Learning importance of relationships for reference disambiguation

Dmitri V. Kalashnikov Sharad Mehrotra
Computer Science Department
University of California, Irvine
http://www.ics.uci.edu/~dvk/RelDC

Contents

Abstract

In this paper we address the problem of *reference disambiguation*. Specifically, we consider a situation where entities in the database are referred to using descriptions (e.g., a set of instantiated attributes). The objective of reference disambiguation is to identify the unique entity to which each description corresponds. The key difference between our approach and the traditional techniques is that it analyzes not only object features but also inter-object relationships to improve the disambiguation quality. To analyze relationships our approach utilizes a model that measures the degree of interconnectivity of two entities to each other via chains of relationships that exist between them. Such a model is a key component of the proposed technique. In this paper we concentrate on a method for learning such a model by learning the importance of different relationships from the data itself. This allows our reference disambiguation algorithm to automatically adapt to different datasets being cleaned.

1 Introduction

Recent surveys [1] show that more than 80% of researchers working on data mining projects spend more than 40% of their project time on cleaning and preparation of data. The data cleaning problem often arises when information from heterogeneous sources is merged to create a single database. Many distinct data cleaning challenges have been identified in the literature: dealing with missing data [22], handling erroneous data [23], record linkage [3, 18, 4], and so on. In this paper we address one such challenge which we refer to as *reference disambiguation*[1].

The reference disambiguation problem arises when entities in a database contain references to other entities. If entities were referred to using unique identifiers then disambiguating those references would be straightforward. Instead, frequently, entities are represented using properties/descriptions that may not uniquely identify them leading to ambiguity. For instance, a database may store information about two distinct individuals 'Donald L. White' and 'Donald E. White', both of whom are referred to as 'D. White' in another database. References may also be ambiguous due to differences in the representations of the same entity and errors in data entries (e.g., 'Don White' misspelled as 'Don Whitex'). The **goal** of reference disambiguation is for each reference to correctly identify the unique entity it refers to.

The reference disambiguation problem is related to the problem of *record deduplication* or *record linkage* [18, 4, 3] that often arise when multiple tables (from different data sources) are merged to create a single table. The causes of record linkage and reference disambiguation problems are similar; viz., differences in representations of objects across different data sets, data entry errors, etc. The differences between the two can be intuitively viewed using the relational terminology as follows: while the record linkage problem consists of determining when two records are the same, reference disambiguation corresponds to ensuring that references (i.e., "foreign keys"[2]) in a database point to the correct entities.

Given the tight relationship between the two data cleaning tasks and the similarity of their causes, existing approaches to record linkage can be adapted for reference disambiguation. In particular, *feature-based similarity (FBS)* methods that analyze similarity of record attribute values (to determine whether or not two records are the same) can be used to determine if a particular reference corresponds to a given entity or not. This paper argues that quality of disambiguation can be significantly improved by exploring additional semantic information. In particular, we observe that references occur within a context and define relationships/connections between entities. For instance, 'D. White' might be used to refer to an author in the context of a particular publication. This publication might also refer to different authors, which can be linked to their affiliated organizations etc, forming chains of relationships among entities. Such knowledge can be exploited alongside attribute-based similarity resulting in improved accuracy of disambiguation.

In [25] we have proposed a domain-independent data cleaning approach for reference disambiguation, referred to as Relationship-based Data Cleaning (RelDC), that, unlike traditional techniques, systematically exploits not only features but also relationships among entities for the purpose of disambiguation. RelDC views the database as a graph of entities that are linked to each other via relationships. It first utilizes a feature based method to identify a set of candidate entities (choices) for a reference to be disambiguated. Graph theoretic techniques are then used to discover and analyze relationships that exist between the entity containing the reference and the set of candidates.

The data cleaning algorithm [25] has several limitations that have been identified when it was presented to representatives of intelligence community:

- *Minimizing participation of domain expert.* RelDC requires a presence of a domain analyst. This analyst should first identify which types of relationships are irrelevant for the reference disambiguation purpose. Then the analyst should remove these irrelevant relationship types from the entity-

[1] The reference disambiguation problem has been previously identified in [20] where it was referred to as *cleaning spurious links*.

[2] We are using the term foreign key loosely. Usually, foreign key refers to a unique identifier of an entity in another table. Instead, foreign key above means the set of properties that serve as a reference to an entity.

relationship graph. Even though this task is not expected to be complex for a domain expert, this step was identified as being undesirable: rather techniques should be designed that achieve this goal automatically by analyzing the dataset being processed.

- *Learning the importance of connections from data.* RelDC operates by analyzing various connections (which correspond to paths in the entity-relationship graph) that exist between certain pairs of nodes. It has a *model* that weighs different connections to decide which connection is better. A simple example of such a model can be the length of the connection, i.e. a shorter connection is always better then a longer connection – [25] uses a more advanced model. The disadvantage of the model in [25] is that it is not adaptable: it does not learn how to weigh connections from data itself. In particular, it is not capable of handling the situation where a presence of certain type of connection between two entities implies that those two entities are not related (anti-correlation). Even though this model has been shown to work well for several datasets, the lack of such an adaptiveness has been identified as a weakness of the approach.

