrence. It then finds the matching values by computing the distances between the models while refining the models using user feedback through iterations. Our experimental results suggest that our approach successfully establishes value mappings even in the presence of opaque data values and thus can be a useful addition to the existing data integration techniques.

Categories and Subject Descriptors

H.2.5 [**Database Management**]: Heterogeneous Databases —*Data translation*

General Terms

Algorithms, Management

Keywords

Value Mapping, Semantic Correspondences, User Feedback, Statistical Model

1. INTRODUCTION

Integrating data from multiple heterogeneous sources often involves two related subtasks: (1) reconciling structural heterogeneity of data by mapping schema elements across the data sources – *schema matching* problem; and (2) resolving semantic heterogeneity of data by mapping data instances across the tables – *object mapping* problem. Depending on the granularity of the object, the object mapping problem is also known as various names. For instance, if the object is a record, then it becomes the record linkage problem. Similarly, for tuple, it becomes the database join problem. In this paper, in particular, we focus on the object mapping problem with the object being "values," thus named as the **value mapping** problem.

Virtually all previous value mapping works assume the data values in each corresponding columns are drawn from the same domain or at least they bear some textual similarity. However, this assumption is often challenged in practice where sources use various different representations for describing their data. For example, "two-door front wheel drive" can be represented as "2DR-FWD" or "R2FD", or even as "CAR TYPE 3" in different data sources. Some smart string distance algorithms may be able to suggest correspondences among the first three representations, but they will fail to establish any mapping for "CAR TYPE 3" as it bears no syntactic or semantic clue except the fact that it is about car type.

This problem poses a substantial challenge to the existing object mapping techniques. To address this, we present a novel, semi-automated technique that can be of assistance in the particularly difficult cases in which the data values to be matched are "opaque," or difficult to understand, and have little syntactic or lexical similarity. To gain insight into our approach, consider the employee tables in Table 1. The task of matching values in the $Title$ and $Degree$ columns is not quite straightforward. For this case, most traditional techniques that rely on the textual similarity of data, will likely fail to identify the value mapping.

We propose techniques that use the co-occurrence of values and statistical methods like entropy that captures the distributions of co-occurring values to address the value mapping problem. For example, suppose we are trying to find the value in Table Y that maps to "Professor" in Table X as shown in Table 1. To make the exposition simpler, let us assume that we know the correspondences between the values in the $Degree$ columns ($e.g., Ph.D. \rightarrow D7$, $M.S. \rightarrow D3$, $B.S. \rightarrow D2$). Intuitively, we will see a higher correlation between "Ph.D." and "Professor" than between "Ph.D." and "T.A.", as is the case in Table X. If we can measure the correlation between "D7" (which we know is "Ph.D.") and the two values, "EMP10" and "EMP3", in Table Y, and can compare the measurements across the tables, we may be able to find further correspondences. Now, let us assume that the mappings between the two $Degree$ columns were not known *a priori* (i.e., the mappings for both $Title$ and $Degree$ are unknown.). How can we proceed?

To address this problem, we propose an algorithm that works as follows: (1) We first construct for each table a statistical model (e.g., co-occurrence frequency or entropy

Name	Gender	Title	Degree	Status
J. Smith	M	Professor	Ph.D.	Married
R. Smith	F	T.A.	B.S.	Single
B. Jones	F	T.A.	M.S.	Married
T. Hanks	M	Professor	Ph.D.	Married

(a) Table X

Name	Gender	Title	Degree	Status
S. Smith	F	Emp10	D7	SGL
T. Davis	M	Emp3	D3	SGL
R. King	M	Emp10	D7	MRD
A. Jobs	F	Emp3	D2	MRD

(b) Table Y

Table 1: University employee tables.

vector) that captures the unique characteristics of values and their co-occurrence within the table; and then (2) finds the matching values by computing the distances between the models (not the values themselves) while refining the models using user feedback through iterations. In this paper we make the following contributions:

- We present the value mapping problem as a first class citizen in data integration that is considered as a real and hindering problem in the industry.
- We propose an interactive and iterative algorithm that can build the mapping incrementally while incorporating user feedback through iterations.
- Since the actual tokens representing the values are not taken into consideration in the matching process, our algorithm works well in matching even opaque data values.
- Our algorithm complements many existing algorithms as it utilizes different types of information that is not commonly used in the existing algorithms, and hence can be a useful addition to the existing data integration techniques.

2. PROBLEM & SOLUTION OVERVIEW

Problem Definition. We have two tables: the source table, S, with columns, $s_1, \ldots, s_n$, and its corresponding target table, T, with columns, $t_1, \ldots, t_n$. Furthermore, via column-level schema matching, the matching columns between S and T have already been identified. Using the schema match the two tables are pre-processed such that the schema match is converted from an m:n match to a bijective match. For example, if the schema match indicates that $S.Name$ is equivalent to the concatenation of $T.Firstname$ and $T.Lastname$, we preprocess the two tables. During the pre-processing, the columns $T.Firstname$ and $T.Lastname$ are composed to create a column with the concatenation of the first name and the last name. In the processed tables, there is a bijective mapping, say f, from columns in S to columns in T (after pre-processing). Formally, we consider the following as the **Value Mapping Problem**:

> For a pair of corresponding columns, s_i and t_j, such that $f : s_i \rightarrow t_j$, where f is a schema-mapping function, find a value mapping function $g : v_i \rightarrow v_j$, where v_i is a value in column s_i and v_j is a value in t_j, $1 \leq i, j \leq n$ such that v_i and v_j represent the same real-world values.

