Domain-Independent Data Cleaning via Analysis of Entity-Relationship Graph

DMITRI V. KALASHNIKOV and SHARAD MEHROTRA
University of California, Irvine

In this article, 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 the approach we propose (called RelDC) and the traditional techniques is that RelDC analyzes not only object features but also inter-object relationships to improve the disambiguation quality. Our extensive experiments over two real data sets and over synthetic datasets show that analysis of relationships significantly improves quality of the result.

Categories and Subject Descriptors: H.2.m [**Database Management**]: Miscellaneous—*Data cleaning*; H.2.8 [**Database Management**]: Database Applications—*Data mining*; H.2.5 [**Information Systems**]: Heterogeneous Databases; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

General Terms: Algorithms, Design, Experimentation, Performance, Theory

Additional Key Words and Phrases: Connection strength, data cleaning, entity resolution, graph analysis, reference disambiguation, relationship analysis, RelDC

1. INTRODUCTION

Recent surveys [KDSurvey 2003] 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 [Little and Rubin 1986], handling erroneous data [Maletic

This material is based upon work supported by the National Science Foundation under Award Numbers 0331707 and 0331690 and in part by the National Science Foundation under Award Numbers IIS-0083489 as part of the Knowledge Discovery and Dissemination program. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Corresponding Author's addresses: D. V. Kalashnikov, University of California, Irvine, 4300 Calit2 Building, Irvine, CA 92697-2815; email: dvk@ics.uci.edu.

and Marcus 2000], record linkage [Bilenko and Mooney 2003; Jin et al. 2003; Chaudhuri et al. 2003], and so on. In this article, we address one such challenge called *reference disambiguation*, which is also known as "fuzzy match" [Chaudhuri et al. 2003] and "fuzzy lookup" [Chaudhuri et al. 2005].

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* [Jin et al. 2003; Chaudhuri et al. 2003; Bilenko and Mooney 2003] 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"[1]) 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 two records are the same) can be used to determine if a particular reference corresponds to a given entity or not. This article 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 this article, we propose a domain-independent data cleaning approach for reference disambiguation, referred to as Relationship-based Data Cleaning (RELDC), which systematically exploits not only features but also relationships

[1]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.

$\langle A_1$, 'Dave White', 'Intel'$\rangle$
$\langle A_2$, 'Don White', 'CMU'$\rangle$
$\langle A_3$, 'Susan Grey', 'MIT'$\rangle$
$\langle A_4$, 'John Black', 'MIT'$\rangle$
$\langle A_5$, 'Joe Brown', unknown$\rangle$
$\langle A_6$, 'Liz Pink', unknown$\rangle$

Fig. 1. *Author* records.

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 primary contributions of this article are:

(1) developing a systematic approach to exploiting both attributes as well as relationships among entities for reference disambiguation,
(2) developing a set of optimizations to achieve an efficient and scalable (to large graphs) implementation of the approach,
(3) establishing that exploiting relationships can significantly improve the quality of reference disambiguation by testing the developed approach over 2 real-world data sets as well as synthetic data sets.

A preliminary version of this article appeared in Kalashnikov et al. [2005]; it presents an overview of the approach, without implementation and other details, required for implementing the approach in practice. The rest of this article is organized as follows: Section 2 presents a motivational example. In Section 3, we precisely formulate the problem of reference disambiguation and introduce notation that will help explain the RELDC approach. Section 4 describes the RELDC approach. The empirical results of RELDC are presented in Section 6. Section 7 contains the related work, and Section 8 concludes the article.

2. 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.

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 1 and 2.

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

⟨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'**⟩

Fig. 2. *Publication* records.

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 interconnected entities (modeled as nodes) linked to each other via relationships (modeled as edges). Figure 3 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

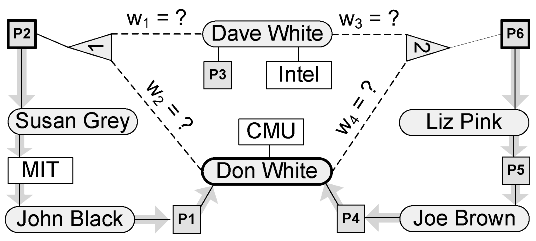

Fig. 3. Graph for the publications example.

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, where $\ell = 1, 2, \ldots, N$ and $\ell \neq j$.*

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: $P_2 \leftrightarrow$ 'Susan Grey' $\leftrightarrow$ 'MIT' $\leftrightarrow$ 'John Black' $\leftrightarrow P_1 \leftrightarrow$ '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: $P_6 \leftrightarrow$ 'Liz Pink' $\leftrightarrow P_5 \leftrightarrow$ 'Joe Brown' $\leftrightarrow P_4 \leftrightarrow$ '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). Therefore, 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. The above discussion naturally leads to two questions:

(1) Does the context attraction principle hold over real data sets. That is, if we disambiguate references based on the principle, will the references be correctly disambiguated?
(2) Can we design a generic solution to exploiting relationships for disambiguation?

Table I. Notation

Notation	Meaning
$\mathcal{D}$	the database
$X = \{x\}$	the set of all entities in $\mathcal{D}$
$R = \{r\}$	the set of all unresolved references
r	a reference
r^*	the to-be-found entity that r refers to
x_r	the context entity of r
S_r	the option set of r
$N = \|S_r\|$	convenience notation for $\|S_r\|$
$y_{r1}, y_{r2}, \ldots, y_{rN}$	all elements of S_r
$G = (V, E)$	the entity-relationship graph for $\mathcal{D}$
e_{rj}	the edge (r, y_{rj})
w_{rj}	the weight of edge e_{rj}
L	the path length limit parameter
$\mathcal{P}_L(u, v)$	the set of all L-short simple u-v paths in G
$c(u, v)$	the connection strength between nodes u and v in G
$\mathcal{N}_k(v)$	the neighborhood of node v of radius k in graph G

Of course, the second question is only important if the answer to the first is positive. However, we cannot really answer the first unless we develop a general strategy to exploiting relationships for disambiguation and testing it over real data. We will develop one such general, domain-independent strategy for exploiting relationships for disambiguation, which we refer to as RelDC in Section 4. We perform extensive testing of RelDC over both real data from two different domains as well as synthetic data to establish that exploiting relationships (as is done by RelDC) significantly improves the data quality. Before we develop RelDC, we first develop notation and concepts needed to explain our approach in Section 3.

3. PROBLEM DEFINITION

In this section, we first develop notation and then formally define the problem of reference disambiguation. The notation is summarized in Table I.

3.1 References

Let $\mathcal{D}$ be the database that contains references that are to be resolved. Let $X = \{x\}$ be the set of all entities[2] in $\mathcal{D}$. Each entity x consists of a set of m properties $\{a_1, a_2, \ldots, a_m\}$ and of a set of n **references** $\{r_1, r_2, \ldots, r_n\}$. The number of attributes m and the number of references n in those two sets can be different for different entities. Each reference r is essentially a description of some entity and may itself consist of one or more attributes. For instance, in the example in Section 2, *paper* entities contain one-attribute `authorRef` references in the form $\langle$*author name*$\rangle$. If, besides author names, author affiliation were also stored in the *paper records*, then `authorRef` references would have consisted of two attributes $\langle$*author name*, *author affiliation*$\rangle$. Each reference r

[2]Here entities have essentially the same meaning as in the standard E/R model.

maps uniquely into the entity, in the context of which it is made. That entity is called the *context entity* of reference r and denoted x_r. The set of all references in the database $\mathcal{D}$ will be denoted as R.

3.1.1 *Option Set.* Each reference $r \in R$ semantically refers to a single specific entity in X called the *answer* for reference r and denoted r^*. The description provided by r may, however, match a set of one or more entities in X. We refer to this set as the *option set* of reference r and denote it by S_r. The option set consists of all the entities that r could potentially refer to. Set S_r consists of $|S_r|$ elements $y_{r1}, y_{r2}, \dots, y_{r|S_r|}$ called *options* of reference r:

$$S_r = \{y_{r1}, y_{r2}, \dots, y_{r|S_r|}\}.$$

We assume S_r is given for each r. If it is not given, we assume a feature-based similarity approach is used to construct S_r by choosing all of the candidates such that FBS similarity between them and r exceed a given threshold. To simplify notation, we will use N to mean $|S_r|$, that is $N = |S_r|$.

Example 3.1.1 (*Notation*). Let us consider an example of using notation. Consider the reference $r =$ 'John Black' in publication P_1 illustrated in Figure 2. Then $x_r = P_1$, $r^* = A_4$, $S_r = \{A_4\}$, and $y_{r1} = A_4$. If we consider the reference $s =$ 'D. White' in publication P_2 and assume that in reality it refers to A_2 (i.e., 'Don White'), then $x_s = P_2$, $s^* = A_2$, $S_s = \{A_1, A_2\}$, $y_{s1} = A_1$, and $y_{s2} = A_2$.

3.2 The Entity-Relationship Graph

RelDC views the resulting database $\mathcal{D}$ as an undirected entity-relationship graph[3] $G = (V, E)$, where V is the set of nodes and E is the set of edges. The set of nodes V is comprised of the set of **regular** nodes V_{reg} and the set of choice nodes V_{cho}, that is $V = V_{reg} \cup V_{cho}$. Each regular node corresponds to some entity $x \in X$. We will use the same notation x for both the entity and the node that represents x. Choice nodes will be defined shortly. Each edge in the graph corresponds to a relationship.[4] Let us note that if entity x_1 contains a reference to entity x_2, then nodes x_1 and x_2 are linked via an edge, since the reference establishes a relationship between the two entities. For instance, an `authorRef` reference from a paper to an author corresponds to the *writes* relationship between the author and the paper.

In the graph, the edges have weights and the nodes do not have weights. Each edge weight is a real number between zero and one. It reflects the degree of confidence the relationship that corresponds to the edge exists. For instance, in the context of our author matching example, if we are absolutely confident a given author is affiliated with a given organization, then we assign the weight

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

[4]We concentrate primarily on binary relationships. Multi-way relationships are rare and most of them can be converted to binary relationships [Garcia-Molina et al. 2002]. Most of the design models/tools only deal with binary relationships, for instance ODL (Object Definition Language) supports only binary relationships.

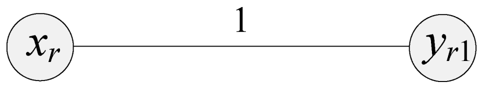

Fig. 4. Direct edge.

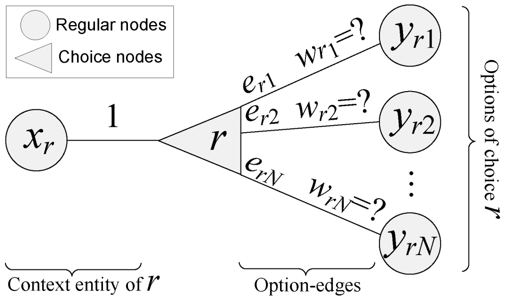

Fig. 5. Choice node.

of 1 to the corresponding edge. By default, all edge weights are equal to 1.[5] We will use the notation *edge label* and *edge weight* interchangeably.

3.2.1 *References and Linking.* If S_r has only one element y_{r1}, then r is resolved to y_{r1}, and graph G contains an edge between x_r and y_{r1} as shown in Figure 4. This edge is assigned weight 1 to denote that we are confident that r^* is y_{r1}.

If S_r has more than 1 element, then a *choice node* is created for reference r, as illustrated in Figure 5, to reflect the fact that r^* can be one of $y_{r1}, y_{r2}, \dots, y_{rN}$. Given the direct correspondence between a reference r and its choice node, we will use the same notation r for both of them. Node r is linked with node x_r via edge (x_r, r). Node r is also linked with N nodes $y_{r1}, y_{r2}, \dots, y_{rN}$, for each y_{rj} in S_r, via edges $e_{rj} = (r, y_{rj})$ for $j = 1, 2, \dots, N$. Edges $e_{r1}, e_{r2}, \dots, e_{rN}$ are called the *option-edges* of choice r. The weights of option-edges are called option-edge weights or simply *option weights*. The weight of edge (x_r, r) is 1. Each weight w_{rj} of edge e_{rj} for $j = 1, 2, \dots, N$ is undefined initially. Since option-edges $e_{r1}, e_{r2}, \dots, e_{rN}$ represent mutually exclusive alternatives, the sum of their weights should be 1: $w_{r1} + w_{r2} + \cdots + w_{rN} = 1$. Option-weights are different from other edge weights: they are *variables* the values of which are to be determined by the disambiguation algorithm, whereas other edge weights are constants.

[5]To illustrate when the edge weights can be different from 1, consider the following scenario. Assume the dataset being processed is originally derived from raw data by some *extraction software*. For example, the author affiliations can be derived by crawling the web, retrieving various web pages perceived to be faculty homepages, extracting what appears to be faculty names and their affiliations and putting them in the dataset. The extraction software is not always 100% confident in all the associations among entities it extracts, but might be able to assign the degree of its belief instead.

3.3 The Objective of Reference Disambiguation

To *resolve* reference r means to choose one entity y_{rj} from S_r in order to determine r^*. If entity y_{rj} is chosen as the outcome of such a disambiguation, then r is said to be *resolved to* y_{rj} or simply *resolved*. Reference r is said to be *resolved correctly* if this y_{rj} is r^*. Notice, if S_r has just one element y_{r1} (i.e., $N = 1$), then reference r is automatically resolved to y_{r1}. Thus, reference r is said to be *unresolved* or *uncertain* if it is not resolved yet to any y_{rj} and $N > 1$.

From the graph theoretic perspective, to resolve r means to assign weight 1 to one edge e_{rj}, where $1 \leq j \leq N$, and assign weight 0 to the other $N - 1$ edges $e_{r\ell}$, where $\ell = 1, 2, \ldots, N; \ell \neq j$. This will indicate the algorithm chooses y_{rj} as r^*.

The *goal* of reference disambiguation is to resolve all references as correctly as possible, that is, for each reference $r \in R$ to correctly identify r^*. The *accuracy* of reference disambiguation is the fraction of references being resolved that are resolved correctly.

The *alternative goal* is for each $y_{rj} \in S_r$ to associate weight w_{rj} that reflects the degree of confidence that y_{rj} is r^*. For this alternative goal, each edge e_{rj} should be labeled with such a weight. Those weights can be *interpreted* later to achieve the main goal: for each r try to identify only one y_{rj} as r^* correctly. We emphasize this alternative goal since most of our discussion will be devoted to a method for computing those weights. An interpretation of those weights (in order to try to identify r^*) is a small final step of RelDC. Namely, it achieves this by picking y_{rj} such that w_{rj} is the largest among $w_{r1}, w_{r2}, \ldots, w_{rN}$. That is, the algorithm resolves r to y_{rj} where $j : w_{rj} = \max_{\ell=1,2,\ldots,N} w_{r\ell}$.

3.4 Connection Strength and CAP Principle

RelDC resolves references by employing the context attraction principle presented in Section 2. We now state the principle more formally. Crucial to the principle is the notion of the *connection strength* $c(x_1, x_2)$ between two entities x_1 and x_2, which captures how strongly x_1 and x_2 are connected to each other through relationships. Many different approaches can be used to measure $c(x_1, x_2)$, they will be discussed in Section 4. Given the concept of $c(x_1, x_2)$, we can restate the context attraction principle as follows:

> **Context Attraction Principle:** Let r be a reference and $y_{r1}, y_{r2}, \ldots, y_{rN}$ be elements of its option set S_r with the corresponding option weights $w_{r1}, w_{r2}, \ldots, w_{rN}$. Then, for all $j, \ell \in [1, N]$, if $c_{r\ell} \geq c_{rj}$, then it is likely that $w_{r\ell} \geq w_{rj}$, where $c_{r\ell} = c(x_r, y_{r\ell})$ and $c_{rj} = c(x_r, y_{rj})$.

Let us remark that, as will be elaborated in Section 4.1.2, not all of the connection strength models are symmetric, that is, $c(u, v) \neq c(v, u)$ for some of them. In the above definition of the CAP, when resolving a reference r, the connection strength is measured in the direction of from x_r to y_{rj}, for any j.

4. RelDC APPROACH

We now have developed all the concepts and notation needed to explain the RelDC approach for reference disambiguation. The input to RelDC is the

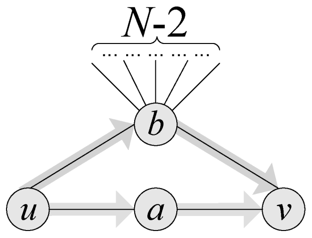

Fig. 6. Motivating $c(p)$ formula.

entity-relationship graph G discussed in Section 3. We assume that feature-based similarity approaches are 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 disambiguates references using the following four steps:

(1) *Compute Connection Strengths*. For each reference $r \in R$, compute the connection strength $c(x_r, y_{rj})$ for each $y_{rj} \in S_r$. The result is a set of equations that relate $c(x_r, y_{rj})$ with the option weights $\mathbf{w}$: $c(x_r, y_{rj}) = g_{rj}(\mathbf{w})$. Here, $\mathbf{w}$ is the set of all option weights in the graph G: $\mathbf{w} = \{w_{rj} : \text{ for all } r, j\}$.
(2) *Determine Equations for Option Weights*. Using the equations from Step (1) and the CAP, determine a set of equations that relate option weights to each other.
(3) *Compute Weights*. Solve the set of equations from Step (2).
(4) *Resolve References*. Interpret the weights computed in Step (3), as well as attribute-based similarity, to resolve references.