The first drawback can be addressed by a simple algorithm provided in Appendix A. The primary contributions of this paper is a method which addresses the second drawback identified above thus making RelDC adaptable to various datasets. The rest of this paper is organized as follows. Section 2 contains the related work. Section 3 presents a motivational example. In Section 4, we precisely formulate the problem of reference disambiguation and introduce notation necessary to explain the RelDC approach. Section 5 sketches the algorithm used by RelDC and presents a method that allows RelDC to automatically adapt to the dataset being cleaned. A summary of the experimental results is presented in Section 6. Finally, Section 7 concludes the paper.

2 Related Work

Many research challenges have been explored in the context of data cleaning in the literature: dealing with missing data, handling erroneous data, record linkage, and so on. The closest to the problem of reference disambiguation addressed in this paper is the problem of record linkage. The importance of record linkage is underscored by the large number of companies, such as Trillium, Vality, FirstLogic, DataFlux, which have developed (domain-dependent) record linkage solutions.

Researchers have also explored domain-independent techniques [28, 11, 14, 33, 2, 24]. Their work can be viewed as addressing two challenges: (1) improving similarity function, as in [3, 18]; and (2) improving efficiency of linkage, as in [4]. Typically two-level similarity functions are employed to compare two records. First, such a function computes attribute-level similarities by comparing values in the same attributes of two records. Next the function combines the attribute-level similarity measures to compute the overall similarity of two records. A recent trend has been to employ machine learning techniques, e.g. SVM, to learn the best similarity function for a given domain [3]. Many techniques have been proposed to address the efficiency challenge as well: techniques that use specialized data structures (e.g. specialized indexes [4]), merge/purge, sorting and window based techniques.

The above mentioned domain-independent techniques utilize only attributes for the purpose of cleaning. The only work we are aware of that can be viewed as using relationships for data cleaning is [2, 20]. In [2] Ananthakrishna et al. employs co-occurrence similarity function to compare contextual attributes for a case with hierarchical relationships. In [20] Lee et al. develop a association-rules mining based method where similarity of records is determined by similarity of the context attributes. Also, in DKDM04, Getoor et al. proposes a method that solves the author matching problem using the notion of context and clustering techniques. The method requires "seeding" of clusters which makes it of limited applicability in practice. To summarize, previous approaches to data cleaning that explored relationships have either addressed other data cleaning problems (not reference disambiguation) or have only considered specific types of relationships (e.g. hierarchy). None of those approaches *discovers* relationship chains and their analysis is limited to directly linked entities. RelDC is the first data cleaning framework that employs a

systematic analysis of relationships for the purpose of cleaning.

3 Motivation for analyzing relationships

In this section we will use an instance of the "author matching" problem to illustrate that exploiting relationships among entities can improve the quality of reference disambiguation. We will also schematically describe one approach that analyzes relationships in a systematic domain-independent fashion.

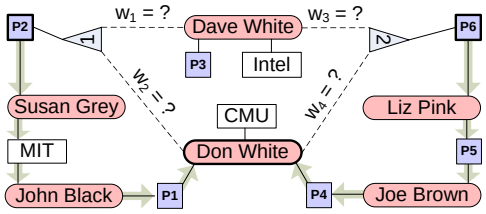

Figure 1: Graph for the publications example

Consider a database about *authors* and *publications*. Authors are represented in the database using the attributes ⟨`id`, `authorName`, `affiliation`⟩ and information about papers is stored in the form ⟨`id`, `title`, `authorRef1`, `authorRef2`,..., `authorRefN`⟩. Consider a toy database consisting of the *author* and *publication* records shown in Figures 2 and 3.

⟨A_1, 'Dave White', 'Intel'⟩
⟨A_2, 'Don White', 'CMU'⟩
⟨A_3, 'Susan Grey', 'MIT'⟩
⟨A_4, 'John Black', 'MIT'⟩
⟨A_5, 'Joe Brown', unknown⟩
⟨A_6, 'Liz Pink', unknown⟩

Figure 2: *author* records

⟨P_1, 'Databases ...', 'John Black', 'Don White'⟩
⟨P_2, 'Multimedia ...', 'Sue Grey', **'D. White'**⟩
⟨P_3, 'Title3 ...', 'Dave White'⟩
⟨P_4, 'Title5 ...', 'Don White', 'Joe Brown'⟩
⟨P_5, 'Title6 ...', 'Joe Brown', 'Liz Pink'⟩
⟨P_6, 'Title7 ...', 'Liz Pink', **'D. White'**⟩

Figure 3: *publication* records

The goal of the author matching problem is to identify for each `authorRef` in each paper the correct author it refers to.