In general, g can be many-to-many and our algorithm is capable of finding many-to-many value mappings.

```
Input  : Value sets, $D(s)$ and $R(t)$
Output : Value mappings, $g : v_s \rightarrow v_t \| v_s \in D(s), v_t \in R(t)$

$M(s) \leftarrow$ Build($D(s)$); $M(t) \leftarrow$ Build($R(t)$);
while there is an unfound mapping do
    $[g_1, ..., g_n] \leftarrow$ Match($M(s), M(t)$);
    User confirms/denounces $g_i : v(s_i) \rightarrow v(t_j)$;
    $M(s) \leftarrow$ Refine($D(s) + \Delta(v(s_i))$);
    $M(t) \leftarrow$ Refine($R(t) + \Delta(v(t_j))$);
end
```

Algorithm 1: Overview of our approach.

	A1	A2	A3	A4	A5	A6	A7	A8	A9
A1	0.5	0	0.5	0	0.5	0	0	0.5	0
A2	0	0.5	0	0.5	0	0.25	0.25	0.25	0.25
A3	0.5	0	0.5	0	0.5	0	0	0.5	0
A4	0	0.5	0	0.5	0	0.25	0.25	0.25	0.25
A5	0.5	0	0.5	0	0.5	0	0	0.5	0
A6	0	0.25	0	0.25	0	0.25	0	0.25	0
A7	0	0.25	0	0.25	0	0	0.25	0	0.25
A8	0.5	0.25	0.5	0.25	0.5	0.25	0	0.75	0
A9	0	0.25	0	0.25	0	0	0.25	0	0.25

Table 2: A co-occurrence matrix for Table 1(a).

Solution Overview. Our value mapping algorithm finds a mapping among values using their signature vectors (e.g., co-occurrence frequency or entropy vectors; refer to Section 3 for more details) without interpreting individual values. An overview of our solution is illustrated in Algorithm 1, where g_i is a value mapping between two values, $g_i : v_i \rightarrow v_j$, and $\Delta(v_i)$ depicts the incorporation of user feedback on the value v_i. The algorithm works mainly in two phases: (1) in the first step, **Build()** takes two table instances as input and produces corresponding dependency models (i.e., the signature vectors) for each unique value, and (2) in the second step, **Match()**, using the dependency models, compares the distances among all pairs of unmapped values, and proposes a ranked list of candidate mappings to users. Incorporating the user's feedback, the dependency models are improved in the **Refine()** step. The matching and refining process repeats until complete mappings are found (or the user stops it).

3. OUR APPROACH

3.1 Modeling Statistical Dependencies

Now, we describe the **Build()** phase of Algorithm 1.

Co-occurrence Frequency Vector Model. Let n be the number of rows in which terms a and b co-occur, and r be the total number of rows in the table. Then, the *co-occurrence frequency* of (a, b) is defined as n/r. For example, in Figure 1(a), the co-occurrence frequency of $(M, Married)$ is $2/4 = 0.5$, while that of $(T.A., Ph.D.)$ is $0/4 = 0$. The *co-occurrence matrix*, C_i, captures pair-wise co-occurrence frequency between all pairs of values in the corresponding table. Table 2 shows a co-occurrence matrix, C, corresponding to Table 1(a). In the table, A1–A9 refer to values {M, F, Professor, TA, Ph.D, M.S, B.S, Married, Single}.

One way to improve the accuracy of co-occurrence model is to weight terms according to their *information content*. That is, rare terms carry more weights when they co-occur than terms that occur frequently (e.g., "male" or "female"). In this work, we use a standard inverse document frequency weighting [6]. Each entry, $C(i,j)$, in the co-occurrence matrix C can be weighted by multiplying the following weight: $W(i,j) = (1 - \frac{k}{N}) * (1 - \frac{l}{N})$ where k is the number of times the term i occurs in the table, l is the number of times the term j occurs in the table, and N is the total number of rows in the table. We incorporated the weighting scheme in all the models for our experimentation, but have not weighted the examples in this paper to retain their simplicity.

Entropy Vector Model. For a given random variable, Shannon's *entropy* [27] is defined as follows. Let X be a discrete random variable on a finite set $\mathcal{X} = \{x_1, \ldots, x_n\}$, with probability distribution function $p(x) = \Pr(X = x)$. Then, the entropy $H(X)$ of X is defined as:

$$H(X) \quad = \quad -\sum_{x \in \mathcal{X}} p(x) \log p(x) \qquad (1)$$

The conditional entropy of a column Y conditioned on a column X is: $H(Y|X) = -\sum_{y \in \mathcal{Y}} \sum_{x \in \mathcal{X}} p(y|x) \log p(y|x)$. In our work, we are interested in matching individual values and not entire columns. We can obtain the entropy of a column conditioned on an individual value as: $H(Y|X = x) = -\sum_{y \in \mathcal{Y}} p(y|x) \log p(y|x)$. Now, when two values x and y co-occur, the entropies of a column conditioned on two values can be similarly computed as: $H(Z|X = x, Y = y) = -\sum_{z \in \mathcal{Z}} p(z|x,y) \log p(z|x,y)$.

Similar to a co-occurrence frequency matrix, we can construct a co-occurrence entropy matrix where the entries, instead of being co-occurrence frequencies, are entropy vectors. If the table has N columns, the entropy vector has $N-2$ elements, each element in the vector corresponding to a column in table. The entropy vector has 2 less elements, because the two columns whose values have been fixed have an entropy of 0 and we take them out. Note that the diagonals of the matrix have a vector of size $N-1$ elements since in this case only one value has been fixed.

3.2 Constructing Signature Vectors of Values

Signature vector are constructed during the **Build()** and the **Refine()** phases of Algorithm 1. Note that our value mapping approach is iterative and interactive. In each iteration, using a combination of statistical dependencies defined in Section 3.1 (i.e., co-occurrence frequency and entropy), the algorithm captures the unique statistical properties of values in a *Signature Vector*. Then algorithm uses a similarity measure between the two vectors to find mappings between the values they represent. We now present a few methods of constructing signature vectors in the following. We denote a signature vector of a value v by $\overrightarrow{sig}(v)$.