We now discuss the above steps in more detail in the following sections.

4.1 Computing Connection Strength

The concept of *connection strength* is at the core of the proposed data cleaning approach. The connection strength $c(u, v)$ between the two nodes u and v should reflect how strongly these nodes are related to each other via relationships in the graph G. The value of $c(u, v)$ should be computed according to some connection strength *model*. Below, we first motivate one way for computing $c(u, v)$ (Section 4.1.1), followed by a discussion of the existing connection strength (c.s.) models (Section 4.1.2). We will conclude our discussion on c.s. models with one specific model, called the weight-based model, which is the primary model we use in our empirical evaluation of RelDC (Section 4.1.3).

4.1.1 *Motivating a Way to Compute $c(u, v)$.* Many existing measures, such as the length of the shortest path or the value of the maximum network flow between nodes u and v, could potentially be used for this purpose. Such measures, however, have some drawbacks in our context.

For instance, consider the example in Figure 6 which illustrates a subgraph of G. The subgraph contains two paths between nodes u and v: $p_a = u \leftrightarrow a \leftrightarrow v$ and $p_b = u \leftrightarrow b \leftrightarrow v$. In this example, node b connects multiple nodes, not just

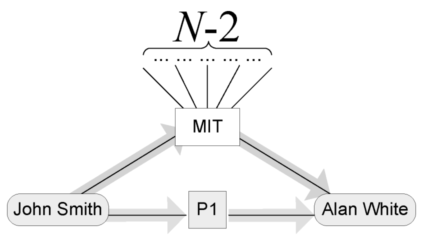

Fig. 7. Example for author matching.

u and v, whereas node a connects only u and v. If this subgraph is a part of an author-publication graph, for example, as illustrated in Figure 7, then nodes u and v may correspond to two authors, node a to a specific publication, and node b to a university which connects numerous authors. Intuitively, we expect that the connection strength between u and v via a is stronger than the connection strength between u and v via b: Since b connects many nodes, it is not surprising we can also connect u and v via b, whereas the connection via a is unique to u and v. Let $c(p_a)$ and $c(p_b)$ denote the connection strength via paths p_a and p_b. Then, based on our intuition, we should require that $c(p_a) > c(p_b)$.

Let us observe that measures such as *path length* and *maximum network flow* do not capture the fact that $c(p_a) > c(p_b)$. That is, the path length of p_a and p_b is 2. The maximum network flow via p_a is 1 and via p_b is 1 as well [Cormen et al. 2001], provided the weight of all edges in Figure 6 is 1. Next, we cover several existing connection strength models, most of which will return for the case in Figure 6 that $c(p_a) > c(p_b)$, as we desire. To measure $c(u, v)$, several of those models try to send a 'flow' in the graph G from the node u and then analyze which fraction of this flow reaches v. This fraction then determines the value of $c(u, v)$.

4.1.2 *Existing Connection Strength Models.* Recently, there has been a spike of interest by various research communities in the measures directly related to the $c(u, v)$ measure. Below we summarize several principal models. The reader who is primarily interested in the actual model employed by RelDC, might skip this section and proceed directly to Section 4.1.3 covering the weight-based model.

To measure $c(u, v)$ between nodes u and v, a connection strength model can take many factors into account: the paths in G between u and v, how they overlap, how they connect to the rest of G, the degree of nodes on those paths, the types of relationships that constitute those paths and so on. The existing models, and the two models proposed in this article, analyze only some of those factors. Nevertheless, we will show that even those models can be employed quite effectively by RelDC.

4.1.2.1 *Diffusion Kernels.* The area of kernel-based pattern analysis [Shawe-Taylor and Cristianni 2004] studies 'diffusion kernels on graph nodes', which are closely related to c.s. models. They are defined as follows.

A *base similarity graph* $G = (S, E)$ for a dataset S is considered. The vertices in the graph are the data items in S. The undirected edges in this graph are labeled with a 'base' similarity $\tau(\mathbf{x}, \mathbf{y})$ measure. That measure is also denoted as $\tau_1(\mathbf{x}, \mathbf{y})$, because only the direct links (of size 1) between nodes are utilized to derive this similarity. The base similarity matrix $\mathbf{B} = \mathbf{B}_1$ is then defined as the matrix whose elements $\mathbf{B}_{\mathbf{xy}}$, indexed by data items, are computed as $\mathbf{B}_{\mathbf{xy}} = \tau(\mathbf{x}, \mathbf{y}) = \tau_1(\mathbf{x}, \mathbf{y})$. Next, the concept of base similarity is naturally extended to path of arbitrary length k. To define $\tau_k(\mathbf{x}, \mathbf{y})$, the set of all paths $P^k_{\mathbf{xy}}$ of length k between the data items $\mathbf{x}$ and $\mathbf{y}$ is considered. The similarity is defined as the sum over all these paths of the products of the base similarities of their edges:

$$\tau_k(\mathbf{x}, \mathbf{y}) = \sum_{(\mathbf{x}_1\mathbf{x}_2\ldots\mathbf{x}_k)\in P^k_{\mathbf{xy}}} \prod_{i=1}^{k} \tau_1(\mathbf{x}_{i-1}, \mathbf{x}_i)$$

Given such $\tau_k(\mathbf{x}, \mathbf{y})$ measure, the corresponding similarity matrix $\mathbf{B}_k$ is defined. It can be shown that $\mathbf{B}_k = \mathbf{B}^k$. The idea behind this process is to enhance the base similarity by those indirect similarities. For example, the base similarity $\mathbf{B}_1$ can be enhanced with similarity $\mathbf{B}_2$, for example, by considering a combination of the two matrices: $\mathbf{B}_1 + \mathbf{B}_2$. The idea generalizes to more then two matrices. For instance, by observing that in practice the relevance of longer paths should decay, a decay factor λ is used, resulting in the so-called *exponential diffusion kernel*: $\mathbf{K} = \sum_{k=0}^{\infty} \frac{1}{k!}\lambda^k \mathbf{B}^k = \exp(\lambda\mathbf{B})$. The *von Neumann diffusion kernel* is defined similarly: $\mathbf{K} = \sum_{k=0}^{\infty} \lambda^k \mathbf{B}^k = (\mathbf{I} - \lambda\mathbf{B})^{-1}$. The elements of the matrix $\mathbf{K}$ exactly define what we refer to as the connection strength: $c(\mathbf{x}, \mathbf{y}) = \mathbf{K}_{\mathbf{xy}}$. While connection strength between nodes in a graph can be computed using diffusion kernels, it is of limited utility in our setting since the procedure in Shawe-Taylor and Cristianni [2004] cannot be directly applied in graphs with choice nodes.

4.1.2.2 *Relevant Importance in Graphs.* A natural way to compute the connection strength $c(u, v)$ between node u and v is to compute it as the probability of reaching node v from node u via random walks in graph G. Each step of the random walk is done according to certain probability derived from edge labels. Such problems have been studied for graphs in the previous work under Markovian assumptions. For example, White and Smyth [2003] study the related problem of computing the *relative importance* of a given set of nodes with respect to the set of "root" nodes by generalizing the PageRank algorithm [Brin and Page 1998]. Their primary approach views such a graph as a Markov chain where nodes represent states of the Markov chain and probabilities are determined by edge labels. White and Smyth [2003] also evaluate different models and compare them with the PageRank-based model.

The problem of computing $c(u, v)$ can be postulated as computing the relevant importance of node u with respect to the root node v. The procedural part of the PageRank-based algorithm in White and Smyth [2003], however, cannot be employed directly in our approach. The main reason is that the Markovian assumptions do not hold for our graphs. For example, consider two paths $G \leftrightarrow F$ and $D \leftrightarrow F$ in Figure 8. In that figure, F is a choice node and BF and FD are its mutually exclusive option-edges. In general, we can continue $G \leftrightarrow F$ path by following $F \leftrightarrow B$ link; however, we cannot continue $D \leftrightarrow F$ path by following

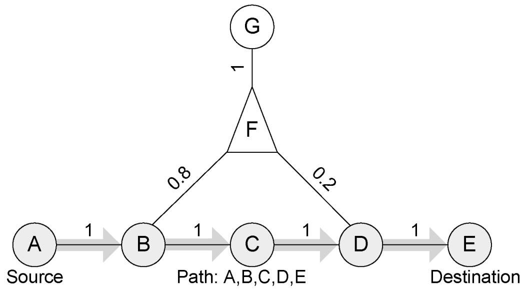

Fig. 8. Sample graph.

the same $F \leftrightarrow B$ link. Thus, the decision of whether we can follow $F \leftrightarrow B$ link is determined by the past links on the path. This violates the Markovian assumption, since a Markov chain is a random process, which has the property that, conditional on its present value, the future is independent of the past.

4.1.2.3 *Electric Circuit Analogy.* Faloutsos et al. [2004] consider a model for computing $c(u, v)$. They view the graph as an electric circuit consisting of resistors, and compute $c(u, v)$ as the amount of electric current that goes from u to v. One of the primary contributions of that article is the optimizations that scale their approach to large graphs.

4.1.3 *Weight-Based Model.* This section covers one of the c.s. models, which we will use to empirically evaluate RelDC in Section 6. We refer to this model as the weigh-based model (WM). WM is a simplification of the probabilistic model (PM), covered in the *electronic appendix*. PM model is more involved than WM model and will be a significant diversion from our main objective of explaining RelDC. PM computes $c(u, v)$ as the probability of reaching the node u starting from the node v in the graph G. Like the model proposed in White and Smyth [2003], WM and PM are also random walk-based models.

WM computes $c(u, v)$ as the sum of the connection strengths of each simple path between nodes u and v. The connection strength $c(p)$ of each path p from u to v is computed as the probability of following path p in graph G. In WM, computation of $c(u, v)$ consists of two phases. The first phase discovers connections (paths) between u and v. The second phase measures the strength in the connections discovered by the first phase.

Phase I: Path Discovering. In general, there can be many connections between nodes u and v 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 $\mathcal{P}_L(u, v)$ between nodes u and v 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.

Complications due to choice nodes. Not all of the discovered paths are considered when computing $c(x_r, y_{rj})$ to resolve reference r: some of them are *illegal paths*, which are ignored by the algorithm. Let $e_{r1}, e_{r2}, \ldots, e_{rN}$ be the option-edges associated with the reference r. When resolving r, RelDC tries do

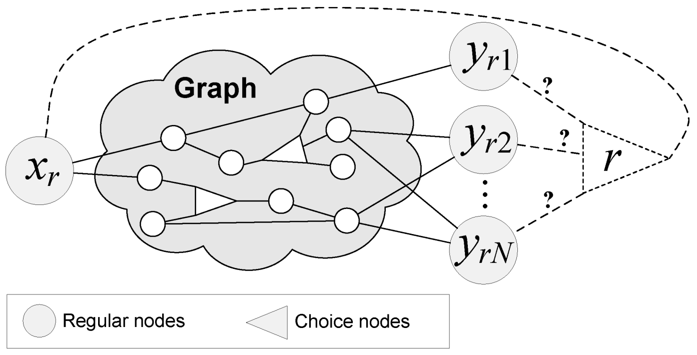

Fig. 9. Graph.

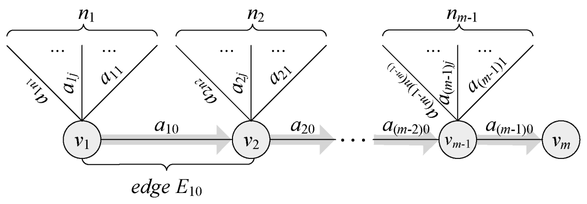

Fig. 10. Computing $c(p)$ of path $p = v_1 \leftrightarrow \cdots \leftrightarrow v_m$. Only possible-to-follow edges are shown.

determine weights of these edges via connections that exist in the remainder of the graph not including those edges. To achieve this, RELDC actually discovers paths not in graph G, but in $G_r = G - r$, as illustrated in Figure 9. That is, G_r is graph G with node r removed. Also, in general, paths considered when computing $c(x_r, y_{rj})$ may contain option-edges of some choice nodes. If a path p contains an option-edge e_{sj} of some choice node s, it should not contain another option-edge $e_{s\ell}$, where $\ell \neq j$, of the same choice node s, because edges e_{sj} and $e_{s\ell}$ are mutually exclusive.

Phase II: Measuring Connection Strength. In general, each L-short simple path p can be viewed as a sequence of m nodes $v_1 \leftrightarrow v_2 \leftrightarrow \cdots \leftrightarrow v_m$, where $m \leq L+1$, as illustrated in Figure 10. This figure shows that from a node v_i, where $i = 1, 2, \ldots, m-1$, it is possible-to-follow[6] $n_i + 1$ edges, labeled $a_{i0}, a_{i1}, \ldots, a_{in_i}$. WM computes the connection strength of path p as the probability Pr of following path p: $c(p) = Pr$. Probability Pr is computed as the product of two probabilities: $Pr = Pr_1 Pr_2$, where Pr_1 is the probability that path p exists and Pr_2 is the probability of following path p given that p exists.

Probability that Path Exists. First, path p should exist and thus each edge on this path should exist. WM computes the probability Pr_1 that p exist as the product of probabilities that each edge on path p exists: $Pr_1 = a_{10}a_{20} \times \cdots \times a_{(m-1)0}$. That is, WM assumes that each edge E_{i0} exists independently from edge $E_{\ell 0}$ where $\ell \neq i$.

New Labels, Given Path Exists. If we assume that p exists, then situation will look like that illustrated in Figure 11. In that figure, all edges are

[6]It is not possible to follow the edges, following which would make the path not simple.

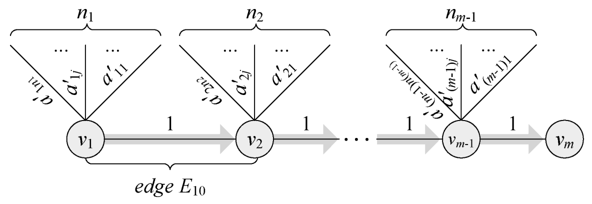

Fig. 11. Computing $c(p)$: new labels under assumption that p exists.

labeled with new weights a'_{ij}, derived from the old weights a_{ij} under the assumption that path p exists. For example, $a'_{i0} = 1$ for all i, because each edge E_{i0} exists if path p exists. For each a'_{ij} where $j \neq 0$ it holds that either $a'_{ij} = a_{ij}$, or $a'_{ij} = 0$. To understand why a'_{ij} can be zero, consider path p_1 = 'Don White' $\leftrightarrow P_4 \leftrightarrow$ Joe $\leftrightarrow P_5 \leftrightarrow$ Liz $\leftrightarrow P_6 \leftrightarrow$ '2' $\leftrightarrow$ 'Dave White' in Figure 3 as an example. If we assume p_1 exists, then edge ('2', 'Dave White') must exist and consequently edge ('2', 'Don White') does not exist. Thus, if path p_1 exists, the weight of edge ('2', 'Don White') is zero. That is why in general either $a'_{ij} = a_{ij}$, or, if the corresponding edge E_{ij} cannot exist under assumption that path p exists, then $a'_{ij} = 0$.

General Walk Model. Before we present the way WM computes probability Pr_2 of following path p given that p exists, let us discuss the general *walk model* utilized by WM when computing $c(u, v)$. Each path starts from u. Assume WM at some intermediate stage of a walk observes a path $p_x = u \rightsquigarrow x$. If $x = v$, then p_x is a legitimate u-v path and the walk stops. If $x \neq v$ and the length of the path p is L, then the walk stops since L is the path length limit. Otherwise, WM examines the incident edges of x, specifically those that 'can be followed', such that if an edge is followed the new path will remain simple (will not have duplicate edges). If there are no such edges, than this walk reached a 'dead-end' and the walk stops. Otherwise, WM chooses one edge to follow and continues the path p_x by following this edge to some new node y, such that $p_y = u \rightsquigarrow x \rightarrow y$. Then, the above procedure is repeated for p_y.

Probability of Following Path. Next, WM computes probability Pr_2 of following path p given that p exists as the product of probabilities of following each edge on p. In WM, the probability of following an edge is proportional to the weight of the edge. For example, the probability of following edge E_{10} in Figure 11 is: $\frac{1}{1+a'_{11}+a'_{12}+\cdots+a'_{1n_1}}$.

Total Formula. The connection strength of path p is computed as $c(p) = Pr_1 Pr_2$. The final formula for $c(p)$ is:

$$c(p) = \prod_{i=1}^{m-1} \frac{a'_{i0}}{1 + a'_{i1} + a'_{i2} + \cdots + a'_{in_i}}. \tag{1}$$

The total connection strength between nodes u and v is computed as the sum of connection strengths of paths in $\mathcal{P}_L(u, v)$:

$$c(u, v) = \sum_{p \in \mathcal{P}_L(u,v)} c(p). \tag{2}$$

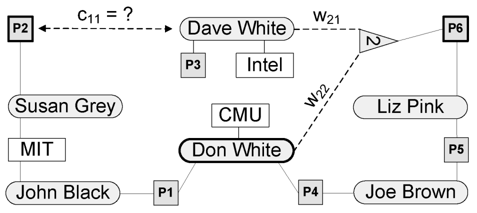

Fig. 12. Computing $c_{11} = c(P_2, \text{Dave})$ in the graph $G_r = G - $ '1'.

Measure $c(u, v)$ is the probability of reaching v from u by following only L-short simple paths, such that the probability of following an edge is proportional to the weight of the edge.