We can use existing feature-based similarity (FBS) techniques to compare the description contained in each `authorRef` in papers with values in `authorName` attribute in authors. This would allow us to resolve almost every `authorRef` references in the above example. For instance, such methods would identify that 'Sue Grey' reference in P_2 refers to A_3 ('Susan Grey'). The only exception will be 'D. White' references in P_2 and P_6: 'D. White' could match either A_1 ('Dave White') or A_2 ('Don White').

Perhaps, we could disambiguate the reference 'D. White' in P_2 and P_6 by exploiting additional attributes. For instance, the titles of papers P_1 and P_2 might be similar while titles of P_2 and P_3 might not, suggesting that 'D. White' of P_2 is indeed 'Don White' of paper P_1. We next show that it may still be possible to disambiguate the references 'D. White' in P_2 and P_6 by analyzing relationships among entities even if we are unable to disambiguate the references using title (or other attributes).

First, we observe that author 'Don White' has co-authored a paper (P_1) with 'John Black' who is at MIT, while the author 'Dave White' does not have any co-authored papers with authors at MIT. We can use this observation to disambiguate between the two authors. In particular, since the co-author of 'D. White' in P_2 is 'Susan Grey' of MIT, there is a higher likelihood that the author 'D. White' in P_2 is 'Don White'. The reason is that the data suggests a connection between author 'Don White' with MIT and an absence of it between 'Dave White' and MIT.

Second, we observe that author 'Don White' has co-authored a paper ($P4$) with 'Joe Brown' who in turn has co-authored a paper with 'Liz Pink'. In contrast, author 'Dave White' has not co-authored any papers with either 'Liz Pink' or 'Joe Brown'. Since 'Liz Pink' is a co-author of P_6, there is a higher likelihood that 'D. White' in P_6 refers to author 'Don White' compared to author 'Dave White'. The reason is that often co-author networks form groups/clusters of authors that do related research and may publish with each other. The data suggests that 'Don White', 'Joe Brown' and 'Liz Pink' are part of the cluster, while 'Dave White' is not.

At first glance, the analysis above (used to disambiguate references that could not be resolved using conventional feature-based techniques) may seem domain specific. A general principle emerges if we view

the database as a graph of inter-connected entities (modeled as nodes) linked to each other via relationships (modeled as edges). Figure 1 illustrates the entity-relationship graph corresponding to the toy database consisting of *authors* and *papers* records. In the graph, entities containing references are linked to the entities they refer to. For instance, since the reference 'Sue Grey' in P_2 is unambiguously resolved to author 'Susan Grey', paper P_2 is connected by an edge to author A_3. Similarly, paper P_5 is connected to authors A_5 ('Joe Brown') and A_6 ('Liz Pink'). The ambiguity of the references 'D. White' in P_2 and P_6 is captured by linking papers P_2 and P_6 to both 'Dave White' and 'Don White' via two "choice nodes" (labeled '1' and '2' in the figure). These "choice nodes" represent the fact that the reference 'D. White' refers to either one of the entities linked to the choice nodes.

Given the graph view of the toy database, the analysis we used to disambiguate 'D. White' in P_2 and P_6 can be viewed as an application of the following general principle:

Context Attraction Principle (CAP): *If reference r made in the context of entity x refers to an entity y_j whereas the description provided by r matches multiple entities $y_1, y_2, \ldots, y_j, \ldots, y_N$, then x and y_j are likely to be more strongly connected to each other via chains of relationships than x and y_l ($l = 1, 2, \ldots, N$; $l \neq j$).* □

Let us now get back to the toy database. The first observation we made, regarding disambiguation of 'D. White' in P_2, corresponds to the presence of the following path (i.e., *relationship chain* or *connection*) between the nodes 'Don White' and P_2 in the graph: $P_2 \rightarrow$ 'Susan Grey' $\rightarrow$ 'MIT' $\rightarrow$ 'John Black' $\rightarrow P_1 \rightarrow$ 'Don White'. Similarly, the second observation, regarding disambiguation of 'D. White' in P_6 as 'Don White', was based on the presence of the following path: $P_6 \rightarrow$ 'Liz Pink' $\rightarrow P_5 \rightarrow$ 'Joe Brown' $\rightarrow P_4 \rightarrow$ 'Don White'. There were no paths between P_2 and 'Dave White' or between P_6 and 'Dave White' (if we ignore '1' and '2' nodes). So, after applying the CAP principle, we concluded that the reference 'D. White' in both cases probably corresponded to the author 'Don White'. In general, there could have been paths not only between P_2 (P_6) and 'Don White' but also between P_2 (P_6) and 'Dave White'. In that case, to determine if 'D. White' is 'Don White' or 'Dave White' we should have been able to measure whether 'Don White' or 'Dave White' is more strongly connected to P_2 (P_6).

The generic approach therefore first *discovers connections* between the entity, in the context of which the reference appears, and the matching candidates for that reference. It then *measures the connection strength* of the discovered connections in order to give preference to one of the matching candidates.