1. *Frequency-based Signature Vector*: In this method, the first entry of $\overrightarrow{sig}(v_1)$ is the frequency of the value v_1 itself. Then, for each known value mapping, $x \to y$, a new entry is added to $\overrightarrow{sig}(v_1)$ as follows. Let $x \to y$ be a pair of values mapped during the i^{th} iteration. Without loss of generality, assume that a value v_1 is in the same table as x. Then, in the $(i+1)^{th}$ iteration, $\overrightarrow{sig}(v_1)$ will contain the co-occurrence frequency of (v_1, x) as a new entry. Note that in the first iteration, if no value-mapping is known, then the signature vectors of all values simply contain one entry each — the frequency of the value itself.

   For example, the initial frequency-based signature vector for the value $Married$ of Table 1(a), $\overrightarrow{sig}(Married)$, is [0.75], since the frequency of $Married$ is 0.75. Now, suppose the value mapping $Ph.D. \to D7$ is given. Then, in the next iteration, the algorithm computes the co-occurrence frequency of $Ph.D.$ and $Married$ as $2/4 = 0.5$, and thus modify $\overrightarrow{sig}(Married)$ to [0.75, 0.5]. Furthermore, if another value mapping $M.S. \to D3$ is known, the algorithm again computes the co-occurrence frequency of $M.S.$ and $Married$ as: $1/4 = 0.25$. Thus, the $\overrightarrow{sig}(Married)$ will have [0.75, 0.5, 0.25] after the two value mappings have been processed. The signature vectors for values in the corresponding column in Table 1(b) can also be constructed in the same way.

2. *Entropy-based Signature Vector*: Before any value mapping is known, the signature vector of a value v_1, $\overrightarrow{sig}(v_1)$, is constructed as follows. Select only the rows that have v_1 in them. For these rows, compute the entropy of each column. Then, $\overrightarrow{sig}(v_1)$ consists of all these entropy values. Note that if the table has n columns, the signature vector will be of length $n-1$ consisting of one entry for each column except the column in which v_1 occurs. Similar to the frequency-based signature vector computation, after a value mapping $x \to y$ is known, where x and v_1 are not in the same column, the algorithm augments the signature vector as follows. Assume that v_1 co-occurs with x. Select only the rows containing both x and v_1, and compute the entropy of all the other columns except those containing x and v_1. Add all these entropy values to the $\overrightarrow{sig}(v_1)$ as new entries. Note that in this case, we will be adding $n-2$ values for each value mapping — one entropy value for each column except the entropy values of the columns containing x and v_1.

   Using the same example, we illustrate how entropy-based signature vectors are computed. First, we select the three rows whose status column has the value "Married" in *Status* column of Figure 1(a). Next, we compute the entropy of the values in the remaining columns – $Name$, $Gender$, $Title$, and $Degree$ – for those three rows. Therefore, the entropy-based signature vector, $\overrightarrow{sig}(Married)$, for the first iteration is [1.59, 0.92, 0.92, 0.92]. Subsequently, when we know the value mapping $Ph.D. \to D7$, the algorithm selects the first and the last row — the two rows where "Ph.D." and "Married" co-occur. For these two rows, it then computes the entropies for the three remaining columns – $Name$, $Gender$, and $Title$ – that come to [1, 0, 0]. Therefore, the resulting augmented $\overrightarrow{sig}(Married)$ is [1.59, 0.92, 0.92, 0.92, 1, 0, 0]. Similarly, upon adding the value mapping $M.S. \to D3$, the algorithm selects the third row, and the augmented $\overrightarrow{sig}(Married)$ becomes [1.59, 0.92, 0.92, 0.92, 1, 0, 0, 0, 0, 0]. Note that in this illustration we have a lot of 0's because of the small data-sizes of the selected rows.

3. *Hybrid Signature Vector*: In this scheme, the algorithm initially utilizes the advantages of the entropy-based methods, and after a few matches are found it switches over to the (relatively) inexpensive frequency-based method. This switch can be achieved using

two variations: (a) *Signature Switching Mode*: After the entropy-based iterations, the algorithm keeps the matches obtained in those iterations. The next iterations use frequency-based signature vectors created from the existing matches; and (b) *Mixed Signature Mode*: After the entropy-based iterations, we keep the matches from the previous iterations and retain the signature vectors, but the future augmentation of the signature vector is done using the co-occurrence frequencies. Note that in this case, we are mixing entropy-based and frequency-based values. Therefore, it may be that the entropy values dominate the similarity computations. To reduce this effect, we normalize the values so that all values in the vectors fall in the [0,1] range by dividing each entropy value by $\log(N)$, which is the maximum possible entropy (corresponding to the case when all the values are different) for a column with N values.

4. *Primary Entropy Signature Vector*: The entropy-based signature vector introduced above mainly consists of two parts: 1) the part containing entropies calculated initially while fixing the value in the table that the entropy vector describes, and 2) the part with augmented entropies that are appended as mappings are discovered. We call the first part of the entropy-based signature vector as *Primary Entropy Signature Vector*. Unlike the other models, this method uses only the primary entropy vector throughout the iterations without augmenting new values to the vector along the way.