Example 4.1.1. Let us demonstrate WM formulae on the example in Figure 6. WM measures $c(p_a)$ and $c(p_b)$ as the probabilities of following paths p_a and p_b, respectively. WM computes those probabilities as follows. For path p_a, we start from u, we go to a with probability $\frac{1}{2}$ at which point we have no choice but to go to v, so the probability of following p_a is $\frac{1}{2}$. For path p_b, we start from u. Next, we have a choice to go to a or b with probabilities of $\frac{1}{2}$, and we choose to follow (u, b) edge. From node b, we can go to any of the $N - 1$ nodes (cannot go back to u) but we go specifically to v. Therefore, the probability of reaching v via path p_b is $\frac{1}{2(N-1)}$. Let us observe that $\frac{1}{2} > \frac{1}{2(N-1)}$ when $N \geq 2$ and thus WM captures that $c(p_a) > c(p_b)$.

4.1.3.1 *Connection Strengths in Toy Database.* Let us compute connection strengths c_{11}, c_{12}, c_{21}, c_{22} for the toy database illustrated in Figure 3. Those connection strength are defined as:

$$\begin{aligned} c_{11} &= c(P_2, \text{'Dave White'}) \\ c_{12} &= c(P_2, \text{'Don White'}) \\ c_{21} &= c(P_6, \text{'Dave White'}) \\ c_{22} &= c(P_6, \text{'Don White'}). \end{aligned} \tag{3}$$

Later, those connection strengths will be used to compute option weights w_{11}, w_{12}, w_{21}, w_{22}.

Consider first computing $c_{11} = c(P_2, \text{'Dave White'})$ in the context of disambiguating 'D. White' reference in P_2. Recall, for that reference choice node '1' has been created. The first step is to remove choice '1' from consideration. The resulting graph $G_r = G - $ '1' is shown in Figure 12. The next step is to discover all L-short simple paths in graph G_r between P_2 and 'Dave White'. Let us set $L = \infty$, then there is only one such path: $p_1 = P_2 \leftrightarrow \text{Susan} \leftrightarrow \text{MIT} \leftrightarrow \text{John} \leftrightarrow P_1 \leftrightarrow \text{Don} \leftrightarrow P_4 \leftrightarrow \text{Joe} \leftrightarrow P_5 \leftrightarrow \text{Liz} \leftrightarrow P_6 \leftrightarrow \text{'2'} \leftrightarrow$ Dave White. The discovered connection is too long to be meaningful in practice, but we will consider it for pedagogical reasons. To compute $c(p_1)$ we first compute the probability Pr_1 that path p_1 exists. Path p_1 exists if and only if edge between '2' and 'Dave White' exists, so $Pr_1 = w_{21}$. Now we assume that p_1 exists and compute the probability Pr_2 of following p_1 given that p_1 exists on the graph shown in Figure 13. That probability is $Pr_2 = \frac{1}{2}$. Thus, $c(p_1) = Pr_1 Pr_2 = \frac{w_{21}}{2}$.

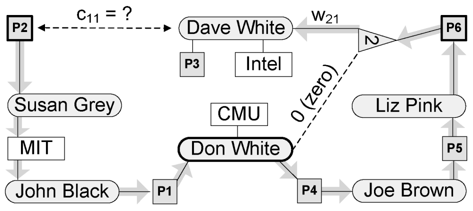

Fig. 13. Computing $c_{11} = c(P_2, \text{Dave})$ under the assumption that path $P_2 \rightsquigarrow$ 'Dave White' exists. Since edge '2' $\leftrightarrow$ 'Dave' exists, thus edge '2' $\leftrightarrow$ 'Don' does not exist.

The same result can be obtained by directly applying Equation (1). After computing c_{12}, c_{21}, and c_{22} in a similar fashion we have:

$$\begin{cases} c_{11} &= \frac{w_{21}}{2} \quad = c(p_1) \\ c_{12} &= 1 \quad\;\; = c(P_2 \leftrightarrow \text{Susan} \leftrightarrow \text{MIT} \leftrightarrow \text{John} \leftrightarrow P_1 \leftrightarrow \text{'Don White'}) \\ c_{21} &= \frac{w_{11}}{2} \\ c_{22} &= 1. \end{cases} \tag{4}$$

Notice, the toy database is small and MIT connects only two authors. In practice, MIT would connect many authors and thus connections via MIT will be weak.

4.2 Determining Equations for Option-Edge Weights

Given the connection strength measures $c(x_r, y_{rj})$ for each unresolved reference r and its options $y_{r1}, y_{r2}, \ldots, y_{rN}$, we can use the context attraction principle to determine the relationships between the weights associated with the option-edges in the graph G. Note that the context attraction principle does not contain any specific strategy on how to relate weights to connection strengths. Any strategy that assigns weight such that, if $c_{r\ell} \geq c_{rj}$, then $w_{r\ell} \geq w_{rj}$ is appropriate, where $c_{r\ell} = c(x_r, y_{r\ell})$ and $c_{rj} = c(x_r, y_{rj})$. In particular, we use the strategy where weights $w_{r1}, w_{r2}, \ldots, w_{rN}$ are proportional to the corresponding connection strengths: $w_{rj} c_{r\ell} = w_{r\ell} c_{rj}$. Using this strategy and given that $w_{r1} + w_{r2} + \cdots + w_{rN} = 1$, the weight w_{rj}, for $j = 1, 2, \ldots, N$, is computed as:

$$w_{rj} = \begin{cases} \frac{c_{rj}}{c_{r1}+c_{r2}+\cdots+c_{rN}} & \text{if } (c_{r1} + c_{r2} + \cdots + c_{rN}) > 0; \\ \frac{1}{N} & \text{if } (c_{r1} + c_{r2} + \cdots + c_{rN}) = 0. \end{cases} \tag{5}$$

For instance, for the toy database we have:

$$\begin{cases} w_{11} = c_{11}/(c_{11} + c_{12}) = \frac{w_{21}}{2} / \left(1 + \frac{w_{21}}{2}\right) \\ w_{12} = c_{12}/(c_{11} + c_{12}) = 1/\left(1 + \frac{w_{21}}{2}\right) \\ w_{21} = c_{21}/(c_{21} + c_{22}) = \frac{w_{11}}{2} / \left(1 + \frac{w_{11}}{2}\right) \\ w_{22} = c_{22}/(c_{21} + c_{22}) = 1/\left(1 + \frac{w_{11}}{2}\right). \end{cases} \tag{6}$$

4.3 Determining All Weights by Solving Equations

Given a system of equations, relating option-edge weights as derived in Section 4.2, our goal next is to determine values for the option-edge weights that satisfy the equations.

4.3.1 *Solving Equations for Toy Database.* Before we discuss how such equations can be solved in general, let us first solve Eqs. (6) for the toy example. Those equations, given an additional constraint $0 \leq w_{11}, w_{12}, w_{21}, w_{22} \leq 1$, have a unique solution $w_{11} = 0$, $w_{12} = 1$, $w_{21} = 0$, $w_{22} = 1$. Once we have computed the weights, RelDC will interpret these weights to resolve the references. In the toy example, the above computed weights will lead RelDC to resolve 'D. White' in both P_2 and P_6 to 'Don White', since $w_{12} > w_{11}$ and $w_{22} > w_{21}$.

4.3.2 *General Case.* In general case, Eqs. (1), (2), and (5), define each option weight as a function of other option weights $w_{rj} = f_{rj}(\mathbf{w})$:

$$\begin{cases} w_{rj} = f_{rj}(\mathbf{w}) \text{ (for all } r, j) \\ 0 \leq w_{rj} \leq 1 \quad \text{(for all } r, j). \end{cases} \tag{7}$$

The exact function for w_{rj} is determined by Eqs. (1), (2), and (5), and by the paths that exists between x_r and y_{rj} in G. In practice, often $f_{rj}(\mathbf{w})$ is constant leading to the equation of the form $w_{rj} = const$.

The goal is to solve System (7). System (7) might be over-constrained and thus might not have an exact solution. Thus, we add slack to it by transforming each equation $w_{rj} = f_{rj}(\mathbf{w})$ into $f_{rj}(\mathbf{w}) - \xi_{rj} \leq w_{rj} \leq f_{rj}(\mathbf{w}) + \xi_{rj}$. Here, ξ_{rj} is a slack variable that can take on any real nonnegative value. The problem transforms into solving the nonlinear programming problem (NLP) where the objective is to minimize the sum of all ξ_{rj}:

$$\begin{cases} \text{Constraints:} & \\ f_{rj}(\mathbf{w}) - \xi_{rj} \leq w_{rj} \leq f_{rj}(\mathbf{w}) + \xi_{rj} & \text{(for all } r, j) \\ 0 \leq w_{rj} \leq 1 & \text{(for all } r, j) \\ 0 \leq \xi_{rj} & \text{(for all } r, j) \\ & \\ \text{Objective: Minimize } \sum_{r,j} \xi_{rj} & \end{cases} \tag{8}$$

System (8) always has a solution. To show that, it is sufficient to prove that there is at least one solution that satisfies the constraints of System (8). Let us prove that by constructing such a solution. Notice, functions $f_{rj}(\mathbf{w})$ (for all r, j) are such that $0 \leq f_{rj}(\mathbf{w}) \leq 1$, if $0 \leq w_{s\ell} \leq 1$ (for all s, ℓ). Thus, the following combination: $w_{rj} = 0$ and $\xi_{rj} = 1$ (for all r, j) is a solution that satisfies the constraints of System (8), though it does not satisfy the objective in general. The goal, of course, is to find a better solution that minimizes $\sum_{r,j} \xi_{rj}$. The pseudocode for the above procedure will be discussed in Section 5.1.1.

4.3.3 *Iterative Solution.* The straightforward approach to solving the resulting NLP problem (8) is to use one of the off-the-shelf math solver such as

SNOPT. Such solvers, however, do not scale to large problem sizes that we encounter in data cleaning as will be discussed in Section 6. We therefore exploit a simple iterative approach, which is outlined below. Note, however, other methods can be devised to solve (8) as well, for example, in Kalashnikov and Mehrotra [2005], we sketch another approximate algorithm for solving (8), which first computes bounding intervals for all option weights w_{rj}'s and then employs techniques from Cheng et al. [2003a, 2003b, 2004]. That method is more involved than the iterative solution, which we will present next. The pseudocode for the iterative method is given in Figure 15 in Section 5.1.2.

The iterative method first iterates over each reference $r \in R$ and assigns initial weights of $\frac{1}{|S_r|}$ to each w_{rj}. It then starts its major iterations in which it first computes $c(x_r, y_{rj})$ for all r, j using Eq. (2). After $c(x_r, y_{rj})$'s are computed, they are used to compute w_{rj} for all r, j using Eq. (5). Note that the values of w_{rj} for all r, j will change from $\frac{1}{|S_r|}$ to new values. The algorithm performs several major iterations until the weights converge (the resulting changes across iterations are negligible) or the algorithm is explicitly stopped.

Let us perform an iteration of the iterative method for the example above. First

$$w_{11} = w_{12} = \frac{1}{2} \text{ and } w_{21} = w_{22} = \frac{1}{2}.$$

Next,

$$c_{11} = \frac{1}{4}, c_{12} = 1, c_{21} = \frac{1}{4}, c_{22} = 1.$$

Finally,

$$w_{11} = \frac{1}{5}, w_{12} = \frac{4}{5}, w_{21} = \frac{1}{5}, w_{22} = \frac{4}{5}.$$

If we stop the algorithm at this point and interpret w_{rj}'s, then the RelDC's answer will be identical to that of the exact solution: 'D. White' is 'Don White'.

Note that the above-described iterative procedure computes only an *approximate* solution for the system whereas the solver finds the exact solution. Let us refer to iterative implementation of RelDC as It-RelDC and denote the implementation that uses a solver as Sl-RelDC. For both It-RelDC and Sl-RelDC, after the weights are computed, those weights are *interpreted* to produce the final result, as discussed in Section 3.3. It turned out that the accuracy of It-RelDC (with a small number of iterations, such as 10–20) and of Sl-RelDC is practically identical. This is because even though the iterative method does not find the exact weights, the weights computed by the iterative algorithm are close enough to those computed using a solver. Thus, when the weights are *interpreted*, both methods obtain similar results.

4.4 Resolving References by Interpreting Weights

When resolving references r and deciding which entity among $y_{r1}, y_{r2}, \ldots, y_{rN}$ from S_r is r^*, RelDC chooses such y_{rj} that w_{rj} is the largest among $w_{r1}, w_{r2}, \ldots, w_{rN}$. Notice, to resolve r we could have also combined w_{rj} weights with feature-based similarities $FBS(x_r, y_{rj})$ (e.g., as a weighted sum), but we do

```
NAÏVE-SL-RELDC()
1  if GR-RELDC then
2      INITIALIZE($\mathbf{w}$)
3  for each $r \in R$
4      $G_r \leftarrow G - r$
5      for $j \leftarrow 1$ to $|S_r|$ do
6          $\mathcal{P}_L(x_r, y_{rj}) \leftarrow$ ALL-PATHS($G_r, x_r, y_{rj}, L$)
7          $EQ[c(x_r, y_{rj})] \leftarrow$ EQ-CON-STRENGTH($\mathbf{w}, \mathcal{P}_L(x_r, y_{rj})$)
8          delete $\mathcal{P}_L(x_r, y_{rj})$ //free storage
9  for each $r \in R$
10     for $j \leftarrow 1$ to $|S_r|$ do
11         $EQ[w_{rj}] \leftarrow$ EQ-ASSIGN-WEIGHT($j, c(x_r, y_{r1}), \ldots, c(x_r, y_{r|S_r|})$)
12 $S \leftarrow$ PREPARE-FOR-SOLVER(all $EQ[w_{rj}]$)
13 $\mathbf{w} \leftarrow$ SOLVE-USING-SOLVER($S$)
14 INTERPRET($\mathbf{w}$)

INITIALIZE($\mathbf{w}$)
1  for each $r \in R$
2      for $j \leftarrow 1$ to $|S_r|$ do
3          $w_{rj} \leftarrow \frac{1}{|S_r|}$
```

Fig. 14. NAÏVE-SL-RELDC.

not study that approach in this paper. Once the interpretation of the weights is done, the main disambiguation goal is achieved, and the outcome of the disambiguation can be used to create a regular database.

5. IMPLEMENTATIONS

In this section, we discuss several implementations of RELDC, which are crucial to scale the approach to large datasets.

5.1 Iterative and Solver Implementations of RELDC

The NLP problem in Eq. (8) can be solved iteratively or using a solver. In this section we present pseudocode for naïve implementations of SL-RELDC and IT-RELDC. In the subsequent sections, we discuss optimizations of these naïve implementations.

5.1.1 *Solver.* Figure 14 shows an outline of SL-RELDC, which we have discussed in Section 4. In lines 1–2, if the greedy implementation of ALL-PATHS is used (see Section 5.3), the algorithm initializes weights. Initial values of option weights $w_{r1}, w_{r2}, \ldots, w_{rN}$ of each choice node r are assigned such that $w_{r1} = w_{r2} = \cdots = w_{rN} = \frac{1}{N}$ and $w_{r1} + w_{r2} + \cdots + w_{rN} = 1$. Lines 3–9 correspond to creating equations for connection strengths $c(x_r, y_{rj})$ (for all r, j) described in Section 4.1: each $c(x_r, y_{rj})$ is derived based on the simple paths that exist between nodes for x_r and y_{rj} in the graph. Lines 10–13 correspond to the procedure from Section 4.2 that constructs the equations for option weighs w_{rj} (for all r, j). Then, in Line 14, the algorithm takes the NLP problem shown in Eq. (8) and creates its representation S suitable for the solver. Next, the solver takes the input S, solves the problem, and outputs the resulting weights. As the final steps, all the references are resolved by interpreting those weights.

5.1.2 *Iterative.* The pseudocode in Figure 15 formalizes the IT-RELDC procedure described in Section 4.3. IT-RELDC first initializes weights. Then, it

```
NAÏVE-IT-RELDC($N_{iter}$)
1  INITIALIZE($\mathbf{w}$)
2  MAIN-LOOP($\mathbf{w}$, $N_{iter}$)
3  INTERPRET($\mathbf{w}$)

MAIN-LOOP($\mathbf{w}$, $N_{iter}$)
1  for $\ell \leftarrow 1$ to $N_{iter}$ do
2    for each $r \in R$
3      $G_r \leftarrow G - r$
4      for $j \leftarrow 1$ to $|S_r|$ do
5        $\mathcal{P}_L(x_r, y_{rj}) \leftarrow$ ALL-PATHS($G_r, x_r, y_{rj}, L$)
6        $c(x_r, y_{rj}) \leftarrow$ CON-STRENGTH($\mathbf{w}, \mathcal{P}_L(x_r, y_{rj})$)
7        delete $\mathcal{P}_L(x_r, y_{rj})$ //free storage
8    for each $r \in R$
9      for $j \leftarrow 1$ to $|S_r|$ do
10       $w_{rj} \leftarrow$ ASSIGN-WEIGHT($j, c(x_r, y_{r1}), \ldots, c(x_r, y_{r|S_r|})$)
```

Fig. 15. NAÏVE-IT-RELDC.

iterates recomputing new values for $c(x_r, y_{rj})$ and w_{rj} (for all r, j). Finally, all the references are resolved by interpreting the weights.