While it has been proven in [25] that the CAP principle does hold over real data sets and that it is possible to design a generic solution to exploiting relationships for disambiguation, the challenge remains of making this solution automatically adaptable to different datasets being cleaned. In this paper we address this challenge. We achieve this by presenting an algorithm that learns how to weigh different connections from the data itself. Before we present this algorithm, we first develop notation and concepts needed to explain our approach in Section 3.

4 Notation and problem definition

In this section we first develop notation and then formally define the problem of reference disambiguation.

4.1 References

Let $\mathcal{D}$ be the database which contains references that are to be resolved. Let X be the set of all entities[3] in $\mathcal{D}$: $X = \{x_1, x_2, \ldots, x_{|X|}\}$. Each entity x_i consists of a set of properties and of a set of n_{x_i} ($n_{x_i} \geq 0$) references $r_{i1}, r_{i2}, \ldots, r_{in_{x_i}}$. Each reference r_{ik} is essentially a description and may itself consist of one or more attributes. For instance, in the example from Section 3, *paper* entities contain *one-attribute* `authorRef` references in the form ⟨*author name*⟩. If, besides author names, author affiliation were also

[3]Entities here have essentially the same meaning as in the standard E/R model.

stored in the *paper records*, then `authorRef` references would have consisted of two attributes – ⟨*author name*, *author affiliation*⟩.

Choice set. Each reference r_{ik} semantically refers to a single specific entity in X which we denote by r^*_{ik}. The description provided by r_{ik} may, however, match a set of one or more entities in X. We refer to this set as the *choice set* of reference r_{ik} and denote it by S_{ik}. The choice set consists of all the entities that r_{ik} could potentially refer to. We assume S_{ik} is given for each r_{ik}. If it is not given, we assume a feature-based similarity approach is used to construct S_{ik} by choosing all of the candidates such that FBS similarity between them and r_{ik} exceed a given threshold. Set S_{ik} consists of $|S_{ik}|$ elements $y_{ik1}, y_{ik2}, \ldots, y_{ik|S_{ik}|}$: $S_{ik} = \{y_{ik1}, y_{ik2}, \ldots, y_{ik|S_{ik}|}\}$.

To avoid using three indexes all the time, such as $ik2$ in y_{ik2}, we will simplify notation. We will present most of the material in the context of r_{ik} reference, and will use N to denote $|S_{ik}|$ and y_j to denote y_{ikj}. That is, we always assume S_{ik} has N (i.e., $N = |S_{ik}|$) elements $y_1, y_2, \ldots, y_N$: $S_{ik} = \{y_1, y_2, \ldots, y_N\}$.

4.2 The entity-relationship graph

RelDC views the resulting database $\mathcal{D}$ as an undirected entity-relationship graph[4] $G = (V, E)$, where V is the set of nodes and E is the set of edges. The set of nodes V is comprised of two sets $V = V^r \cup V^c$: the set of *regular* nodes V^r and the set of *choice* nodes V^c. Each regular node in V^r corresponds to an entity and each edge in E to a relationship.Choice nodes will be defined later. Notation v_{x_i} or v_i denotes the vertex in G that corresponds to entity $x_i \in X$. Note that if entity u contains a reference to entity v, then the nodes in the graph corresponding to u and v are linked since a reference establishes a relationship between the two entities. For instance, `authorRef` reference from paper P to author A corresponds to "A writes P" relationship.

In the graph G, edges have weights, nodes do not have weights. Each edge weight is a real number in $[0, 1]$, which reflects the degree of confidence the relationship, corresponding to the edge, exists. For instance, in the context of our author matching example, if we are 100% confident 'John Black' is affiliated with MIT, then we assign weight of 1 to the corresponding edge. But if we are only 80% confident, we assign the weight of 0.80 to that edge. By default all weights are equal to 1. Notation "edge label" means the same as "edge weight".

References and linking. If S_{ik} has only one element, then r_{ik} is resolved to y_1, and graph G contains an edge between v_i and v_{y_1}. This edge is assigned a weight of 1 to denote that the algorithm is 100% confident that r^*_{ik} is y_1.

If S_{ik} has more than 1 elements, then graph G contains a **choice node** v^*_{ik}, as shown in Figure 4, to reflect the fact that r^*_{ik} can be one of $y_1, y_2, \ldots, y_N$. Node v^*_{ik} is linked with node v_i via edge (v_i, v^*_{ik}). Node v^*_{ik} is also linked with N nodes $v_{y_1}, v_{y_2}, \ldots, v_{y_N}$, for each y_j in S_{ik}, via edges $e_{ikj} = (v^*_{ik}, v_{y_j})$ $(j = 1, 2, \ldots, N)$. To simplify notation, we will use e_j to denote e_{ikj}, that is $e_j = e_{ikj}$. Nodes $v_{y_1}, v_{y_2}, \ldots, v_{y_N}$ are called the *options* of choice v^*_{ik}. Edges $e_1, e_2, \ldots, e_N$ are called the *option-edges* of choice v^*_{ik}. The weights of option-edges are called *option-edge weights* or simply *option weights*. The weight of edge (v_i, v^*_{ik}) is 1. Each weight w_{ikj} of edge e_{ikj} $(j = 1, 2, \ldots, N)$ is undefined initially. We will use w_j as a notation for w_{ikj}: $w_j = w_{ikj}$. Since option-edges $e_1, e_2, \ldots, e_N$ represent mutually exclusive alternatives, the sum of their weights should be 1: $w_1 + w_2 + \cdots + w_N = 1$.