3.3 Matching Models

This is the **Match()** phase of Algorithm 1. Suppose two columns, s and t, are known to be mapped, where s has a domain of values $\{v_1, ..., v_n\}$ and t has a range of values $\{w_1, ..., w_m\}$. In this step, our goal is to find all mappings from v_i to w_j: that is, $g : v_i \rightarrow w_j$. Suppose, in the i^{th} iteration, all signature vectors are properly created using one of the methods in Section 3.2, and thus there are $n + m$ number of signature vectors created: $\{\overrightarrow{sig}(v_1), ..., \overrightarrow{sig}(v_n)\}$ and $\{\overrightarrow{sig}(w_1), ..., \overrightarrow{sig}(w_m)\}$. Then, we measure all pair-wise distances between two vectors, $dist(\overrightarrow{sig}(v_i), \overrightarrow{sig}(w_j))$ ($1 \leq i \leq n$, $1 \leq j \leq m$) to find the most similar two vectors. In particular, we use the following two simple distance metrics. Given two signature vectors $\overrightarrow{x}$ and $\overrightarrow{y}$, where $\overrightarrow{x} = [a_1, ..., a_n]$ and $\overrightarrow{y} = [b_1, ..., b_n]$, respectively: (1) *Euclidean Distance* is: $dist(\overrightarrow{x}, \overrightarrow{y}) = \sqrt{(a_1 - b_1)^2 + ... + (a_n - b_n)^2}$; and (2) Using the Pearson correlation, *Correlation Distance* is: $dist(\overrightarrow{x}, \overrightarrow{y}) = 1/2*(1 - \frac{covariance(\overrightarrow{x}, \overrightarrow{y})}{\sqrt{covariance(\overrightarrow{x}, \overrightarrow{x}) \times covariance(\overrightarrow{y}, \overrightarrow{y})}})$.

Our algorithms construct the initial signature vectors, and augment them by processing matches in order. That is, it takes the value match between x and y, and augments *both* signature vectors and then proceeds to augment the vectors for the next match. This simple procedure ensures the vectors to be aligned at all times and the similarity computation is meaningful.

Top-k Ranking. After computing all pair-wise distances, the algorithm sorts the candidate mappings based on their distances and proposes the top-k ranked candidate mappings to the user. The algorithm returns the two values mapped, their similarity score (Euclidean or correlation distance), and their *p-value*. The p-value is a common metric used in hypothesis testing [23], and can be computed by transforming the correlation of the two vectors (of size N) to create a t-statistic having $N - 2$ degrees of freedom. In our context, the p-value states the probability of observing a candidate-mapping's correlation *by chance* at the level greater than or equal to the observed correlation. The expert uses a combination of the similarity score and the p-value to get an indication of how good the match is. The size of the top-k window can be set by the user. We have experimented with values of 10 and 20 for k. The user then reviews candidates, and either confirms or denounces some of the proposed mappings.

User Feedback. Instead of finding value mappings in one iteration, our iterative and interactive approaches gradually improve mappings by exploiting positive or negative feedbacks from the users. In particular, we consider four plausible scenarios: (1) when users give always correct feedback to all cases, (2) when users give no feedback, (3) when users give false negatives (denouncing a correct mapping as to be incorrect), and (4) when users give false positives (confirming an incorrect mapping as to be correct).

(1) *Correct Feedback.* When users confirm a candidate mapping, m_p, positively, the mapping will be added to a set of "known" mappings, and thus in the next iteration, the signature vector of unmapped value v will have an entry corresponding to m_p in the vector (thus carrying more statistical information). When users reject a candidate mapping, m_n, the mapping is no longer considered as candidate mappings in the subsequent iterations. The user can also go back and reject any mapping confirmed in any previous iteration.

(2) *No Feedback.* In practice, users can be uncertain for some candidate mappings and may leave out those mappings without giving them feedback. The algorithms will just run normally putting the mappings back into the candidate pool and move on to the next iteration.

(3) *False Negative.* Suppose users may mistakenly confirm a bad mapping or denounce a good mapping. We address these problems in two ways. First, we modified the denounce logic of our algorithms not to throw away a mapping that is denounced but to penalize the mapping by applying a negative weight with an exponential decay as follows:

$$d_i = \begin{cases} d \times (1 - c^{i-t}) & \text{for } i > t > 0 \\ d & \text{otherwise} \end{cases}$$

where d is the distance measure between the two signatures, c is the decay ($c < 1$), i is the current iteration, and t is the last iteration where the mapping was denounced (t is zero if the current mapping has never been denounced). After a mapping is denounced, the mapping's similarity measure in the next iteration will be discounted by the factor defined with c. In the subsequent iterations, the decay will exponentially decrease as the iterations elapse. If the denounced mapping was a true mapping, its correlation value will likely improve over the iterations as more user feedbacks are incorporated, and will eventually have a second chance as the penalty diminishes. Once a denounced mapping returns to the top-k, in practice, the algorithms can notify users of a potential mistake and ask if users want to review the mapping again.

(4) *False Positive.* In order to handle false positive cases where users confirm a bad mapping, we introduced a "kick-out" threshold; we used "p-value $\geq$ 0.05" as the threshold in our experiment. The algorithms now not only update the signatures of unmapped values in each iteration, but also update the signatures of the values in the confirmed mappings and continuously monitor the changes in statistics of them as well. If the statistics of a previously confirmed mapping fell below the threshold after some iterations, the user can be given a warning about the mapping or the mapping can be kicked out and included back in the candidate pool.

Stopping Criteria. Our iterative matching process continues until all true mappings are found or users want to stop the process. In practice, however, the number of true mappings can be much smaller than the sizes of the two tables being matched, and thus large numbers of candidate mappings can still be generated even after all true mappings were found. To avoid unnecessary iterations, we need stopping criteria.