5.1.3 *Bottleneck of* RELDC. To optimize RELDC for performance we need to understand where it spends most of its computation time. The most computationally expensive part of both IT-RELDC and SL-RELDC is ALL-PATHS procedure, which discovers connections between two nodes in the graph. For certain combinations of parameters, SOLVE-USING-SOLVER procedure, which invokes the solver to solve the NLP problem, can be expensive as well. However, that procedure is performed by a third party solver, hence there is little possibility of optimizing it. Therefore, all of the optimizations presented in this section target ALL-PATHS procedure.

5.2 Constraining the Problem

This section lists several optimizations that improve the efficiency of RELDC by constraining/simplifying the problem.

Limiting Paths Length. ALL-PATHS algorithm can be specified to look only for paths of length no greater than a parameter L. This optimization is based on the premise that longer paths tend to have smaller connection strengths while RELDC will need to spend more time to discover those paths.

Weight Cut-Off Threshold. This optimization can be applied after a few iterations of IT-RELDC. When resolving reference r, see Figure 5, IT-RELDC can use a threshold to prune several y_{rj}'s from S_r. If the current value of w_{rj} is too small compared to $w_{r\ell}$ for $\ell = 1, 2, \ldots, N; \ell \neq j$, then RELDC will assume y_{rj} cannot be r^* and will remove y_{rj} from S_r. The threshold is computed per each reference r as $\alpha \frac{1}{|S_r|}$, where α $(0 \leq \alpha < 1)$ is a real number (a fixed parameter).[7] This optimization improves the efficiency since if y_{rj} is removed from S_r, then IT-RELDC will not recompute $\mathcal{P}_L(x_r, y_{rj})$, $c(x_r, y_{rj})$, and w_{rj} any longer.

Restricting Path Types. The analyst can specify path types of interest (or for exclusion) explicitly.[8] For example, the analyst can specify that only paths of

[7]The typical choices for α in our experiments are 0.0 (i.e., the optimization is not used), 0.2 and 0.3.
[8]This optimization has not been used in our experiments.

```
DF-ALL-PATHS($G, u, v, L$)
1  $A \leftarrow \emptyset$     // the answer set
2  $Push(S, u)$
3  while $NotEmpty(S)$ do
4      $p \leftarrow Pop(S)$
5      if $LastNode(p) = v$ then
6          $A = A \cup \{p\}$
7      else if $Length(p) < L$ then
8          DF-EXPAND-PATH($p, S$)
9  return $A$

DF-EXPAND-PATH($p, S$)
1  $x \leftarrow LastNode(p)$
2  for each $z \in V : (x, z) \in E$ do
3      if ISLEGAL($p \leftrightarrow z$) then
4          $Push(S, p \leftrightarrow z)$
```

Fig. 16. DF-ALL-PATHS.

```
GR-ALL-PATHS($G, u, v, L$)
1  $A \leftarrow \emptyset$     // the answer set
2  $Insert(Q, u, 1)$
3  while $NotEmpty(Q)$ and
         STOPCONDITION(.) = false do
4      $p \leftarrow Get(Q)$
5      if $LastNode(p) = v$ then
6          $A = A \cup \{p\}$
7      else if $Length(p) < L$ then
8          GR-EXPAND-PATH($p, Q$)
9  return $A$

GR-EXPAND-PATH($p, Q$)
1  $x \leftarrow LastNode(p)$
2  for each $z \in V : (x, z) \in E$ do
3      if ISLEGAL($p \leftrightarrow z$) then
4          $Insert(Q, p \leftrightarrow z, c(p \leftrightarrow z))$
```

Fig. 17. GR-ALL-PATHS.

type $T_1 \leftrightarrow T_2 \leftrightarrow T_4 \leftrightarrow T_1$ are of interest, where T_i's are node types. Some of such rules are easy to specify, however it is clear that for a generic framework here should be some method (e.g., a language) for an analyst to specify rules that are more complicated. Our ongoing work addresses the problem of such a language [Seid and Mehrotra 2006].

5.3 Depth-First and Greedy Versions of ALL-PATHS

RELDC utilizes ALL-PATHS procedure to discover all L-short simple paths between two nodes. We have considered two approaches for implementing ALL-PATHS algorithm: the depth-first (DF-ALL-PATHS) and greedy (GR-ALL-PATHS) provided in Figures 16 and 17 respectively.[9]

The reason for having those two implementations is as follows. The DF-ALL-PATHS is a good choice if skipping of paths is *not allowed*: we shall show that in this case DF-ALL-PATHS is better in terms of time and space complexity than its greedy counterpart. However, GR-ALL-PATHS is a better option if one is

[9]All of the optimizations mentioned in this article can be applied to both of these approaches.

interested in fine-tuning the *accuracy vs. performance* trade-off by restricting the running time of the ALL-PATHS algorithm. The reason for this is as follows. If DF-ALL-PATHS is stopped abruptly in the middle of its execution, then certain important paths can still be not discovered. To address this drawback, GR-ALL-PATHS discovers the most important paths first and least important last.

5.3.1 *Depth-First and Greedy Algorithms.* As can be seen from Figures 16 and 17, the depth-first and greedy algorithms are quite similar. The difference between them is that DF-ALL-PATHS utilizes a stack to account for intermediate paths while GR-ALL-PATHS utilizes a priority queue. The key in this queue is the connection strengths of intermediate paths. Also, GR-ALL-PATHS stops if the stop conditions are met (Line 3 in Figure 17) even if not all paths have been examined yet, whereas DF-ALL-PATHS discovers all paths without skipping any paths.

Both algorithms look for $u \rightsquigarrow v$ paths and start with the intermediate path consisting of just the source node u (Line 2). They iterate until no intermediate paths are left under consideration (Line 3). The algorithms extract the next intermediate path p to consider (from the stack or queue) (Line 4). If p is a $u \rightsquigarrow v$ path, then p is added to the answer set A and algorithm proceeds to Line 3 (Lines 5–6). If p is not a $u \rightsquigarrow v$ path and the length of p is less than L, then the EXPAND-PATH procedure is called for path p (Lines 7–8). The EXPAND-PATH procedure first determines the last node x of the intermediate path $p = u \rightsquigarrow x$. It then analyzes each direct neighbor z of node x and if path $p \leftrightarrow z$ is a legal paths, then it inserts this path into the stack (or queue) for further consideration.

The STOPCONDITION() procedure in Line 3 of GR-ALL-PATHS algorithm allows to fine-tune when to stop the greedy algorithm. Using this procedure, it is possible to restrict the execution time and space required by GR-ALL-PATHS. For example, GR-ALL-PATHS can limit the total number of times Line 4 is executed (the number of intermediate paths examined), the total number of times Line 8 is executed, the maximum number of paths in A and so on. GR-ALL-PATHS achieves that by maintaining auxiliary statistic (count) variables that account for the number of times Line 4 is executed so far and other parameters. Then STOPCONDITION() simply check whether those statistic variables still satisfy the predefined constraints, and if not, GR-ALL-PATHS will stop looking for new paths and it will output the paths discovered so far as its result. Thus, GR-ALL-PATHS discovers most important paths first and least important paths last and can be stopped at a certain point whereas DF-ALL-PATHS discovers all paths.

5.3.2 *Paths Storage.* When looking for all L-short simple u-v paths, ALL-PATHS maintains several intermediate paths. To store paths compactly and efficiently, it uses a data structure called a *paths storage*. DF-ALL-PATHS and GR-ALL-PATHS procedures actually operates with *pointers* to paths while the paths themselves are stored in the paths storage.

Each path is stored as a list, in reverse order. The *paths storage* is organized as a set of *overlapping lists* as follows. Since all of the paths start from u, many

$$
\begin{aligned}
p_1 &= u \leftrightarrow 1\\
p_2 &= u \leftrightarrow 1 \leftrightarrow 2\\
p_3 &= u \leftrightarrow 1 \leftrightarrow 2 \leftrightarrow 3\\
p_4 &= u \leftrightarrow 1 \leftrightarrow 2 \leftrightarrow 4
\end{aligned}
$$

Fig. 18. Example of paths.

$$
\begin{aligned}
\ell_1 &= 1 \to u\\
\ell_2 &= 2 \to 1 \to u\\
\ell_3 &= 3 \to 2 \to 1 \to u\\
\ell_4 &= 4 \to 2 \to 1 \to u
\end{aligned}
$$

Fig. 19. Separate lists for paths.

$$
\begin{aligned}
\ell_1 &= 1 \to u\\
\ell_2 &= 2 \to \ell_1\\
\ell_3 &= 3 \to \ell_2\\
\ell_4 &= 4 \to \ell_2
\end{aligned}
$$

Fig. 20. The paths storage.

of them share common prefix. This gives an opportunity to save space. For example, to store paths shown in Figure 18, it is not necessary to keep four separate lists shown in Figure 19 of lengths 2, 3, 4, and 4, respectively. It is more efficient to store them as shown in Figure 20 where the combined length of the lists is just 8 nodes (versus 13 nodes when keeping separate lists). This storage is also *efficient* because the algorithm always knows where to find the right prefix in the storage—it does not need to scan the *paths storage* to find the right prefix. This is because when the algorithm creates a new intermediate path $p \leftrightarrow z$, the following holds:

(1) p is the prefix of $p \leftrightarrow z$
(2) p is already stored in the *path storage*
(3) the algorithm knows the pointer to p at this point

5.3.3 *Comparing Complexity of Greedy and Depth-First Implementations.* Let us analyze complexity of the depth-first and greedy implementations of ALL-PATHS procedure. The DF-ALL-PATHS and GR-ALL-PATHS procedures in Figures 16 and 17 are conceptually different only in Lines 2 and 3 of ALL-PATHS and in Line 4 of EXPAND-PATHS. The STOPCONDITION procedure in Line 3 allows to fine-tune when to stop the greedy algorithm and determines the complexity of GR-RELDC. But we will analyze only the differences in complexity which arise due to DF-RELDC using a stack and GR-RELDC using a priority queue. That is, we will assume STOPCONDITION always returns `false`.

For a stack, PUSH and POP procedure take $O(1)$ time. If n is the size of a priority queue, each GET and INSERT procedures take $O(\lg n)$ time [Cormen et al. 2001]. Therefore, it takes $O(1)$ time to process Lines 4–8 of DF-ALL-PATHS and it takes $O(\lg n)$ to process the same Lines 4–8 of DF-ALL-PATHS where n is the current size of the priority queue. Also it take $O(degree(x))$ time to execute DF-EXPAND-PATH procedure and it takes $O(degree(x) \cdot \lg{(n + degree(x))}$ to execute GR-EXPAND-PATH procedure.

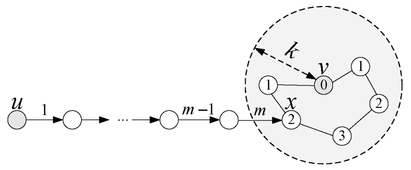

Fig. 21. Neighborhood.

Thus, if the goal is to discover all L-short simple paths *without skipping* any paths, then the Df-All-Paths is expected to show better results than Gr-All-Paths. However, since the greedy version discovers the most important path first, it is a better choice in terms of the 'accuracy versus performance' trade-off than its depth-first counterpart. Therefore, the greedy version is expected to be better if the execution time of the algorithm needs to be constrained.

5.4 NBH Optimization

The NBH optimization is the most important performance optimization presented in this article. It consistently achieves 1–2 orders of magnitude performance improvement under variety of conditions. The *neighborhood* $\mathcal{N}_k(v)$ of node v of radius k is the set of all the nodes that are reachable from v via at most k edges. Each member of the set is tagged with 'the minimum distance to v' information. That is, for graph $G = (V, E)$ the neighborhood of node v of radius k is defined as the following set of pairs: $\mathcal{N}_k(v) = \{(u, d) : u \in V,\ d = \mathit{MinDist}(u, v),\ d \leq k\}$.

Let us recall that, when resolving reference r, the algorithm invokes All-Paths algorithm N times, in order to compute $\mathcal{P}_L(x_r, y_{rj})$ for $j = 1, 2, \ldots, N$, see Figure 5. These computations can be optimized by (a) computing neighborhood $\mathcal{N}_k(x_r)$ once per each distinct x_r; (b) discovering paths not from x_r to y_{rj} but in reverse order: from y_{rj} to x_r; and (c) exploiting $\mathcal{N}_k(x_r)$ to prune certain intermediate paths as explained below.

When resolving references of entity x_r, the algorithm first computes the neighborhood $\mathcal{N}_k(x_r)$ of x_r of radius k, where $k \leq L$, see Figure 21. The neighborhood is computed only once per each distinct x_r and discarded after x_r is processed. There are two factors responsible for the speedup achieved by the NBH optimization:

(1) All-Paths(u, v), with the NBH optimization, first requires building $\mathcal{N}_k(v)$ and only then applying NBH-optimized All-Paths-NBH$(u, v, \mathcal{N}_k(u))$. Nevertheless, the computational cost of building $\mathcal{N}_k(v)$ and then executing All-Paths-NBH$(u, v, \mathcal{N}_k(v))$ is less than the cost of the nonoptimized version of All-Paths(u, v).
(2) The neighborhood $\mathcal{N}_k(x_r)$ is built *once* per each distinct x_r when processing y_{r1}. Then, it is reused when processing $y_{r2}, \ldots, y_{rN}$. After that, $\mathcal{N}_k(x_r)$ is discarded to save space, since it is not needed any longer.

All-Paths procedure shown in Figures 16 and 17 should be modified to be used with NBH. First, it should be able to accept an additional input parameter

```
PRUNE-PATH-NBH($p, \mathcal{N}_k(v)$)
1   $m \leftarrow Length(p)$ // here $p = u \rightsquigarrow x$
2   if $(m + k) < L$ and $k_{act} = k$ then
3       return false // do not prune
4   $x \leftarrow LastNode(p)$
5   if $x \notin \mathcal{N}_k(v)$ then
6       return true // prune
7   $d \leftarrow$ GET-MIN-DIST($x, \mathcal{N}_k(v)$)
8   if $(m + d) \leq L$ then
9       return false // do not prune
10  return true // prune
```

Fig. 22. PRUNE-PATH-NBH().

$\mathcal{N}_k(v)$. Second, Line 7 should be changed from

7 **else if** $Length(p) < L$ **then**

to

7 **else if** $Length(p) < L$ **and** PRUNE-PATH-NBH($p, \mathcal{N}_k(v)$) = `false` **then**

This will allow pruning certain paths using the NBH optimization. The PRUNE-PATH-NBH($p, \mathcal{N}_k(v)$) procedure is provided in Figure 22. It takes advantage of $\mathcal{N}_k(v)$ to decide whether a given intermediate path $p = u \rightsquigarrow x$ can be pruned. First, it determines the length m of path p. If m is such that $(m + k) < L$, then it cannot prune p, so it returns `false`. However, if $(m + k) \geq L$, then x must be inside $\mathcal{N}_k(v)$. If it is not inside, then path p is pruned, because there cannot be an L-short path $u \overset{p}{\rightsquigarrow} x \overset{p_1}{\rightsquigarrow} v$ for any path $p_1 : x \overset{p_1}{\rightsquigarrow} v$. If x is inside $\mathcal{N}_k(v)$, then the procedure retrieves from $\mathcal{N}_k(v)$ the minimum distance d from x to v. This distance d should be such that $(m + d) \leq L$: otherwise path p is pruned.

The NBH optimization can be improved further. Let us first define the *actual radius* of neighborhood $\mathcal{N}_k(v)$: $k_{act} = \max_{u:(u,d) \in \mathcal{N}_k(v)} MinDist(u, v)$. While usually $k_{act} = k$, sometimes[10] $k_{act} < k$. The latter happens when nodes from the neighborhood of v and their incident edges form a cluster which is not connected to the rest of the graph or when this cluster is the whole graph. In this situation $\mathcal{N}_{k_{act}}(v) = \mathcal{N}_\ell(v)$ for any $\ell = k_{act}, k_{act} + 1, \ldots, \infty$. In other words, when $k_{act} < k$, we know the neighborhood of v of radius $k = \infty$. Regarding PRUNE-PATH-NBH, this means that all intermediate nodes must always be inside the according neighborhood. This further improvement is reflected in Line 2 of the PRUNE-PATH-NBH procedure in Figure 22.

5.5 Storing Discovered Paths Explicitly

Once the paths are discovered on the first iteration of IT-RELDC, they can be exploited for speeding up the subsequent iterations when those paths need to be rediscovered again. One solution would be to store such paths explicitly. After paths are stored, the subsequent iterations do not rediscover them, but rather work with the stored paths. Next, we present several techniques that reduce the storage overhead of storing paths explicitly.

5.5.1 *Path Compression.* We store paths because we need to recompute the connection strengths of those paths (on subsequent iterations), which can

[10]Naturally, the greater the k the more frequently this is likely to occur.

change as weights of option-edges change. One way of compressing path information is to find fixed-weight paths. Fixed-weight paths are paths the connection strength of which will not change because it does not depend on any other system variables that can change. Rather than storing a path itself, it is more efficient to store the (fixed) connection strength of that path, which, in turn, can be aggregated with other fixed connection strengths. For WM model, a path connection strength is guaranteed to be fixed if none of the intermediate or source nodes on the path are incident to an option-edge (the weight of which might change).