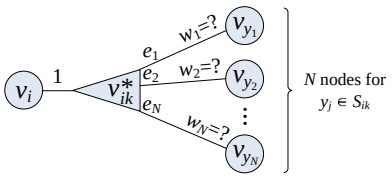

Figure 4: A choice node

4.3 The objective of reference disambiguation

To *resolve* reference r_{ik} means to choose one entity y_j from S_{ik} in order to determine r^*_{ik}. If entity y_j is chosen as the outcome of such a disambiguation, then r_{ik} is said to be *resolved to* y_j or simply *resolved*. Reference r_{ik} is said to be *resolved correctly* if this y_j is r^*_{ik}. Notice, if S_{ik} has just one element y_1 (i.e.,

[4] A standard entity-relationship graph can be visualized as an E/R schema of the database that has been *instantiated* with the actual data.

$N = 1$), then reference r_{ik} is automatically resolved to y_1. Thus reference r_{ik} is said to be *unresolved* or *uncertain* if it is not resolved yet to any y_j and also $N > 1$.

From the graph theoretic perspective, to resolve r_{ik} means to assign weights of 1 to one edge e_j ($1 \leq j \leq N$) and assign weights of 0 to the other $(N-1)$ edges $e_1, e_2, \ldots, e_{j-1}, e_{j+1}, \ldots, e_N$. This will indicate that the algorithm chooses y_j as r^*_{ik}.

The **goal** of reference disambiguation is to resolve all references as accurately as possible, that is, for each reference r_{ik} to correctly identify r^*_{ik}. The *accuracy* of reference disambiguation is the fraction of references being resolved that are resolved correctly.

5 Adaptive solution

We now have developed all the concepts and notation needed to explain RelDC approach for reference disambiguation. The actual procedure used by RelDC in [25] is complex and we will significantly simplify the discussion of it here, emphasizing only the key points.

Input to RelDC is the entity-relationship graph G discussed in Section 3 in which nodes correspond to entities and edges to relationships. We assume that feature-based similarity approaches have been used in constructing the graph G. The choice nodes are created only for those references that could not be disambiguated using only attribute similarity. RelDC will exploit relationships for further disambiguation and will output a resolved graph G in which each entity is fully resolved.

RelDC resolves references based on context attraction principle that was discussed in Section 2. Crucial to the principle is the notion of *connection strength* between two entities x_i and y_j (denoted $c(x_i, y_j)$ which captures how strongly x_i and y_j are connected to each other through relationships. We will discuss how to measure $c(x_i, y_j)$ shortly: we will sketch the non-adaptive model for computing $c(x_i, y_j)$ from [25] and then propose a model that can automatically adapt to different datasets.

Given the concept of $c(x_i, y_j)$, we now can state the main idea of the disambiguation algorithm. To disambiguate a reference r_{ik} the algorithm essentially computes the connection strength $c(x_i, y_j)$ for each $y_j \in S_{ik}$ and then chooses such y_j as r^*_{ik} for which $c(x_i, y_j)$ is the largest.

5.1 Previous (non-adaptive) connection strength model

The connection strength $c(u, v)$ between nodes u and v should reflect how strongly nodes u and v are related to each other via relationships in the graph G. In [25], computation of $c(x_i, y_j)$ consists of two phases. The first phase discovers connections between x_i and y_j. The second phase computes/measures the strength in connections discovered by the first phase.

First phase: discovering connections. In general there can be many connections between v_i and v_{y_j} in G. Intuitively, many of those (e.g., very long ones) are not very important. To capture most important connections while still being efficient, instead of discovering all paths, the algorithm discovers only the set of all L-short simple paths $\mathcal{P}_L(x_i, y_j)$ between nodes v_i and v_{y_j} in graph G. A path is *L-short* if its length is no greater than parameter L. A path is *simple* if it does not contain duplicate nodes. It turns out that, because of choice nodes in G, certain discovered paths cannot semantically exist and this they are ignored in computations.