Ideally the iterations should stop right after the last true mapping is found. However, in general, it is not always easy to tell if there will be more mappings in the next iterations or not. Users have to make a subjective decision. Users may decide to stop if the current candidate list does not have any good mappings, and the statistics of all the candidates in the list indicate that they are statistically insignificant. The next iteration's top-k candidates will likely be worse than the current ones, so stopping there perhaps would make sense for users. On the other hand, even if users failed to find a good mapping in an iteration, they might still want to proceed if the statistics of the current top-k candidates were strong. In practice, users may also force iterations stop by explicitly specifying a cut-off point, α (e.g., 0.05, 0.01). When α is set, the algorithms test in each iteration if the minimum p-value of the current candidates is greater than or equal to α, and if so, they stop.

4. VALIDATING THE FRAMEWORK

Set-up: Our algorithms were implemented using Matlab 7.0. Experiments were conducted on a machine running Windows XP Professional with 3Ghz Pentium 4 and 2 GB of memory. We used census data sets obtained from U.S. Census Bureau[1]. For experiments, we selected 8 different groups of records (17 attributes) including 1) all records for "NY" state, 2) all records for "CA", 3) all records for teen ager group, 4) 20s, 5) 30s, 6) 40s, 7) 50s, and finally 8) all army veterans. The number of rows in each data set ranges from 7K to 10K. We sampled 5K tuples from each table for experiments. Table 3 shows the basic statistics of NY and CA tables. For example, there are 23 and 22 unique values in the PRMJIND1 columns of CA and NY tables respectively, and among them 22 values are common. The entropy of the PRMJIND1 column of the CA table (=3.0251) is slightly higher than that of the NY table (=3.0193).

Evaluation Metrics: We assume users only examine the top-k candidates in each iteration, and confirm or denounce all correct and incorrect mappings among the top-k candidates. We iterate through each step until the complete mapping is found. The *precision* of the system can be calculated as: $Precision = \frac{total\ \#of\ correct\ mappings}{total\ \#of\ candidate\ mappings\ presented\ to\ user}$.

[1]http://dataferrett.census.gov/TheDataWeb/index.html

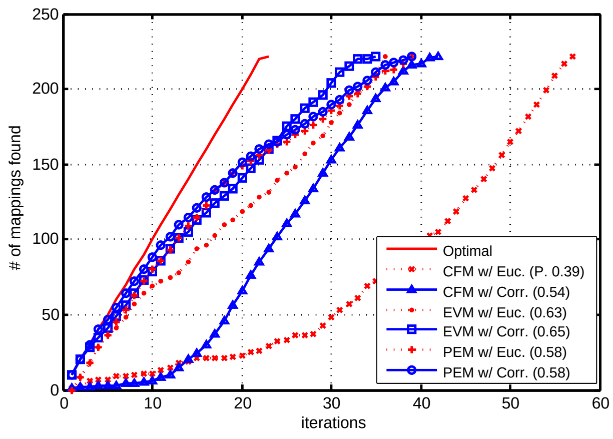

Figure 1: Three models (CFM, EVM, PEM) with Euc and Corr are compared. (top-10 window).

Another important metric for measuring the success of the algorithm is *response time.* The computational complexity of each iteration of the algorithm is bounded by $O(mn^2)$ where m and n are the numbers of entries to be mapped in each table ($n < m$). In each iteration, we compare two signature vectors of size < O(n) (=maximum possible size of a signature vector) mn times. We considered these two metrics, *precision* and *response time*, to evaluate our approach.

Experiments: We present the results in the following order: 1) comparing the base-line models, distance metrics, top-k window sizes, 2) hybrid models, 3) response time, 4) error correction, 5) sensitivity to different data distributions, and finally 6) effects of pre-established mappings.

We also evaluated the effects of term weighting (discussed in Section 3). However, since the weighted models outperformed non-weighted models on average by 15%, in this paper, we presented only the results of the weighted models.

4.1 Effects of Different Models, Similarity Metrics, and top-k Window Sizes

Figure 1 presents the result of mapping over NY and CA with the three dependency models: Co-occurrence Frequency (CFM), Entropy Vector (EVM), and Primary Entropy Model (PEM). Each of the three models was tested twice, using the Euclidean distance (Euc) and the correlation ($Corr$) as a similarity metric. Figure 1 shows the result using top-10 windows. The result using top-20 windows was similar, so we omitted the graph.

The straight line ($Optimal$) on the left in Figure 1 is an imaginary line that depicts a best case mapping scenario where an algorithm's top-10 candidates were correct all the time. In both experiments using top-10 and top-20 windows, EVM outperformed the other models slightly, and the $Corr$ metric outperformed Euc. CFM performed much better with $Corr$ than with Euc while the other two models, PEM and EVM, performed equally well with both Euc and $Corr$. Both CFMs (with Euc and $Corr$) started slow until the number of found mappings reached to some level ($Corr$, 10-20 mappings and Euc, about 30-40 mappings). Then, the two models picked up rapidly in the rate that matches or exceeds those of the other models. Unlike CFMs, both EVM and PEM performed well from the beginning. In

No.	Col. name	# of unique values			Entropies		Description
		CA	NY	Common	CA	NY	
1	HEHOUSUT	5	6	5	0.1552	0.1462	household type
2	HETENURE	3	3	3	1.0488	1.0558	own / rent / null
3	HRNUMHOU	13	13	12	2.8451	2.697	# of household members
4	HUFAMINC	17	16	16	3.6958	3.5968	total family income
5	PEEDUCA	17	17	17	3.2893	3.2759	highest degree earned
6	PEMARITL	7	7	7	2.1347	2.227	marital status
7	PERACE	4	4	4	0.9873	1.0149	race
8	PESEX	2	2	2	1	0.9971	sex
9	PUAFEVER	5	5	5	1.2443	1.1532	army veteran
10	PEMLR	8	8	8	2.1884	2.25	employment status
11	PRFTLF	4	4	4	1.8322	1.8075	full time, part time, etc
12	PEIO1COW	9	8	8	1.7832	1.8056	class of worker (fed., priv., etc.)
13	PRDTIND1	49	49	47	3.4289	3.3758	industry code
14	PRDTOCC1	45	45	45	3.4702	3.4921	occupation code
15	PRMJIND1	23	22	22	3.0251	3.0193	industry - major group
16	PRMJOCC1	14	14	14	2.6812	2.664	occupation - major group
17	PEERNLAB	3	3	3	0.5521	0.5754	union member? (y/n/null)

Table 3: Census table summary.