5.5.2 *Storing Graph Instead of Paths.* Instead of storing paths one by one, it is more space efficient to store the connection subgraphs. The set of all L-short simple paths $\mathcal{P}_L(u, v)$ between nodes u and v defines the connection subgraph $\mathcal{G}(u, v)$ between u and v. Storing $\mathcal{G}(u, v)$ is more efficient because in $\mathcal{P}_L(u, v)$ some of the nodes can be repeated several times, whereas in $\mathcal{G}(u, v)$ each node occurs only once. Notice, when we store $\mathcal{P}_L(u, v)$ or $\mathcal{G}(u, v)$, we store only nodes: edges need not be stored since they can be restored from the original graph G. There is a price to pay for storing only $\mathcal{G}(u, v)$: the paths need to be rediscovered. However, this rediscovering happens in a small subgraph $\mathcal{G}(u, v)$ instead of the whole graph G.

5.6 Miscellaneous Implementation Issues

5.6.1 *Compatibility of Implementations.* In general, it is possible to combine various implementations and optimizations of RelDC. For example, there can be an implementation of RelDC that combines It-RelDC, Df-All-Paths, NBH, and the optimization that stores paths. However, certain implementations and optimizations are mutually exclusive. They are as follows:

(1) It-RelDC vs. Sl-RelDC
(2) Df-RelDC vs. Gr-RelDC
(3) Sl-RelDC and Storing Paths.

Let us note that there are some compatibility issues of Gr-RelDC with Sl-RelDC. Notice, Gr-RelDC computes the connection strengths of intermediate paths. Consequently, it must know weights of certain edges and, in general, it must know weights of option-edges. That is why Lines 1–2 of the Naïve-Sl-RelDC procedure assign to option-edge weights some initial values.

5.6.2 *Preventing Path Rediscovering.* Since the All-Paths algorithm is the bottleneck of the approach, once $\mathcal{P}_L(u, v)$ is computed for given nodes u and v, it should not be recomputed later on again for the same u and v, neither as $\mathcal{P}_L(u, v)$ nor as $\mathcal{P}_L(v, u)$. Currently, this issue does not arise in our implementation of RelDC due to a systematic way the processing is done. Specifically, for the datasets being tested, when computing $\mathcal{P}_L(u, v)$, all u's belong to one class of entities C_1 (e.g., publications), and all v's belong to a different class C_2 (e.g., authors). The algorithm first iterates over each entity u in C_1 and then over each reference r in entity u. After u is processed, the algorithm never comes back to it on the same iteration, so that the issue mentioned above cannot arise.

In general case, this issue can be handled by maintaining the set $S_{\mathcal{P}}$ of all (u, v) pairs for which $\mathcal{P}_L(u, v)$ is already computed by the algorithm. Specifically, after computing $\mathcal{P}_L(u, v)$, if $u.id < v.id$, the algorithm should store (u, v) in $S_{\mathcal{P}}$, and, if $u.id > v.id$, it should store (v, u) in $S_{\mathcal{P}}$. Here, $u.id$ and $v.id$ are unique identifiers of nodes u and v. To check whether $\mathcal{P}_L(u, v)$ has already been discovered, the algorithm should check whether (u, v) pair (if $u.id < v.id$), or (v, u) pair (if $u.id > v.id$) is in $S_{\mathcal{P}}$ yet. This simple procedure will allow the algorithm to prevent rediscovering $\mathcal{P}_L(u, v)$ multiple times, by performing a lookup in $S_{\mathcal{P}}$. Let us clarify that the purpose of the above "$u.id < v.id$" comparisons is to make one lookup in $S_{\mathcal{P}}$, instead of two.

5.7 Computational Complexity of RELDC

Let us analyze the computational complexity of nonoptimized IT-RELDC with GR-ALL-PATHS procedure. GR-ALL-PATHS procedure, provided in Section 5.3, discovers L-short simple $u \rightsquigarrow v$ paths such that it finds paths with the highest connection strength first and with the lowest last. It achieves that by maintaining the current connection strength for intermediate paths and by using a priority queue to retrieve the best (in terms of connection strength) intermediate path to expand next. GR-ALL-PATHS(u, v) maintains the connection strength of intermediate paths, so a straightforward modification of this procedure can return not only the desired set of paths but also the value of $c(u, v)$.

GR-ALL-PATHS has several thresholds that limit the number of nodes it can expand, the total number of edges it can examine, the length of each path, the total number of $u \rightsquigarrow v$ paths it can discover, and the total number of all paths it can examine. Those thresholds can be specified as constants, or as functions of $|V|$, $|E|$, and L. If they are constants, then the time and space complexity needed to compute $c(u, v)$ is limited by constants C_{time} and C_{space}.

Assume that there are N_{ref} references that need to be disambiguated, where typically $N_{ref} = O(|V|)$. The average cardinality of their option sets is typically a constant, or $O(|V|)$. Thus, IT-RELDC will need to compute $c(x_r, y_{rj})$ for at most $O(|V|^2)$ pairs of (x_r, y_{rj}) per iteration. Therefore, the time complexity of an iteration of IT-RELDC is $O(|V|^2)$ multiplied by the complexity of the GR-ALL-PATHS procedure, plus the cost to construct all option sets using an FBS approach, which is at most $O(|V|^2)$. The space complexity is $O(|V| + |E|)$ to store the graph plus the space complexity of one GR-ALL-PATHS procedure.

6. EXPERIMENTAL RESULTS

In this section we experimentally study RELDC using two real (*publications* and *movies*) and synthetic datasets. RELDC was implemented using C++ and SNOPT solver [GAMS solvers 2005]. The system runs on a 1.7GHz Pentium machine. We test and compare the following implementations of RELDC:

(1) *IT-RELDC vs. SL-RELDC*. The prefixes indicate whether the corresponding NLP problem discussed in Section 4.3 is solved *iteratively* or using a *solver*. If none of those prefixes is specified, IT-RELDC is assumed by default. SL-RELDC is applicable only to more restricted problems (e.g., smaller

Table II. Sample Content of the *Publication* Table Derived from CiteSeer

Paper ID	Author Name	Paper ID	Author Name
51470	Hector Garcia-Molina	641294	Surajit Chaudhuri
51470	Anthony Tomasic	641294	Venkatesh Ganti
351993	Hector Garcia-Molina	273193	Venkatesh Ganti
351993	Anthony Tomasic	273193	Johannes Gehrke
351993	Luis Gravano	273193	Raghu Ramakrishnan
641294	Luis Gravano	273193	Wei-Yin Loh

graphs and smaller values of L) than IT-RELDC. SL-RELDC is also slower than IT-RELDC.

(2) *WM-RELDC vs. PM-RELDC*. The prefixes indicate whether the weight-based model (WM) covered in Section 4.1.3, or the probabilistic model (PM) covered in the *electronic appendix*, has been utilized for computing connection strengths. By default, WM-RELDC is assumed.

(3) *DF-RELDC vs. GR-RELDC*. The prefixes specify whether the depth-first (DF) or greedy (GR) implementation of ALL-PATHS is used. By default DF-RELDC is assumed.

(4) *Various optimizations of RELDC* can be turned on or off. By default, optimizations from Section 5 are on.

In each of the RELDC implementations, the value of L used in computing the L-short simple paths is set to 7 by default. In this section, we will demonstrate that WM-DF-IT-RELDC is on of the best implementations of RELDC in terms of both accuracy and efficiency. That is why the bulk of our experiments use that implementation.

6.1 Case Study 1: The Publications Dataset

6.1.1 *Datasets.* In this section, we will introduce RealPub and SynPub datasets. Our experiments will solve *author matching (AM)* problem, defined in Section 2, on these datasets.

RealPub dataset. RealPub is a real data set constructed from *two* public-domain sources: CiteSeer[CiteSeer 2005] and HPSearch[HomePageSearch 2005]. CiteSeer is a collection of information about research publication created by crawling the Web. HPSearch is a collection of information about authors. HPSearch can be viewed as a set of ⟨`id`, `authorName`, `department`, `organization`⟩ tuples. That is, the affiliation consists of not just organization like in Section 2, but also of department. Information stored in CiteSeer is in the same form as specified in Section 2, that is ⟨`id`, `title`, `authorRef1`, `authorRef2`, . . . , `authorRefN`⟩ per each paper, where each `authorRef` reference is a one-attribute ⟨*author name*⟩ reference.

Tables II and III show sample content of two tables derived from CiteSeer and HPSearch based on which the corresponding entity-relationship graph is constructed for RELDC. Figure 23 shows a sample entity-relationship graph that corresponds to the information in those two tables.

The various types of entities and relationships present in RealPub are shown in Table IV. RealPub data consists of 4 *types* of entities: papers (255K), authors

Table III. Sample Content of the *Author* Table Derived from HPSearch. Author from CiteSeer not Found in HPSearch are Also Added

Author ID	Author Name	Organization	Department
1001	Hector Garcia-Molina	Stanford	cs.stanford
1002	Anthony Tomasic	Stanford	cs.stanford
1003	Luis Gravano	Columbia Univ.	cs.columbia
1004	Surajit Chaudhuri	Microsoft	research.ms
1005	Venkatesh Ganti	Microsoft	research.ms
1006	Johannes Gehrke	Cornell	cs.cornell
1007	Raghu Ramakrishnan	Univ. of Wisconsin	cs.wisc
1008	Wei-Yin Loh	Univ. of Wisconsin	stat.wisc

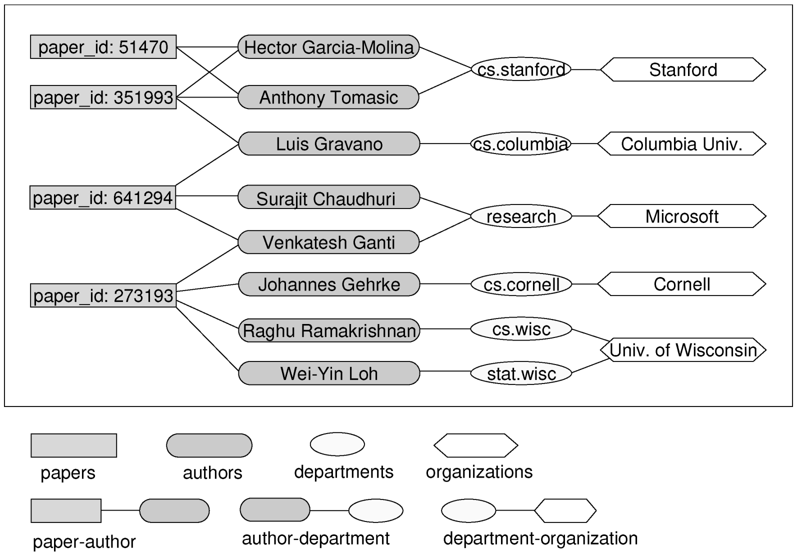

Fig. 23. Sample entity-relationship graph for *Publications* dataset. A paper with `paper_id` of, say 51470 can be retrieved from CiteSeer via URL: `http://citeseer.ist.psu.edu/51470.html`.

(176K), organizations (13K), and departments (25K). To avoid confusion, we use "`authorRef`" for author names in *paper* entities and "`authorName`" for author names in *author* entities. There are 573K `authorRef`'s in total. Our experiments on RealPub will explore the efficiency of RelDC in resolving these references.

To test RelDC, we first constructed an entity-relationship graph G for the RealPub database. Each regular node in the graph corresponds to an entity of one of these types. If author A is affiliated with department D, then there is (A, D) edge in the graph. If department D is a part of organization U, then there is (D, U) edge. If paper P is written by author A, then there is (A, P) edge. For each of the 573K `authorRef` references, feature-based similarity (FBS) was used to construct its option set.

In the RealPub data set, the paper entities refer to authors using *only their names (and not affiliations)*. This is because the paper entities are derived from the data available from CiteSeer, which did not directly contain information about the author's affiliation. As a result, only similarity of author names was used to initially construct the graph G.

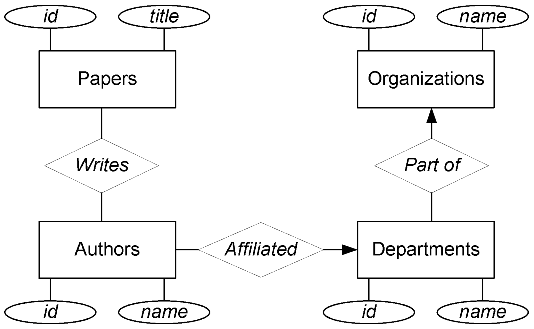

Fig. 24. E/R diagram for RealPub.

Table IV. No Affiliation in *Paper* Entities, thus FBS Cannot Use Affiliations

Entity Types	Entity Structure	Relationship Types
1. Paper	⟨pid, authorRef1, ..., authorRefN⟩	1. Author–Paper
2. Author	⟨aid, name, dept_id⟩	2. Author–Department
3. Department	⟨did, name, org_id⟩	3. Department–Organization
4. Organization	⟨oid, name⟩	

This similarity has been used to construct option sets for all authorRef references. As the result, 86.9% (498K) of all authorRef references had option set of size one and the corresponding papers and authors were linked directly. For the remaining 13.1% (75K) references, 75K choice nodes were created in the graph G. RELDC was used to resolve these remaining references. The specific experiments conducted and results will be discussed later in the section. Notice that the RealPub data set allowed us to test RELDC only under the condition that a majority of the references are already correctly resolved. To test robustness of the technique, we tested RELDC over synthetic data sets where we could vary the uncertainty in the references from 0 to 100%.

SynPub Dataset. We have created two synthetic datasets SynPub1 and SynPub2, which emulate RealPub. The synthetic data sets were created since, for the RealPub dataset, we do not have the true mapping between papers and the authors of those papers. Without such a mapping, as will become clear when we describe experiments, testing for accuracy of reference disambiguation algorithm requires a manual effort (and hence experiments can only validate the accuracy over small samples). In contrast, since in the synthetic data sets, the *paper-author* mapping is known in advance, accuracy of the approach can be tested over the entire data set. Another advantage of the SynPub dataset is that by varying certain parameters we can manually control the nature of this dataset allowing for the evaluation of all aspects of RELDC under various conditions (e.g., varying level of ambiguity/uncertainty in the data set).

Both the SynPub1 and SynPub2 datasets contain 5000 papers, 1000 authors, 25 organizations and 125 departments. The average number of choice nodes that will be created to disambiguate the authorRef's is 15K (notice, the whole RealPub dataset has 75K choice nodes). The difference between SynPub1 and

SynPub2 is that author names are constructed differently as will be explained shortly.

6.1.2 *Accuracy Experiments.* In the context, *accuracy* is defined as the fraction of all `authorRef` references that are resolved correctly. This definition includes the references that have option sets of cardinality 1. In all figures in this section, the higher accuracy of RelDC, compared to FBS, is the direct result of applying the CAP principle, discussed in Section 3.4, to the dataset being processed.

Experiment 1 (*RealPub: Manually Checking Samples for Accuracy*). Since the correct *paper-author* mapping is not available for RealPub, it is infeasible to test the accuracy on this dataset. However, it is possible to find a portion of this *paper-author* mapping *manually* for a sample of RealPub by going to authors web pages and examining their publications.

We have applied RelDC to RealPub in order to test the effectiveness of *analyzing relationships*. To analyze the accuracy of the result, we concentrated only on the 13.1% of uncertain `authorRef` references. Recall, the cardinality of the option set of each such reference is at least two. For 8% of those references, there were no $x_r \rightsquigarrow y_{rj}$ paths for all j's, thus RelDC used only FBS and not relationships. Since we want to test the effectiveness of analyzing relationships, we remove those 8% of references from further consideration as well. We then chose a random sample of 50 uncertain references that were still left under consideration. For this sample, we compared the reference disambiguation result produced by RelDC with the true matches. The true matches for `authorRef` references in those papers were computed manually. In this experiment, RelDC was able to resolve *all* of the 50 sample references correctly! This outcome is, in reality, not very surprising since in the RealPub data sets, the number of references that were ambiguous was only 13.1%. Our experiments over the synthetic data sets will show that RelDC reaches very high disambiguation accuracy when the number of uncertain references is not very high.

Ideally, we would have liked to perform further accuracy tests over RealPub by testing on larger samples: around 1,000 references should be tested to get an estimation of the accuracy within 3% error interval and 95% confidence. However, due to the time-consuming manual nature of this experiment, this was infeasible. Instead, we next present another experiment that studies accuracy of RelDC on the whole RealPub.

Experiment 2 (*RealPub: Accuracy of Identifying Author First Names*). We conducted another experiment over the RealPub data set to test the accuracy of RelDC in disambiguating references. We first remove from RealPub all the paper entities that have an `authorRef` in format "first initial + last name". This leaves only papers with `authorRef`'s in format "full first name + last name". Then, we pretend we only know "first initial + last name" for those `authorRef`'s. Next, we run FBS and RelDC and see whether or not they would disambiguate those `authorRef`'s to authors whose full first names coincide with the original full first names. In this experiment, for 82% of the `authorRef`'s the cardinality of their option sets is 1 and there is nothing to resolve. For the rest

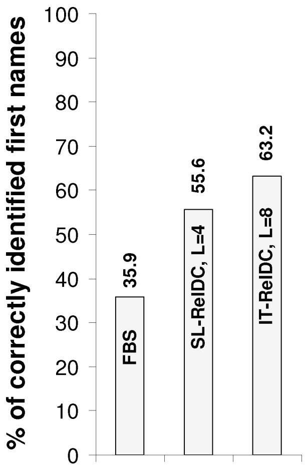

Fig. 25. RealPub: Identifying first names.

18% the problem is more interesting: the cardinality of their option sets is at least 2. Figure 25 shows the outcome for those 18%.