Second phase: measuring connection strength. In [25] $c(u, v)$ is modeled as the probability to reach v from u via a random walk in G by following only L-short simple paths, such that the probability to follow an edge is proportional to the weight of the edge. It can be shown that such defined $c(u, v)$ can be compute as the sum of the connection strength $c(p)$ of each path p from $\mathcal{P}_L(u, v)$, i.e.

$$c(u, v) = \sum_{p \in \mathcal{P}_L(u,v)} c(p). \tag{1}$$

In this formula $c(p)$ is computed as the probability to follow path p in G.

5.2 Adaptive connection strength model

While [25] imposes a specific formula for computing connection strength, it is desirable to derive a method for computing connection strength that would automatically adjust to different datasets being cleaned. But first we should understand what does it mean to make a model for $c(u, v)$ adaptive.

5.2.1 The ideal connection strength model

Let us assume that training datasets are available in which for each unresolved reference r_{ik} not only its choice set S_{ik} is known but also which y_j from S_{ik} is r^*_{ik}. Recall that RelDC utilizes the CAP principle for disambiguation: it chooses such y_j from S_{ik} as r^*_{ik} which maximizes $c(x_i, y_j)$. Consequently, the ideal model for computing $c(u, v)$ should be constructed such that, when resolving each reference r_{ik} the following must hold:

$$c(x_i, y_j) > c(x_i, y_l) \text{ for } y_j = r^*_{ik} \ (l = 1, 2, \ldots, |S_{ik}|; l \neq j). \tag{2}$$

The goal of designing such an ideal $c(u, v)$ model might not be attainable, e.g., one of the reasons is because the CAP is expected to work most of the time, but not always. So the goal is to design a model for computing $c(x_i, y_j)$ for which (2) would hold most of the time.

5.2.2 A linear adaptive connection strength model

In this section we discuss a connection strength model that follows the criteria for a good $c(x_i, y_j)$ model identified in the previous section.

Notice that the formula used by the non-adaptive model (1) computes $c(u, v)$ by summing up the connection strength $c(p)$ of each individual path p between nodes u and v. While it does characterize $c(u, v)$ by paths that exists between nodes u and v, it has the drawback that it employs a non-adaptable function to compute the connection strength $c(p)$ of a given path p. This function, given path p in a graph G, maps it into a non-negative real number. To do so, the function makes some reasonable assumptions of what the connection strength should be in general for a given path p in a given graph G, however for a given dataset these assumptions might not be valid or simply a better formula can be derived.

Assume that we can classify each path that the algorithm finds in graph G during disambiguation into a finite set of path types $S_T = \{T_1, T_2, \ldots T_n\}$. That is there is a function $T()$ that, given a path p and graph G, can map this path to one of those types: $T(p, G) \rightarrow S_T$. Suppose that those types are such that if two different paths are of the same type then those two paths are identical from the point of view of disambiguation. Then, for given nodes u and v we can characterize connections between them via a path-type count vector $T_{uv} = (c_1, c_2, \ldots, c_n)$, where c_i $(i = 1, 2, \ldots, n)$ is a non-negative integer that shows the number of paths of type T_i between nodes u and v. In other words, if we treat those paths as features, each $c(u, v)$ can be characterized by the count of each feature.

The problem now can be formulated as finding a model that would allow us to compute $c(u, v)$ as a function of feature-count vector for u-v connections (i.e., T_{uv}) such that (2) holds on the training set. That is, the goal is to be able to (a) compute $c(u, v)$ as $c(u, v) = f(c_1, c_2, \ldots, c_n)$; and (b) $f()$ should be computed according to an (adaptive) connection strength model.

Formula (1) suggests that connection strength of individual paths can be summed up to obtain the total connection strength. Therefore, one possible solution for an adaptive model is to assign weight w_i to each path type T_i $(i = 1, 2, \ldots, n)$ and compute $c(u, v)$, given count-vector of features T_{uv}, as

$$c(u, v) = c_1 w_1 + c_2 w_2 + \ldots + c_n w_n, \text{ for } T_{uv} = (c_1, c_2, \ldots, c_n). \tag{3}$$

The key idea is to assign those weights correctly, i.e. learn them from the data. This can be achieved by the following procedure which converts this learning task into solving a linear programming problem.

First, we will normalize weights to $[0, 1]$ domain by requiring that

$$0 \leq w_i \leq 1, \ (i = 1, 2, \ldots, n). \tag{4}$$

If we want to also consider anti-correlation (where a presence of a path of a given type implies that the two entities are not related) then the domain should be changed from $[0, 1]$ to $[-1, 1]$.

Next, for each reference r_{ik} we can construct the set of inequalities (2) such that each $c(u, v)$ is computed according to (3). At this point we need to solve the linear programming problem where the goal is to find a combination of weights $(w_1, w_2, \ldots, w_n)$ such that constraints (4) and (2) are satisfied.