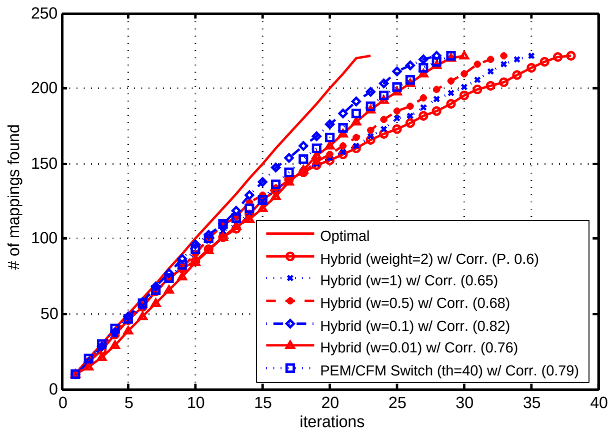

Figure 2: Performance comparison of hybrid approaches using correlation as a similarity metric. (top-10 Window).

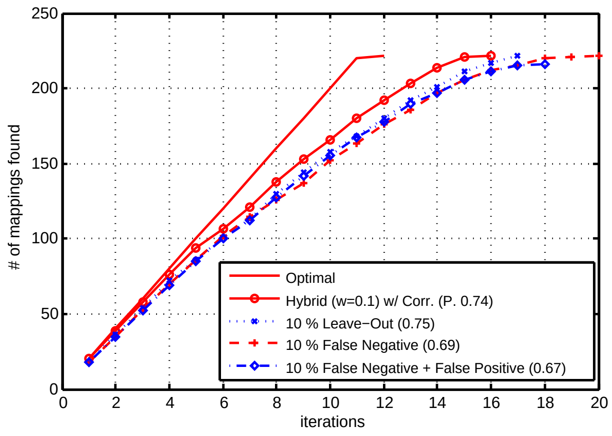

Figure 3: Evaluation of error-correction strategy for handling incorrect or incomplete user feedbacks.

fact, EVM with $Corr$ (EVM-$Corr$ hereafter) and PEM-$Corr$ produced almost perfect candidate lists for initial iterations (first four iterations in Figure 1). Their performance, however, degraded gradually after the initial runs and in the latter stage their growth rate became worse than that of CFM-$Corr$. EVM-$Corr$ was the best performer achieving 65% precision.

4.2 Improving Performance with Hybrid Approaches

We tested two hybrid models, switching and mixed signatures, over the same data sets, and showed the result in Figure 2 (only the top-10 test shown). The PEM/CFM switching model with threshold = 40 (i.e., $PEM/CFM40$) uses PEM until it finds 40 initial mappings and then switches to CFM. Also, five mixed signature models with different weights were tested. Recall that unlike the other models, the mixed signature model contains two different types of values: entropies and co-occurrence frequencies. Because the two different types of values carry different information, we can further tune the hybrid model by applying different weights to the two types of values. We tested five different weights (w=2, 1, 0.5, 0.1, 0.01) applied to the entropy values; that is, if w=1, we weigh the two types of information equally and if w=0.1, we discount the entropy values by $1/10th$.

Among the different weights tried, $w = 0.1$ outperformed the others in both top-10 and top-20 tests. In the top-10 tests, it achieved 82% precision producing more than 8 correct mappings out of 10 candidate mappings on average. In the top-20 tests, it achieved about 74% precision. The PEM/CFM switching model also performed well; it achieved 79% and 74% precision in top-10 and 20 tests, respectively. In summary, the hybrid models were quite successful; Mixed (w=0.1) outperformed the previous top performer, EVM, by 26% and 21% in top-10 and 20 tests, respectively.

4.3 Response Times

We compared the response times of the two top performing algorithms, $Mixed$ and PEM/CFM, for both top-10 and top-20 windows. In top-10 window tests, $Mixed$ completed in 28 iterations – one iteration less than PEM/CFM, while in the top-20 tests, both algorithms finished in 16 iterations. All four algorithms ran on average 2.2 – 2.7 seconds per iteration.

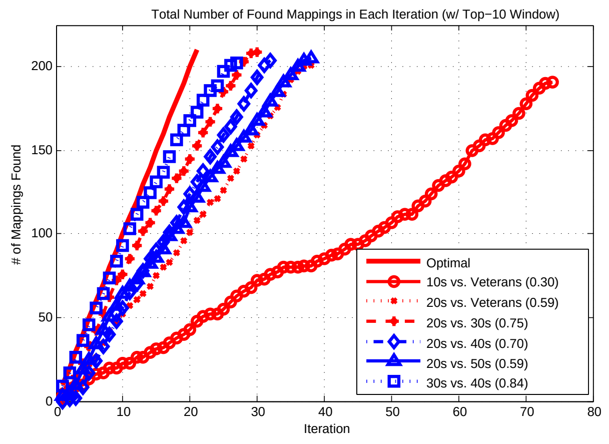

Figure 4: Sensitivity of algorithm against different data distributions.