Notice that the reference disambiguation problem tested in the above experiment is of a limited nature. The tasks of identifying (a) the correct first name of the author and (b) the correct author, are not the same in general.[11] Nevertheless, the experiment allows us to test the accuracy of RELDC over the entire database and does show the strength of the approach.

Let us compare Experiments 1 and 2. Experiment 1 addresses the lack of the *paper-author* mapping by requiring laborious manual work and allows only testing on a sample of authors. Experiment 2 does not suffer from those drawbacks. However, Experiment 2 introduces substantial uncertainty to data by assuming that only the first initial instead of the full first name is available for each `authorRef`. Knowing the full first name in an `authorRef`, instead of just the first initial, would have allowed to significantly narrow down the option set for this `authorRef` and, thus, improve the accuracy of disambiguating this and, potentially, other references. To address the drawbacks of Experiments 1 and 2, we next study the approach on synthetic datasets.

Accuracy on SynPub. The next set of experiments tests accuracy of RELDC and FBS approaches on SynPub dataset. "RELDC 100%" ("RELDC 80%") means for 100% (80%) of *author* entities the affiliation information is available. Once again, *paper* entities do not have author affiliation attributes, so FBS cannot use affiliation, see Table IV. Thus, those 100% and 80% have no effect on the outcome of FBS. Notation "L=4" means RELDC explores paths of length no greater than 4.

Experiment 3 (*Accuracy on SynPub*1). SynPub1 uses *uncertainty of type* 1 defined as follows. There are $N_{auth} = 1000$ unique authors in SynPub1, but

[11]That is, it is not enough to correctly identify that 'J.' in 'J. Smith' corresponds to 'John' if there are multiple 'John Smith's in the dataset.

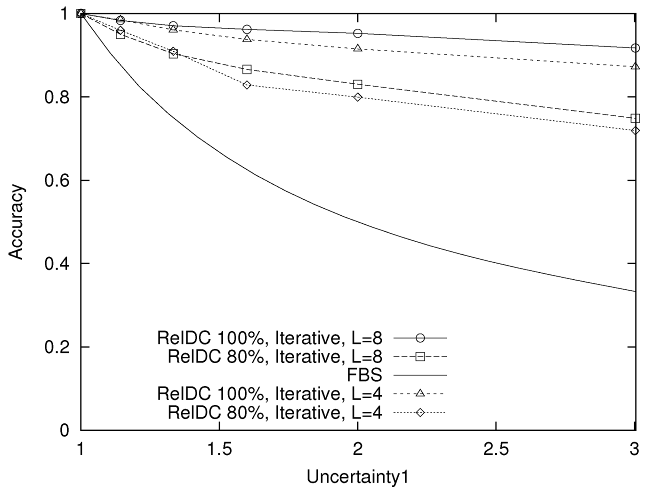

Fig. 26. SynPub1: Accuracy vs. unc_1.

there are only N_{name}, where $1 \leq N_{name} \leq N_{auth}$, unique `authorName`'s. We construct the `authorName` of the author with `id` $= k$, for $k = 0, 1, \ldots, 999$, as 'name' concatenated with (k mod N_{name}). Each `authorRef` specifies one of those `authorName`'s. Parameter unc_1 is $unc_1 = \frac{N_{auth}}{N_{name}}$ ratio. For instance, if $N_{name} = 750$, then the authors with `id` $= 1$ and `id` $= 751$ have the same `authorName` $=$ 'name1', and $unc_1 = \frac{1000}{750} = 1\frac{1}{3}$. In SynPub1, for each author whose name is not unique, one can never identify with 100% confidence any paper this author has written. Thus, the uncertainty for such authors is very high.

Figure 26 studies the effect of unc_1 on accuracy of RelDC and FBS. If $unc_1 = 1.0$, then there is no uncertainty and all methods have accuracy of 1.0. As expected, the accuracy of all methods monotonically decreases as uncertainty increases. If $unc_1 = 2.0$, the uncertainty is very high: for any given author, there is exactly one other author with the identical `authorName`. For this case, any FBS have no choice but to guess one of the two authors. Therefore, the accuracy of any FBS, as shown in Figure 26, is 0.5. However, the accuracy of RelDC 100% (RelDC 80%) when $unc_1 = 2.0$ is 94%(82%). The gap between RelDC 100% and RelDC 80% curves shows that in SynPub1 RelDC relies substantially on author affiliations for the disambiguation.

Comparing the RelDC *implementations.* Figure 27 shows that the accuracy results of WM-It-RelDC, PM-It-RelDC, WM-Sl-RelDC implementations are comparable, whereas Figure 28 demonstrates that WM-It-RelDC is the fastest among them.

Experiment 4 (*Accuracy on SynPub2*). SynPub2 uses *uncertainty of type 2*.

In SynPub2, `authorName`'s (in *author* entities) are constructed such that the following holds, see Table IV. If an `authorRef` reference (in a *paper* entity) is in the format "first name + last name" then it matches only one (correct) author.

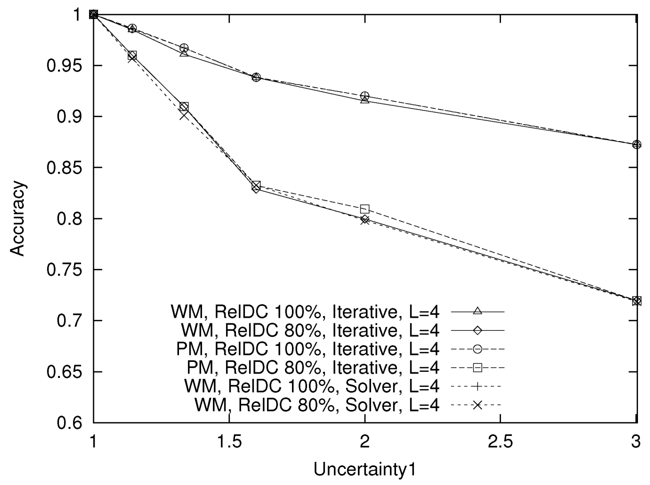

Fig. 27. SynPub1: The accuracy results for SL-RELDC, IT-RELDC, and IT-RELDC with PM model are comparable.

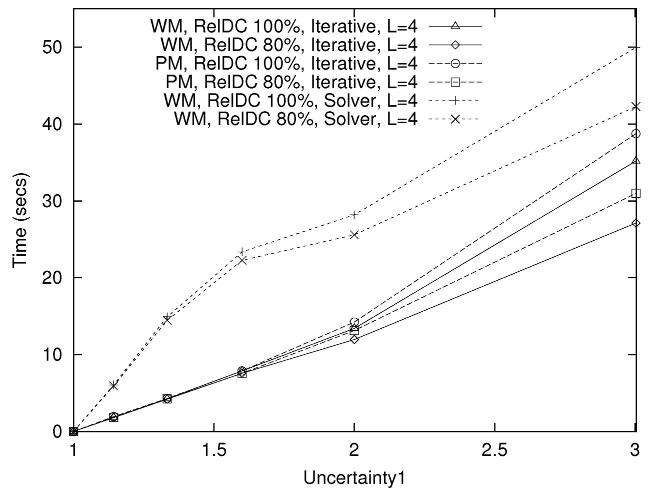

Fig. 28. SynPub1: IT-RELDC is more efficient than (i)SL-RELDC and (ii)IT-RELDC with PM model.

But if it is in the format "first initial + last name" it matches exactly two authors. Parameter unc_2 is the fraction of `authorRef`'s specified as "first initial + last name". If $unc_2 = 0$, then there is no uncertainty and the accuracy of all methods is 1. Also notice that the case when $unc_2 = 1.0$ is equivalent to $unc_1 = 2.0$.

There is less uncertainty in Experiment 4 than in Experiment 3. This is because for each author there is a chance that he is referenced to by his full name in some of his papers, so for these cases the *paper-author* associations are known with 100% confidence.

Figure 29 shows the effect of unc_2 on the accuracy of RELDC. As in Figure 26, in Figure 29 the accuracy decreases as uncertainty increases. However, this

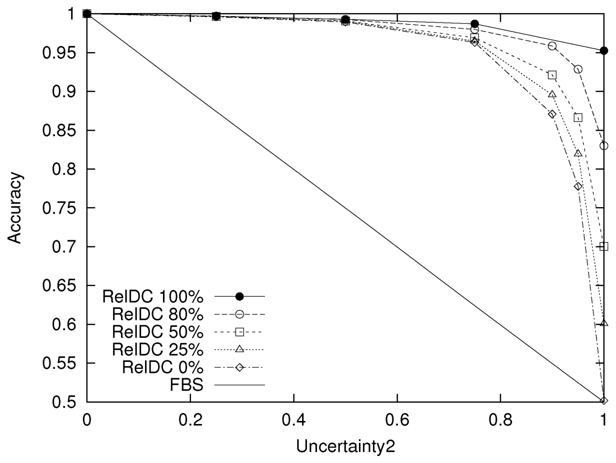

Fig. 29. SynPub2: Accuracy vs. unc_2.

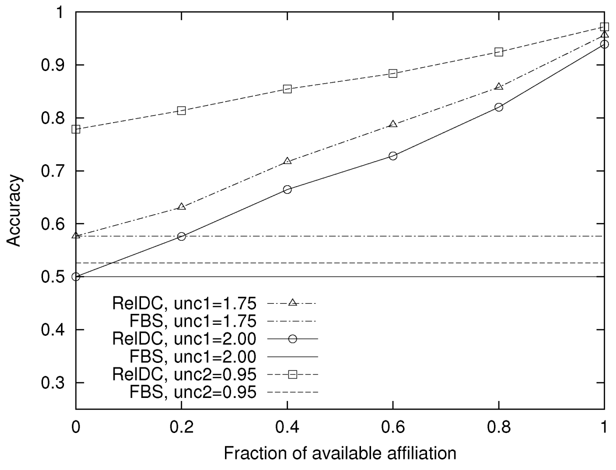

Fig. 30. SynPub: Accuracy vs. fraction of available affiliation.

time the accuracy of RelDC is much higher. The fact that curves for RelDC 100% and RelDC 80% are almost indiscernible until unc_2 reaches 0.5, shows that RelDC relies less heavily on weak author affiliation relationships but rather on stronger connections via papers.

6.1.3 *Other Experiments*

Experiment 5 (*Importance of Relationships*). Figure 30 studies the effect of the number of relationships and the number of relationship *types* on the accuracy of RelDC. When resolving `authorRef`'s, RelDC uses three types of relationships: (1) *paper-author*, (2) *author-department*, (3) *department-organization*.[12]

[12]Note, there is a difference between a *type of relationship* and a *chain of relationships*: for example, RelDC can discover paths like: paper1-author1-dept1-org1-dept2-author2.

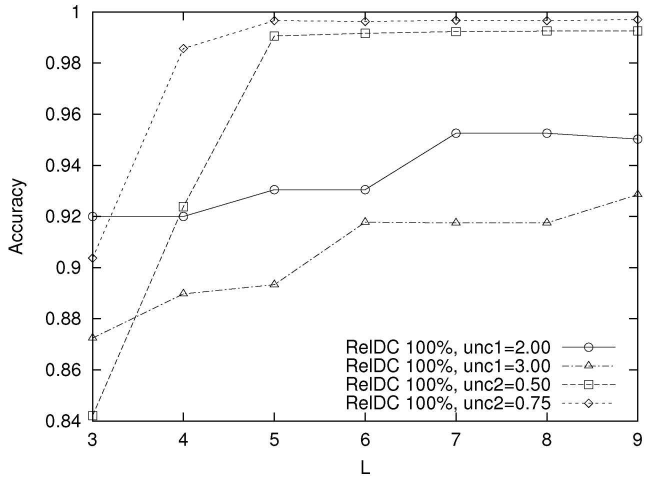

Fig. 31. SynPub: Accuracy vs. L.

The affiliation relationships (i.e., (2) and (3)) are derived from the affiliation information in *author* entities.

The affiliation information is not always available for each *author* entity in RealPub. In our synthetic datasets, we can manually vary the amount of available affiliation information. The x-axis shows the fraction ρ of *author* entities for which their affiliation is known. If $\rho = 0$, then the affiliation relationships are eliminated completely and RELDC has to rely solely on connections via *paper-author* relationships. If $\rho = 1$, then the complete knowledge of author affiliations is available. Figure 30 studies the effect of ρ on accuracy. The curves in this figure are for both SynPub1 and SynPub2: $unc_1 = 1.75$, $unc_1 = 2.00$, and $unc_2 = 0.95$. The accuracy increases as ρ increases showing that RELDC deals with newly available relationships well.

Experiment 6 (*Longer Paths*). Figure 31 examines the effect of path limit parameter L on the accuracy. For all the curves in the figure, the accuracy monotonically increases as L increases with the only one exception for "RELDC 100%, unc1 = 2" and $L = 8$. The usefulness of longer paths depends on the combination of other parameters. For SynPub, L of 7 is a reasonable compromise between accuracy and efficiency.

Experiment 7 (*The Neighborhood Optimization*). We have developed several optimizations that make RELDC 1–2 orders of magnitude more efficient. Figure 32 shows the effect of one of those optimizations, called NBH (see Section 5.4), for subsets of 11K and 22K papers of CiteSeer. In this figure, the radius of neighborhood is varied from 0 to 8. The radius of zero corresponds to the case where NBH is not used. Figure 33 shows the speedup achieved by NBH optimization with respect to the case when NBH is off. The figure shows another positive aspect of NBH optimization: the speed up grows as the size of the dataset and L increase.

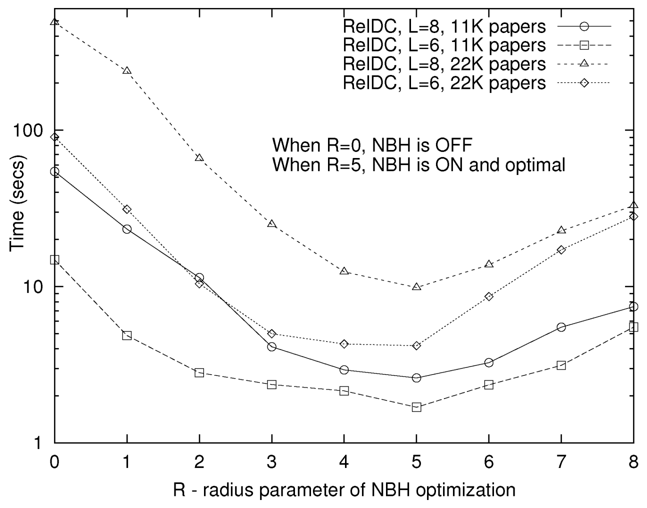

Fig. 32. RealPub: Optimizations are crucial.

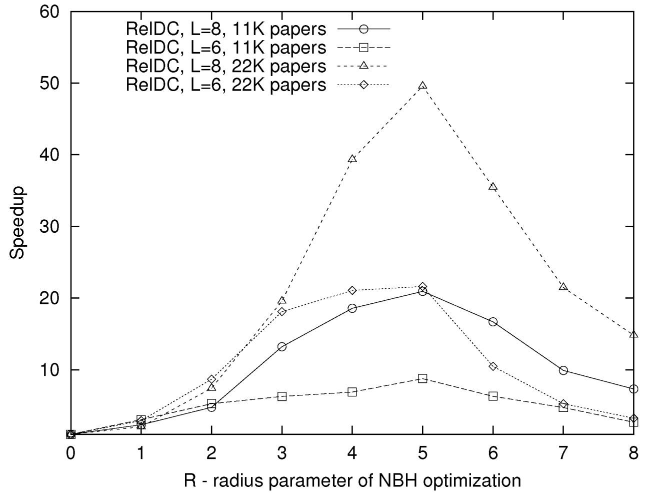

Fig. 33. RealPub: Speedup achieved by NBH.

Experiment 8 (*Efficiency of* RELDC). To show the applicability of RELDC to a large dataset we have successfully applied it to clean RealPub with L ranging from 2 up to 8. Figure 34 shows the execution time of RELDC as a function of the fraction of papers from RealPub dataset, for example, 1.0 corresponds to all papers in RealPub (the whole CiteSeer) dataset.

Experiment 9 (*Greedy vs. Depth-First* ALL-PATHS *Implementations*). This experiment compares accuracy and performance of greedy and depth-first versions of RELDC. The depth-first version discovers exhaustively *all* paths in a depth-first fashion. RELDC has been heavily optimized and this discovery process is very efficient. The greedy implementation of ALL-PATHS discovers

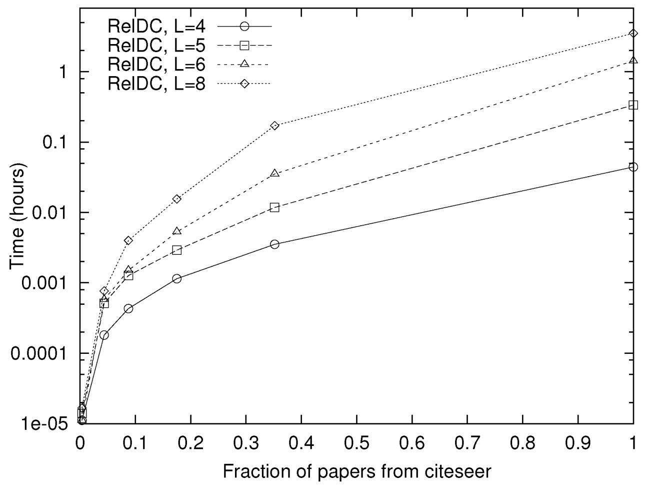

Fig. 34. RealPub: Time vs. Fraction of RealPub.