The above solution however needs to be adjusted. The problem is that the CAP principle works most of the time, but not always, consequently (2) might not always hold and hence our linear programming problem might not have a solution in its present form. This complication can be resolved by letting each equation in (2) have its own tolerance δ_{ikj} (a non-negative real number) and then requiring that the sum of each tolerances be minimized:

$$\begin{array}{l} c(x_i, y_j) + \delta_{ikj} > c(x_i, y_l) \text{ for } y_j = r^*_{ik} \ (l = 1, 2, \ldots, |S_{ik}|; l \neq j) \\ \delta_{ikj} \geq 0 \text{ (for all } i, k, j) \\ \text{Goal: mimimize } \sum_{ikj} \delta_{ikj}. \end{array} \quad (5)$$

By construction (5) always has a solution. Given that linear programming problems can be solved efficiently [15], we have specified a method for constructing adaptive $c(u, v)$ model by learning weights of path types from data itself.

Classifying paths into types. To finalize the discussion we should explain how $T(p, G) \rightarrow S_T$ function, that maps paths into a finite set of types, can be implemented in practice. There can be many ways to provide this mapping and we do not insist on a particular one. Recall, the requirement for creating path types is the following. Path types should be such that the disambiguation algorithm should be able to consider different paths of the same type as identical features. That is, if p_1 and p_2 are of the same type, then p_1 is no better and no worse than p_2 from the point of view of the disambiguation algorithm.

The approach that we use for classifying paths is the following. Each path can be viewed as a sequence of edges $\langle E_1, E_2, \ldots, E_k \rangle$. Each edge E_i has its type T_{E_i}, e.g. in the toy example from Section 3 we had edges of two types: author-paper and author-organization. We have chosen to represent the path type of a path via the types of its edges: $\langle T_{E_1}, T_{E_2}, \ldots, T_{E_k} \rangle$. We can do so for the following reason. Consider the author matching problem again. Assume we are trying to determine whether author A_1 or A_2 wrote paper P and we found two connections between P and A_1, and P and A_2: $P \rightarrow A_3 \rightarrow U_1 \rightarrow A_1$ and $P \rightarrow A_4 \rightarrow U_2 \rightarrow A_2$, where A_3 and A_4 are some authors and U_1 and U_2 are two different universities. This connection are similar – both paths while being different are of the same nature: they are "from a paper to an author to a university to an author" and there is little reason to assume that one path is better then the other.

6 Discussion

Notice, the problem we solved is different from classical classification tasks [31]. In a classical classification task each objects is described by a feature vector which captures if a given feature is absent or present in this objects. Based on those features the goal is to classify the objects in one of the classes. In our case we have a feature count-vector which tells us not only whether a feature is present or absent, but also the count of that feature. Out task is different as well: given a *group* (i.e., not all) of feature count vectors to decide which vector is the 'best'. The same vector can be the 'best' in one group and the 'worst' in another. The problem is further complicated because the CAP principle works most of the time but not always, that is why the first intermediate method we have devised for learning weight might not work in general. Hence in Section 5.2 we have also developed a technique to address this issue.

7 Conclusion

In this paper we have presented an algorithm for learning the importance of connections from the data for the purpose of reference disambiguation. This algorithm allows RelDC (the highly accurate domain-

independent reference disambiguation framework that relies on analyzing relationships) to automatically adapt to various datasets (thus removing the requirement of a "smart analyst"). This adaptiveness can have a positive impact on both the efficiency and accuracy of RelDC. The efficiency improves because the algorithm can remove irrelevant relationship types from the graph making it less dense and also RelDC can be specified to discover only important relationships (those with high weights) thus reducing the search space. The accuracy can improve because the algorithm learns formula from data itself rather than simply using some fixed assumptions of what this formula should be.

As our future work we plan to apply similar techniques to the problem of record linkage. Another promising direction is to employ space and time attributes, which are frequently found in many datasets, to improve the quality of disambiguation.

References

[1] Knowledge Discovery. `http://www.kdnuggets.com/polls/2003/data_preparation.htm`.

[2] R. Ananthakrishna, S. Chaudhuri, and V. Ganti. Eliminating fuzzy duplicates in data warehouses. In *Proc. VLDB*, 2002.

[3] M. Bilenko and R. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *SIGKDD*, 2003.

[4] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani. Robust and efficient fuzzy match for online data cleaning. In *Proc. of ACM SIGMOD Conf.*, 2003.

[5] P. Christen, T. Churches, and J. X. Zhu. Probabilistic name and address cleaning and standardization. The Australasian Data Mining Workshop, 2002.

[6] W. Cohen, H. Kautz, and D. McAllester. Hardening soft information sources. In *Proc. of ACM SIGKDD Conf.*, 2000.

[7] W. W. Cohen. Integration of heterogeneous databases without common domains using queries based on textual similarity. In *Proc. of ACM SIGMOD Conf.*, 1998.

[8] W. W. Cohen, P. Ravikumar, and S. E. Fienberg. A comparison of string distance metrics for name-matching tasks. IIWeb Workshop, 2003.

[9] W. W. Cohen and J. Richman. Learning to match and cluster large high-dimensional data sets for data integration. In *Proc. of ACM SIGKDD Conf.*, 2002.