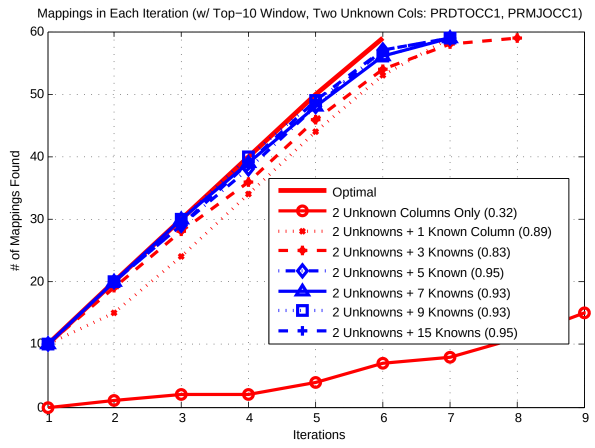

Figure 5: Effects of pre-established mappings.

4.4 Handling Incorrect/Incomplete User Feedback

We tested how our proposed solutions can handle false positives/negatives (i.e., incorrect feedback from users). Figure 3 shows the result of the experiment where we tested the error correction strategies in three scenarios: 10% leave-outs, false negatives, and the combination of false positives and negatives. We used Mixed ($w = 0.1$) for this experiment and compared the results of the scenarios to that of the perfect user scenario reported earlier. In each iteration, we randomly chose 10% of the candidate mappings from the top-k window and applied the errors. For the 10% leave-out test, we simply send the chosen subset back to the algorithm with no feedback. For the 10% false negative test, we assured on average about 10% of the true mappings in each iteration were falsely identified as negative. Lastly, we tried the combination errors where we denounced true mappings (false negatives) and confirmed false mappings (false positives) for the 10% of the candidates.

As shown in Figure 3, 10% leave-out test finished in 17 iterations finding all true mappings while 10% false negative test finished in 20 iterations also finding all true mappings. On the other hand, 10% combination error test failed to find all true mappings and stopped short at the 18th iteration. We forced the algorithm to stop if all mappings that were confirmed in the previous iteration were immediately kicked out in the next iteration. After its 18th iteration, it produced the mapping result with 216 true mappings (out of 222) and seven false mappings. The seven false mappings survived without being kicked out because their p-values were below the threshold. Small numbers of true mappings were also kicked out as their statistics were too weak; those mappings include the ones with very low frequency values (e.g., values only occur once or twice in the entire table).

4.5 Sensitivity of Algorithm Against Different Data Distributions

We tested the sensitivity of our algorithm against the data sets with different data distribution. Figure 4 shows the result obtained using Mixed ($w = 0.1$) over six different pairs of data sets: 1) 10s (this data set contains only teens) vs. army veterans (contains people who have ever served in the army), 2) 20s vs. army veterans, 3) 20s vs. 30s, 4) 20s vs. 40s, 5) 20s vs. 50s, and lastly 6) 30s vs. 40s. The tests over 20s vs. 30s and 30s vs. 40s, representing groups with similar distributions, achieved 75% and 84% accuracy, while the tests over 10s vs. veterans and 20s vs. veterans, representing groups with less similar distributions, achieved only 30% and 59% accuracy, respectively. The 10s vs. veterans test turned out far worse than the rest of the tests. In fact, 10s table represents a completely different distribution from the rest of the tables; teens are very unlikely to own a house, have a job, or be married. Confirming this, the 20s vs. veterans test resulted in a much better result (finished in 38 iterations while 10s-veterans, in 74 iterations). This result suggests that we can expect a good performance from our algorithm when applied to datasets with reasonably similar domains.

4.6 Effects of Pre-Established Mappings

All the above tests were done assuming no pre-established mappings exist between the values (all to all match). In practice, however, there can be some mappings that are obvious because of their textual similarity. We examined the effect of such pre-established mappings to our algorithms. Figure 5 shows the result.

We compared the performance of Mixed ($w = 0.1$) over NY and CA while increasing the number of pre-established mappings. The line at the bottom shows the mapping result of two columns (PRDTOCC1 and PRMJOCC1) across the tables without considering any other columns. It achieved only 32% accuracy.

We then added an extra column (PRDTIND1) with pre-established mapping, to the algorithm. The accuracy of the mapping improved significantly to 89%. As more known columns were added, the accuracy of the result gradually improved up to 95% (when all 17 columns used). The result suggested that even a small number of pre-established mappings can improve the performance of the algorithm significantly. Also suggested is that the algorithm can work even with a small number of common columns across the tables, and hence can be applicable to other related problems such as approximate joins [5, 13].

5. RELATED WORK

There are vast amount of related work to ours – notably (1) *schema matching* (e.g., [2, 7, 9, 14, 16–18, 20, 21]) and (2) *object mapping* (e.g., [1, 3–5, 10, 12, 13, 15, 19, 28]) problems.

(1) In this work, we assumed that we knew the correspondences between the columns across the tables. However, in general settings, such correspondences are not known but have to be found first. To generate such correspondences, various schema matching techniques have been proposed. Some employs Machine Learning (e.g., LSD [9]), rules (e.g., TranScm [21]), Neural Network (e.g., SemInt [17]), structural similarity (e.g. Cupid [18]), or interactive user feedback (e.g., Clio [14]). Recent development (e.g., iMAP [7]) even enables to find not only 1-1, but also more complex *n-m* schema matches (e.g., `name` = concat(`first`, `last`) or `euro` = 1.32 × `dollar`). For a good survey and comparison, see [8, 25]. In particular, our proposals in this paper is an extension of authors' previous attempt [16] to the object mapping problem – in [16], the schema matching problem in the presence of opaque column names and data values are addressed.