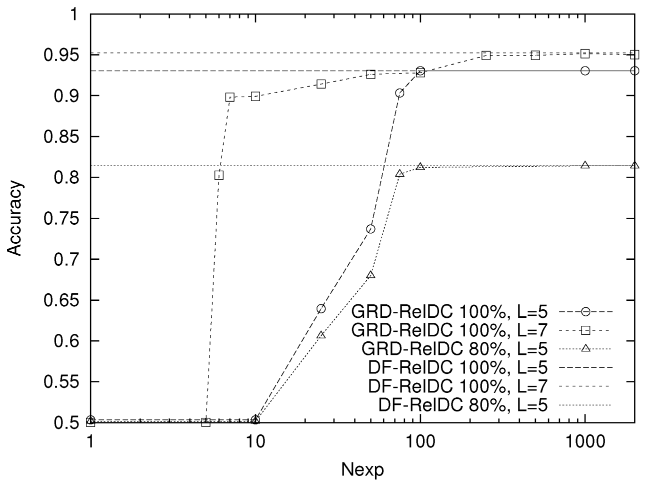

Fig. 35. SynPub1: $unc_1 = 2$. Accuracy of GR-RELDC vs. DF-RELDC.

paths with the best connection strength first and with the worst last. This gives an opportunity to fine-tune in a meaningful way when to stop the algorithm by using various thresholds. Those thresholds can limit, for example, not only path length but also the memory that all intermediate paths can occupy, the total number of paths that can be analyzed, and so on.

Figures 35 and 36 study the effect of N_{exp} parameter on the accuracy and efficiency of GR-RELDC and DF-RELDC. Parameter N_{exp} is the *upper bound* on the number of paths that can be extracted from the priority queue for GR-RELDC. The ALL-PATHS part of GR-RELDC stops if either N_{exp} is exceeded or the priority queue is empty.

The series in the experiment are obtained by varying: (1) DF-RELDC and GR-RELDC, (2) path length limit $L = 5$ and $L = 7$ and (3) the amount of affiliation

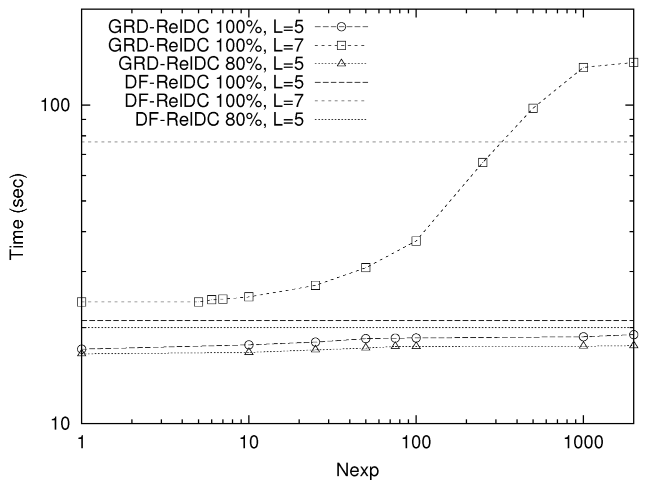

Fig. 36. SynPub1: $unc_1 = 2$. Time of GR-RELDC vs. DF-RELDC.

information 100% and 80%. Since DF-RELDC does not use N_{exp} parameter, all DF-RELDC curves are flat. Let us analyze what behavior is expected from GR-RELDC and then see if the figures corroborate it. We will always assume that *path length is limited* for both DF-RELDC and GR-RELDC.

If N_{exp} is small, then GR-RELDC should discover only a few paths and its accuracy should be close to that of FBS. If N_{exp} is sufficiently large, then GR-RELDC should discover the same paths as DF-RELDC. That is, we can compute $m_{rj} = |\mathcal{P}_L(x_r, y_{rj})|$, where $|\mathcal{P}_L(x_r, y_{rj})|$ is the number of paths in $\mathcal{P}_L(x_r, y_{rj})$. Then, if we choose N_{exp} such that $N_{exp} \geq \max_{r,j}(m_{rj})$, then the set of all paths that GR-RELDC will discover will be identical to that of DF-RELDC. Thus, the accuracy of GR-RELDC is expected to increase monotonically and then stabilize (and be equal to the accuracy of DF-RELDC) as N_{exp} increases. The execution time of GR-RELDC should increase monotonically and then stabilizes as well (and be larger than the execution time of DF-RELDC after stabilizing).

The curves in Figures 35 and 36 behave as expected except for one surprise: when $L = 5$, GR-RELDC is actually faster than DF-RELDC. It is explained by the fact that when $L = 5$, NBH optimization prunes very effectively many paths.

That keeps the priority queue small. Thus, the performance of DF-RELDC and GR-RELDC becomes comparable. Notice, in all of the experiments NBH optimization was turned on, because the efficiency of any implementation of RELDC with NBH off is substantially worse than the efficiency of any implementation with NBH on.

Figure 37 combines Figures 35 and 36. It plots the achieved accuracy by DF-RELDC and GR-RELDC when $L = 5$ and $L = 7$ as a function of time. Using this figure, it is possible to perform a retrospective analysis of which implementation has shown the best accuracy when allowed to spend only at most certain amount of time t on the cleaning task. For example, in time interval [0, 17.7) RELDC cannot achieve better accuracy than FBS, so it is more efficient just to use FBS. In time interval [17.7, 18.6), it is better to use GR-RELDC with $L = 5$. If one

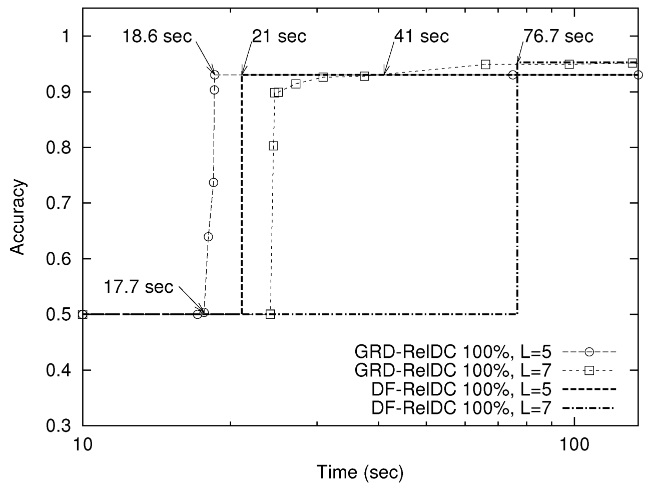

Fig. 37. SynPub1: $unc_1 = 2$. Choosing the best among Gr-RelDC $L = 5$, Gr-RelDC $L = 7$, Df-RelDC $L = 5$, Df-RelDC $L = 7$ at each moment in time.

is allowed to spend only [18.6, 41) seconds, it is better to use Gr-RelDC with $L = 5$ for only 18.6 seconds. If you intend to spend between 41 and 76.7 seconds it is better to use Gr-RelDC with $L = 7$. If you can spend 76.7 seconds or more, it is better to run Df-RelDC with $L = 7$, which will terminate in 76.7 seconds.

6.2 Case Study 2: The Movies Dataset

6.2.1 *Dataset.* RealMov is a real public-domain movies dataset described in [Wiederhold 2005], which has been made popular by the textbook [Garcia-Molina et al. 2002]. Unlike RealPub dataset, in RealMov all the needed correct mappings are known, so it is possible to test the disambiguation accuracy of various approaches more extensively. However, RealMov dataset is much smaller, compared to RealPub. RealMov contains entities of three types: *movies* (11, 453 entities), *studios* (992 entities), and *people* (22, 121 entities). There are five types of relationships in the RealMov dataset: *actors*, *directors*, *producers*, *producingStudios*, and *distributingStudios*. Relationships *actors*, *directors*, and *producers* map entities of type *movies* to entities of type *people*. Relationships *producingStudios* and *distributingStudios* map *movies* to *studios*. Figure 38 illustrate a sample graph for RealMov dataset, and Figure 39 shows its E/R diagram.

6.2.2 *Accuracy Experiments*

Experiment 10 (*RealMov: Accuracy of Disambiguating Director References*). In this experiment, we study the accuracy of disambiguating references from movies to directors of those movies.

Since in RealMov each reference, including each *director* reference, already points directly to the right match, we artificially introduce ambiguity in the references manually. Similar approach to testing data cleaning algorithms have also been commonly used by other researchers, for example, in Singla and

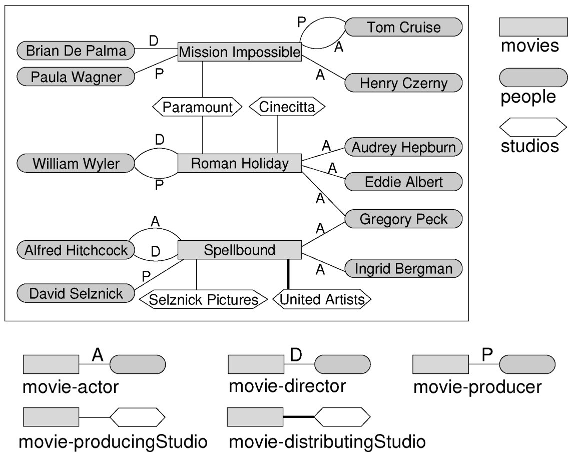

Fig. 38. Sample entity-relationship graph for *movies* dataset.

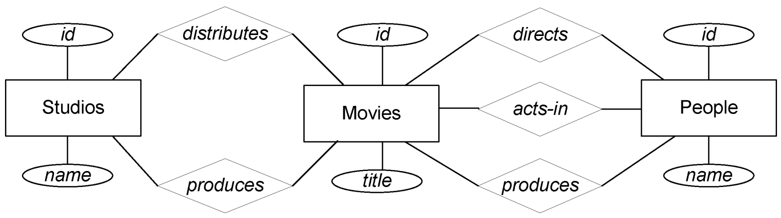

Fig. 39. E/R diagram for RealMov.

Domingos [2004] and Chaudhuri et al. [2003]. Given the specifics of our problem, to study the accuracy of RelDC, we will simulate that we used FBS to determine the option set of each reference but FBS was uncertain in some of the cases.

To achieve that, we first choose a fraction ρ of *director* references (that will be uncertain). For each reference in this fraction, we will simulate that FBS part of RelDC has done its best but still was uncertain as follows. Each *director* reference from this fraction is assigned a option set of N people. One of those people is the true director, the rest $(N - 1)$ are chosen randomly from the set of *people* entities.

Figure 41 studies the accuracy as ρ is varied from 0 to 1 and where N is distributed according to the probability mass function (pmf) shown in Figure 42.[13]

Figure 40 is similar to Figure 41 but N is always 2. The figures show that RelDC achieves better accuracy than FBS. The accuracy is 1.0 when $\rho = 0$, since all references are linked directly. The accuracy decreases almost linearly as ρ increases to 1. When $\rho = 1$, the cardinality of the option set of each reference is at least 2. The larger the value of L, the better the results. The accuracy of RelDC improves significantly as L increases from 3 to 4. However, the

[13]The distribution in Figure 42 is computed as taking integer part of the value of a random variable distributed according to the normal distribution with mean of 3.0 and standard deviation of 3.0. Values are regenerated until they fall inside the [2, 20] interval.

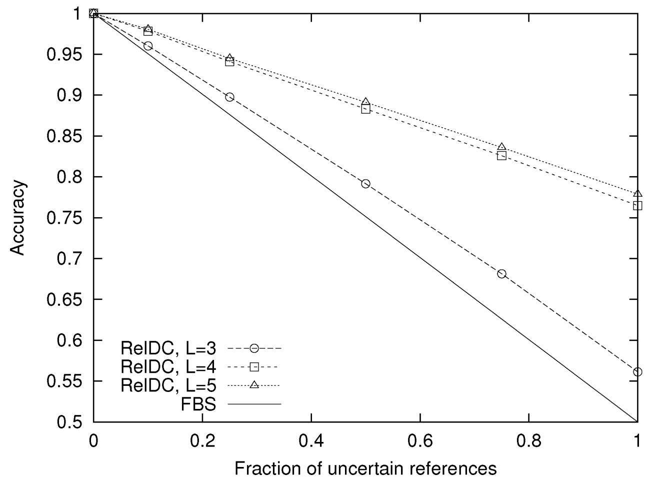

Fig. 40. RealMov: disambiguating *director* references. The size of the option set of each *uncertain* reference is 2.

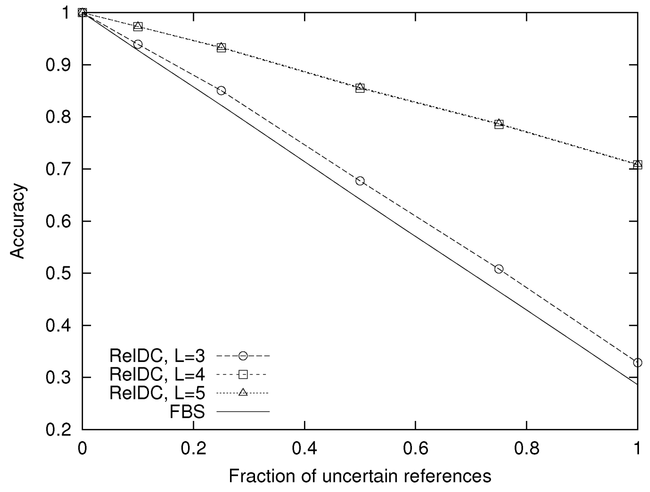

Fig. 41. RealMov: disambiguating *director* references. The pmf of sizes of option sets of *uncertain* references is given in Figure 42.

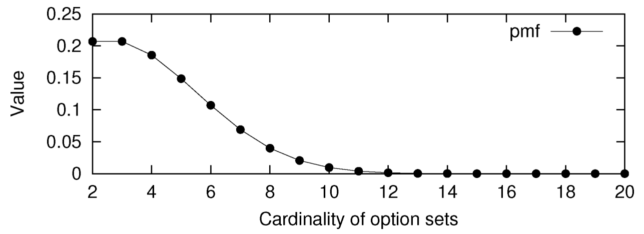

Fig. 42. PMF of sizes of option sets.

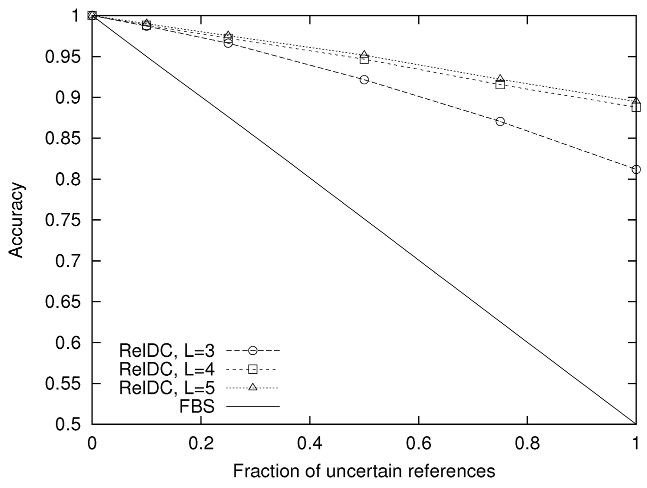

Fig. 43. RealMov: disambiguating *studio* references. The size of the option set of each *uncertain* reference is 2.

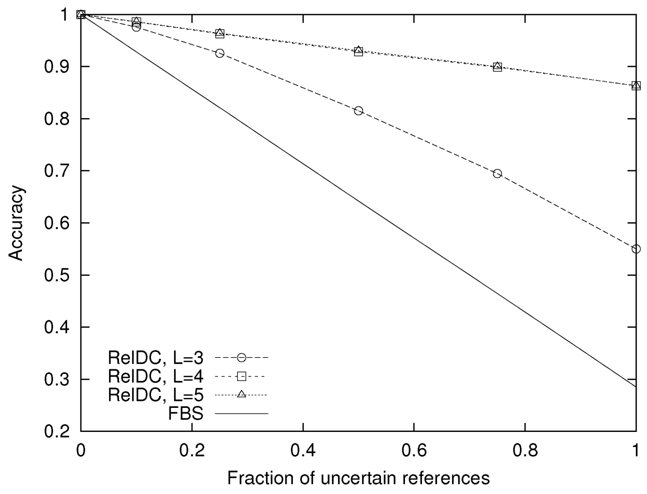

Fig. 44. RealMov: disambiguating *studio* references. The pmf of sizes of option sets of *uncertain* references is given in Figure 42.

improvement is less significant as L increases from 4 to 5. Thus, the analyst must decide whether to spend more time to obtain higher accuracy with $L = 5$, or whether $L = 4$ is sufficient.

Experiment 11 (*RealMov: Accuracy of Disambiguating Studio References*). This experiment is similar to the previous Experiment 10, but now we disambiguate *producingStudio* references, instead of *director* references. Figure 43 corresponds to Figure 40 and Figure 44 to Figure 41. The RelDC's accuracy of disambiguating *studio* references is even higher.

7. RELATED WORK

In recent years the data-cleaning challenge has attracted numerous efforts both in the industry as well as academia [Raedt et al. 2001; Getoor 2001; Cohen and Richman 2002; Monge and Elkan 1996; Gravano et al. 2001; Verykios et al. 2003; Christen et al. 2002; Cohen 1998; Sarawagi and Bhamidipaty 2002; Ristad and Yianilos 1998; Cohen et al. 2003]. In this section, we present an overview of the existing work most related to the RELDC approach proposed in this article. We will classify data-cleaning approaches along three dimensions.