[10] C. Faloutsos, K. S. McCurley, and A. Tomkins. Fast discovery of connection subgraphs. In *Proc. of SIGKDD*, 2004.

[11] I. Fellegi and A. Sunter. A theory for record linkage. *Journal of Amer. Statistical Association*, 64(328):1183–1210, 1969.

[12] L. Getoor. Multi-relational data mining using probabilistic relational models: research summary. In *Proceedings of the First Workshop in Multi-relational Data Mining*, 2001.

[13] L. Gravano, P. Ipeirotis, H. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate string joins in a database (almost) for free. In *Proc. of VLDB Conf.*, 2001.

[14] M. Hernandez and S. Stolfo. The merge/purge prob- lem for large databases. In *Proc. of SIGMOD*, 1995.

[15] F. Hillier and G. Lieberman. *Introduction to operations research*. McGraw-Hill, 2001.

[16] M. Jaro. Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. *Journal of the American Statistical Association*, 84(406), 1989.

[17] M. Jaro. Probabilistic linkage of large public health data files. *Statistics in Medicine*, 14(5-7), Mar-Apr 1995.

[18] L. Jin, C. Li, and S. Mehrotra. Efficient record linkage in large data sets. In *Proc. of DASFAA Conf.*, 2003.

[19] e. a. L. De Raedt. Three companions for data mining in first order logic. In *Dzeroski, S. and Lavrac, N., ed. Relational Data Mining*. Springer-Verlag, 2001.

[20] M. Lee, W. Hsu, and V. Kothari. Cleaning the spurious links in data. *IEEE Intelligent Systems*, Mar-Apr 2004.

[21] M. Lee, H. Lu, T. Ling, and Y. Ko. Cleansing data for mining and warehouse. In *Proc. of DEXA Conf.*, 1999.

[22] R. Little and D. Rubin. *Statistical Analysis with Missing Data*. John Wiley and Sons, 1986.

[23] J. Maletic and A. Marcus. Data cleansing: Beyond integrity checking. In *Proc. of Conf. on Information Quality*, 2000.

[24] A. K. McCallum, K. Nigam, and L. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *Proc. of ACM SIGKDD Conf.*, 2000.

[25] S. Mehrotra, D. Kalashnikov, and Z. Chen. Exploiting relationships for domain-independent data cleaning. *In Submission,* 2004. An extended version is at `www.ics.uci.edu/~dvk/RelDC/TR/TR-RESCUE-04-20.pdf`.

[26] A. E. Monge and C. Elkan. The field matching problem: Algorithms and applications. In *Proc. of SIGKDD Conf.*, 1996.

[27] A. E. Monge and C. P. Elkan. An efficient domain-independent algorithm for detecting approximately duplicate database records. In *Proc. of SIGMOD Wshp. on Research Issues on Data Mining and Knowledge Discovery*, 1997.

[28] H. Newcombe, J. Kennedy, S. Axford, and A. James. Automatic linkage of vital records. *Science*, 130:954–959, 1959.

[29] H. Pasula, B. Marthi, B. Milch, S. Russell, and I. Shpitser. Identity uncertainty and citation matching. In Advances in Neural Processing Systems 15, 2002.

[30] E. Ristad and P. Yianilos. Learning string edit distance. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 20(5):522–532, May 1998.

[31] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2002.

[32] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *Proc. of ACM SIGKDD Conf.*, 2002.
[33] S. Tejada, C. A. Knoblock, and S. Minton. Learning domain-independent string transformation weights for high accuracy object identification. In *Proc. of ACM SIGKDD Conf.*, 2002.
[34] V. Verykios, G.V.Moustakides, and M. Elfeky. A bayesian decision model for cost optimal record matching. *The VLDB Journal*, 12:28–40, 2003.
[35] S. White and P. Smyth. Algorithms for estimating relative importance in networks. In *Proc. of ACM SIGKDD Conf.*, 2003.
[36] W. Winkler. The state of record linkage and current research problems. In *U.S. Bureau of Census, TR99.*
[37] W. E. Winkler. Advanced methods for record linkage. In *U.S. Bureau of Census*, 1994.

Appendix

A A simple algorithm for learning irrelevant relationship types

There is a simple approach that can address the first drawback of the RelDC approach identified in Section 1 – the need for having an domain specialist to remove irrelevant relationship types from the entity-relationship graph. Let us assume that training datasets are available in which for each unresolved reference r_{ik} not only its choice set S_{ik} is known but also which y_j from S_{ik} is r^*_{ik}. In this case to determine whether a given relationship type is irrelevant for a particular disambiguation task we can simply compute the accuracy A_{with} of the approach when relationship of this type are present in the graph and then compute the accuracy of it $A_{without}$ when they are absent. If (consistently) $A_{with} > A_{without}$ then this relationship type is relevant, otherwise it is irrelevant and all the relationships of this relationship type can be removed from the entity-relationship graph.