(2) Our "value mapping" problem is more closely related to the object mapping problem (i.e., value is the object to map), which is also known as various names in diverse contexts: e.g., record linkage [11, 28], citation matching [19, 24], identity uncertainty [24], merge-purge [15], duplicate detection [1, 22, 26], and approximate string join [5, 13]. Common to all these is the problem to find similar objects (e.g., values, records, tuples, citations). Although different proposals have adopted different approaches to solve the problem in different domains, by and large, they focus on syntactic similarities of objects under comparison. On the other hand, our value mapping solutions can identify mappings where two objects have little syntactic similarity. To cope with such difficulties, we proposed to explore statistical characteristics of objects such as co-occurrence frequency or entropy.

6. CONCLUSION

In this paper, we investigated the *value mapping problem* to locate matching pairs of values from two input database tables. We proposed a two-step, iterative algorithm to discover the mapping. Our algorithm uses the statistical dependency model among values (e.g., value co-occurrence frequency or entropy) as the basis for establishing value mappings. In each iteration, it suggests a top-*k* list of candidate mappings to users. The feedback provided by users is used to improve the mappings suggested in subsequent iterations. As validated through extensive experimentations, our solutions successfully establish the mapping between values across tables even in the presence of opaque data values and thus can be a useful addition to existing data integration techniques.

7. REFERENCES

[1] R. Ananthakrishna, S. Chaudhuri, and V. Ganti. "Eliminating Fuzzy Duplicates in Data Warehouses". In *VLDB*, 2002.

[2] P. Andritsos, R. J. Miller, and P. Tsaparas. Information-theoretic tools for mining database structure from large data sets. In *ACM SIGMOD*, 2004.

[3] M. Bilenko and R. J. Mooney. "Adaptive Duplicate Detection Using Learnable String Similarity Measures". In *ACM KDD*, Washington, DC, 2003.

[4] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani. "Robust and Efficient Fuzzy Match for Online Data Cleaning". In *ACM SIGMOD*, 2003.

[5] W. W. Cohen. "Integration of Heterogeneous Databases Without Common Domains using Queries based on Textual Similarity". In *ACM SIGMOD*, 1998.

[6] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. "Indexing by Latent Semantic Analysis". *J. of the American Society of Information Science*, 41(6):391–407, 1990.

[7] R. Dhamankar, Y. Lee, A. Doan, A. Y. Halevy, and P. Domingos. "iMAP: Discovering Complex Mappings between Database Schemas". In *ACM SIGMOD*, 2004.

[8] H. Do, S. Melnik, and E. Rahm. "Comparison of Schema Matching Evaluations". In *GI-Workshop "Web and Databases"*, Oct. 2002.

[9] A. Doan, P. Domingos, and A. Y. Halevy. "Reconciling Schemas of Disparate Data Sources: A Machine-Learning Approach". In *ACM SIGMOD*, 2001.

[10] A. Doan, Y. Lu, Y. Lee, and J. Han. "Object Matching for Data Integration: A Profile-Based Approach". In *Workshop on Info. Integration on the Web*, 2003.

[11] I. P. Fellegi and A. B. Sunter. "A Theory for Record Linkage". *J. of the American Statistical Society*, 64:1183–1210, 1969.

[12] H. Galhardas, D. Florescu, D. Shasha, and E. Simon. "An Extensible Framework for Data Cleaning". In *IEEE ICDE*, 2000.

[13] L. Gravano, P. G. Ipeirotis, N. Koudas, and D. Srivastava. "Text Joins for Data Cleansing and Integration in an RDBMS". In *IEEE ICDE*, 2003.

[14] M. A. Hernandez, R. J. Miller, and L. M. Haas. "Clio: A Semi-Automatic Tool for Schema Mappiong". In *ACM SIGMOD*, 2001.

[15] M. A. Hernandez and S. J. Stolfo. "The Merge/Purge Problem for Large Databases". In *ACM SIGMOD*, 1995.

[16] J. Kang and J. F. Naughton. "On Schema Matching with Opaque Column Names and Data Values". In *ACM SIGMOD*, San Diego, CA, Jun. 2003.

[17] W.-S. Li and C. Clifton. "SEMINT: A Tool for Identifying Attribute Correspondences in Heterogeneous Databases using Neural Networks". *VLDB J.*, 10(4), Dec. 2001.

[18] J. Madhavan, P. A. Bernstein, and E. Rahm. "Generic Schema Matching with Cupid". In *VLDB*, 2001.

[19] A. McCallum, K. Nigam, and L. H. Ungar. "Efficient Clustering of High-Dimensional Data Sets with Application to Reference Matching". In *ACM KDD*, Boston, MA, 2000.

[20] S. Melnik, H. Garcia-Molina, and E. Rahm. "Similarity Flooding: A Versatile Graph Matching Algorithm and Its Application to Schema Matching". In *IEEE ICDE*, 2002.

[21] T. Milo and S. Zohar. "Using Schema Matching to Simplify Heterogeneous Data Translation". In *VLDB*, 1998.

[22] A. E. Monge. *"Adaptive Detection of Approximately Duplicate Database Records and the Database Integration Approach to Information Discovery"*. PhD thesis, University of California, San Diego, 1997.

[23] D. S. Moore and G. P. McCabe. *"Introduction to the Practice of Statistics"*. From Book News, Inc., 1998.

[24] H. Pasula, B. Marthi, B. Milch, S. Russell, and I. Shpitser. "Identity Uncertainty and Citation Matching". In *Advances in Neural Information Processing Systems*. MIT Press, 2003.

[25] E. Rahm and P. A. Bernstein. "On Matching Schemas Automatically". *VLDB J.*, 10(4), Dec. 2001.

[26] S. Sarawagi and A. Bhamidipaty. "Interactive Deduplication using Active Learning". In *ACM SIGMOD*, 2002.

[27] C. E. Shannon. "A Mathematical Theory of Communication". *Bell System Technical J.*, 1948.

[28] W. E. Winkler. "The State of Record Linkage and Current Research Problems". Technical report, US Bureau of the Census, Apr. 1999.