The first dimension is the *type* of data-cleaning problem a particular approach solves. Many types of data-cleaning problems have been identified and explored in the literature: dealing with missing data [Little and Rubin 1986], handling erroneous data [Maletic and Marcus 2000], disambiguation and entity resolution [Ananthakrishna et al. 2002] etc. This article focuses on one of the *disambiguation* problems, called reference disambiguation. In general, there are many variations of disambiguation problems studied by various research communities, such as record linkage [Fellegi and Sunter 1969; Lee et al. 1999; Monge and Elkan 1997; Jaro 1989, 1995], reference matching [McCallum et al. 2000], object identification [Tejada et al. 2002], identity uncertainty [Pasula et al. 2002], name disambiguation [Li et al. 2004; Malin 2005], reference reconciliation [Dong et al. 2005], and so on; though sometimes different names are used for virtually identical problems. In the general setting of the problem, the dataset contains information about the set of objects $O = \{o_1, o_2, \dots, o_n\}$. The objects in the dataset are represented by the set of their descriptions $R = \{r_1, r_2, \dots, r_m\}$, where $m \geq n$, such that each object is represented by one or more descriptions. The subtle differences among the various disambiguation challenges arise because different types of extra information about R and O can be available in each particular problem. The most studied disambiguation challenges include:

—*Reference Disambiguation.* The problem of *reference disambiguation* [Kalashnikov et al. 2005], which is also known as *fuzzy match* [Chaudhuri et al. 2003], assumes that some information about each object in O is available and the goal is for each reference $r_i \in R$ to identify the object $o_j \in O$ to which it refers.

—*Object Consolidation.* In *object consolidation* [McCallum and Wellner 2003] it is assumed that very limited information is available about O. The goal is to correctly group the representations in R that refer to the same object.

—*Record Linkage.* The problem of *record linkage* is similar to the problem of object consolidation. Unlike object consolidation, which deals with objects, record linkage deals with lower-level representation of information—records in one or more tables. The goal is to identify duplicate records in the database. A record, in general, does not have to represent an object from O. Each record has the same fixed set of attributes. It is often assumed that each record has many attributes, which can be very effectively employed for de-duplication. In contrast, in object consolidation often only a few attributes can be available, making the problem more complex [Dong et al. 2005].

The second dimension we use to classify data-cleaning methods is whether a given method employs the standard FBS approach or a more advanced technique. The difference between them will be covered in Sections 7.1 and 7.2.

Finally, an approach can be domain-specific or domain-independent. Let us note that it can be hard to draw a line between domain-specific and domain-independent data-cleaning techniques. In distinguishing between domain independent and dependent techniques, we will follow the classification developed in Chaudhuri et al. [2005] where the authors define domain-specific approaches as those where the analyst, supervising the cleaning process, has to handcraft rules and logic for cleaning, which are applicable only to the specific domain. Let us note that many of the prevalent industrial solutions for data cleaning by companies such as Trillium, Vality, FirstLogic, DataFlux are domain specific. In this section, we focus instead on domain-independent techniques, since they are more directly related to our work. We note that in the RelDC project, we have taken an even more stringent definition of domain-independence than suggested by Chaudhuri et al. [2005]. In our view, a domain-independent solution should be such that (a) it can be incorporated into a real DBMS, (b) be applicable to different types of dataset, (c) scale to datasets of reasonable size, and (d) require minimum participation from the analyst so that even regular (nonexpert) database users can use it. Those have been the guiding principles in designing RELDC. To fully achieve this goal, our ongoing work [Kalashnikov and Mehrotra 2004] studies the issue of learning connection strength models directly from data.

7.1 Traditional FBS Methods

Many domain-independent FBS-based techniques for data cleaning have been proposed in the literature, including Newcombe et al. [1959], Fellegi and Sunter [1969], Hernandez and Stolfo [1995], Ananthakrishna et al. [2002], McCallum et al. [2000], and Winkler [1994, 1999]. Their work can be viewed as addressing two challenges: (1) improving similarity function, as in Bilenko and Mooney [2003]; and (2) improving efficiency of linkage, as in Chaudhuri et al. [2003]. 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, for example, SVM, to learn the best similarity function for a given domain [Bilenko and Mooney 2003]. Many techniques have been proposed to address the efficiency challenge as well: for example, using specialized indexes [Chaudhuri et al. 2003], sortings, etc.

7.2 Methods that Go beyond FBS

The domain-independent techniques mentioned above deal only with attributes. A number of data-cleaning techniques have been developed recently that go beyond the traditional FBS approaches. In particular, many of them in addition to attributes/features, can also take into account so-called 'context attributes', or attributes derived from the context.

For example, Ananthakrishna et al. [2002] employ similarity of directly linked entities, for the case of hierarchical relationships, to solve the record de-duplication challenge. The example the authors consider is as follows: two records 'United States' and 'USA' might have similar context attributes 'CA', 'IN', 'MO' while the records 'Canada' and 'USA' do not have the same context attributes. This fact might suggest that 'United States' and 'USA' are duplicates whereas 'Canada' and 'USA' are not. Let us note that there is no straightforward way to apply this technique to clean author-references in publications. For example, if we were to apply that technique, author-references should play the role of countries, for example, 'United States', and publications will play the role of context attributes, for example, states. First, the relationship author-writes-paper is not a hierarchical relationship. Second, there are no duplicate papers. The latter means if for two author-references their context attributes match, then the two authors are merely co-authors of the same paper, but they are not the same person and must not be de-duplicated. Therefore, such context attributes are of little value in our case and the approach in Ananthakrishna et al. [2002] cannot perform better than FBS methods on our datasets. Bhattacharya and Getoor [2004] propose an interesting object consolidation approach that is related to Ananthakrishna et al. [2002], but does not limit itself to hierarchical relationships only.

Lee et al. [2004] develop an association rule mining based method to disambiguate references using similarity of the context attributes. The proposed technique is still an FBS method, but the paper discusses 'concept hierarchies', which are related to relationships. In their approach, the authors rely on more information/attributes than is available in our datasets and the exact hierarchies they employ are not yet available to others for testing purposes. Overall, however, this is a very promising data-cleaning approach in our view, especially if it is enhanced to make it less analyst dependent.

Cohen et al. [2000] the authors study the problem of 'hardening soft databases', which is related to the problem of object consolidation. While that work does not consider relationships, it is similar to RelDC as it also attempts to find an optimal 'global' solution, rather than using a 'local' approach, such as merging references one by one. Specifically, the authors try to determine what is the most likely hard database, that corresponds to a particular soft database, under the assumptions of their model. The authors prove that the task of finding such a hard database is NP-hard, and instead propose a priority-queue-based algorithm. However, the empirical evaluation, model calibration and many implementation details are amiss in Cohen et al. [2000], making it hard to compare against other object consolidation techniques, such as Chen et al. [2005].

In Pasula et al. [2002], the authors study the problem of identity uncertainty (object consolidation, fuzzy grouping) and specifically its application to the problem of citation matching. The approach builds on a relational probability model and is capable of simultaneous reasoning about various types of references. The authors demonstrate, on datasets consisting of 300 publications, that the approach achieves high disambiguation quality. Unlike

RelDC and many other data-cleaning methods, the approach in Pasula et al. [2002] is analyst-dependent and rather complex for a regular user [Singla and Domingos 2004; McCallum and Wellner 2004], as it requires an analyst (an expert user) with deep understanding of the approach, probabilistic modeling, and nature/aspects of many components of the domain. The analyst should be able to model all the dependencies in the dataset and be capable of identifying the right external datasets/sources for model calibration. The authors suggest that scaling up the approach to 300 publications is a challenge since it employs the MCMC algorithm and discuss a way to improve the efficiency. However, the efficiency of the approach is not evaluated empirically.

In Singla and Domingos [2004], McCallum and Wellner [2004], and Dong et al. [2005] the authors propose relational object consolidation techniques where pair-wise matching decisions are not made independently, but rather a decision of whether given attributes match, or do not match, can further trigger similar decisions about more attributes, which ultimately leads to grouping or not grouping of various object representations. The authors demonstrate that such an approach leads to better consolidation quality, compared to the standard approach, where the pair-wise matching decisions are made independently.

Another interesting solution is proposed in Li et al. [2004], where the authors study the problem of 'robust reading'—a natural language problem that corresponds to the problem of object consolidation and fuzzy grouping in database research. The authors show that grouping documents originated in about the same time period and analyzing co-occurrences of mentions of locations and organizations, found in the documents, can improve the quality of object consolidation, across those documents.

7.3 RelDC Approach

To better distinguish RelDC from other related work, we next review some of its pertinent aspects. RelDC has been first publicly released in Kalashnikov and Mehrotra [2003] as a domain-independent data-cleaning framework that employs analysis of interobject relationships to achieve better quality of the result. RelDC enhances, but does not substitute, the traditional domain-independent FBS methods, as it applies an analysis of relationships only to the cases which FBS methods cannot resolve with high confidence. This article concentrates on one data-cleaning challenge, known as reference disambiguation or fuzzy lookup. In Chen et al. [2005] we have applied similar techniques to another related data-cleaning challenge, known as object consolidation.

Relationships. RelDC is based specifically on analyzing inter-object (chains of) relationships, where 'relationships' has the same definition as in the standard E/R model [Garcia-Molina et al. 2002]. There is a difference between what is known as 'relational' approaches, such as Singla and Domingos [2004], and techniques that analyze interobject relationships for data-cleaning, such as RelDC. 'Relational' data-cleaning techniques are those, which take into account the dependence among multiple co-reference decisions [Singla and Domingos 2004; McCallum and Wellner 2004]. For example, in a relational approach the analyst can specify a rule that if a pair of attributes co-refer, then another pair of attributes must co-refer as well. In general, relational techniques are different from techniques that analyze interobject relationships.

Scalability. The problem of data-cleaning is treated by the database research community as an applied practical problem. A thorough empirical evaluation of the proposed solution is a must, and the authors are expected to demonstrate that their solution is efficient (i.e., scales) and achieves high cleaning quality for datasets of reasonable size. That is why significant effort, and significant portion of this article, has been devoted to the scalability issues of RELDC. Let us note that various communities study problems closely related to data-cleaning. A number of interesting and involved techniques have been designed to address those problems. However, in contrast to data-cleaning, scalability is not always considered to be an important issue there. As a result, scaling those techniques to even small datasets, found in data-cleaning, is often a nontrivial challenge [Pasula et al. 2002], or the authors do not evaluate their solutions empirically at all [Cohen et al. 2000].

Weaknesses of RELDC. RELDC relies on a ALL-PATHS procedure, which is computationally expensive and the absolute bottleneck of the approach. Consequently, the efficiency of RELDC will strongly depend on how well this procedure is implemented and optimized in a particular DBMS system.

7.4 Other Related Work

Finally, let us conclude this section by summarizing related work, which does not deal with the problem of data-cleaning directly. In Bhalotia et al. [2002], the authors propose a system called BANKS (Browsing ANd Keyword Searching) for keyword-based search on relational databases, together with data and schema browsing. While the heuristic algorithm proposed in that paper does not deal with data-cleaning, it has many similarities with the RELDC approach. For instance, the approach in Bhalotia et al. [2002] models the database as a directed graph. Conceptually, the graph is similar (but different) to the undirected graph employed by RELDC. Like RELDC, the BANKS algorithm also weighs edges for computing the 'proximity through links': a measure related to the connection strength measure, employed by RELDC. Finally, similar to RELDC, BANKS is also a domain-independent approach, the way we defined it above. Another related work is on similarity joins [Cohen and Lewis 1999; Kalashnikov and Prabhakar 2003, 2006], as RELDC essentially performs a similarity join when constructing the choice sets for all references. In addition Cohen and Lewis [1999] covers several other related concepts, such as random walks in graphs.

8. CONCLUSION

In this article, we have shown that analysis of interobject relationships allows to significantly improve the quality of reference disambiguation. We have developed a domain-independent approach, called RELDC, which combines traditional feature-based similarity techniques with techniques that analyze relationships for the purpose of reference disambiguation. To analyze relationships, RELDC views the database as the corresponding entity-relationship graph and then utilizes graph theoretic techniques to analyze paths that exist between nodes in the graph, which corresponds to analyzing chains of relationships between entities. Two models have been developed to analyze the connection strength in the discovered paths. Several optimizations of RELDC have been presented to scale the approach to large datasets. The extensive empirical evaluation of the approach has demonstrated that RELDC improves the quality of reference disambiguation beyond the traditional techniques.

This article demonstrated the usefulness of analyzing relationships for one data-cleaning challenge known as the reference disambiguation or fuzzy lookup. In Chen et al. [2005], we have applied similar techniques to the problem of object consolidation. Our ongoing work [Kalashnikov and Mehrotra 2004] addresses the challenge of automatically adapting the proposed data-cleaning techniques to datasets at hand, by learning how to weigh different connections directly from data, in an automated fashion. Solving this challenge, in general, can not only make the approach a plug-and-play solution, but also can improve both the accuracy and efficiency of the approach, as discussed in Kalashnikov and Mehrotra [2004].

ELECTRONIC APPENDIX

The electronic appendix for this article can be accessed in the ACM Digital Library. The appendix contains the description of the Probabilistic Model (PM) and the alternative WM formulae.

REFERENCES

ANANTHAKRISHNA, R., CHAUDHURI, S., AND GANTI, V. 2002. Eliminating fuzzy duplicates in data warehouses. In *Proceedings of the VLDB Conference*.

BHALOTIA, G., HULGERI, A., MAKHE, C., CHAKRABARTI, S., AND SUDARSHAN, S. 2002. Keyword searching and browsing in databases using BANKS. In *Proceedings of the IEEE ICDE Conference*. IEEE Computer Society Press, Los Alamitos, CA.

BHATTACHARYA, I. AND GETOOR, L. 2004. Iterative record linkage for cleaning and integration. In *Proceedings of the DMKD Workshop*.

BILENKO, M. AND MOONEY, R. 2003. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the ACM SIGKDD Conference*. Washington, DC.

BRIN, S. AND PAGE, L. 1998. The anatomy of a large-scale hypertextual web search engine. In *Proceedings of the International World Wide Web Conference*.

CHAUDHURI, S., GANJAM, K., GANTI, V., KAPOOR, R., NARASAYYA, V., AND VASSILAKIS, T. 2005. Data cleaning in Microsoft SQL Server 2005. In *Proceedings of the ACM SIGMOD Conference*. Baltimore, MD.

CHAUDHURI, S., GANJAM, K., GANTI, V., AND MOTWANI, R. 2003. Robust and efficient fuzzy match for online data-cleaning. In *Proceedings of the ACM SIGMOD Conference*. San Diego, CA.

CHEN, Z., KALASHNIKOV, D. V., AND MEHROTRA, S. 2005. Exploiting relationships for object consolidation. In *Proceedings of the IQIS Workshop at ACM SIGMOD Conference*. Baltimore, MD.

CHENG, R., KALASHNIKOV, D., AND PRABHAKAR, S. 2003a. Evaluating probabilistic queries over imprecise data. In *Proceedings of the ACM SIGMOD Conference*. San Diego, CA.

CHENG, R., KALASHNIKOV, D. V., AND PRABHAKAR, S. 2004. Querying imprecise data in moving object environments. *IEEE TKDE J. 16*, 9 (Sept.).

CHENG, R., PRABHAKAR, S., AND KALASHNIKOV, D. 2003b. Querying imprecise data in moving object environments. In *Proceedings of the IEEE ICDE Conference*. Bangalore, India. IEEE Computer Society Press, Los Alamitos, CA.

Christen, P., Churches, T., and Zhu, J. X. 2002. Probabilistic name and address cleaning and standardization. In *Proceedings of the Australasian Data Mining Workshop*.

CiteSeer 2005. http://citeseer.nj.nec.com/cs.

Cohen, W. W. 1998. Integration of heterogeneous databases without common domains using queries based on textual similarity. In *Proceedings of the ACM SIGMOD Conference*, Seattle, WA.

Cohen, E. and Lewis, D. 1999. Approximating matrix multiplication for pattern recognition tasks. *J. Algorithms 30*, 2, 211–252.

Cohen, W., Kautz, H., and McAllester, D. 2000. Hardening soft information sources. In *Proceedings of the ACM SIGKDD Conference*. Boston, MA.

Cohen, W. W., Ravikumar, P., and Fienberg, S. E. 2003. A comparison of string distance metrics for name-matching tasks. In *Proceedings of the IIWeb Workshop*.

Cohen, W. W. and Richman, J. 2002. Learning to match and cluster large high-dimensional data sets for data integration. In *Proceedings of the ACM SIGKDD Conference*. ACM, New York.

Cormen, T., Leiserson, C., Rivest, and Stein. 2001. *Introduction to algorithms*. MIT Press, Cambridge, MA.

De Raedt, L. et al. 2001. Three companions for data mining in first order logic. In *Proceedings of the Relational Data Mining*, S. Dzeroski and N. Lavrac, Eds. Springer-Verlag, New York.

Dong, X., Halevy, A. Y., and Madhavan, J. 2005. Reference reconciliation in complex information spaces. In *Proceedings of the ACM SIGMOD Conference*. Baltimore, MD.

Faloutsos, C., McCurley, K. S., and Tomkins, A. 2004. Fast discovery of connection subgraphs. In *Proceedings of the ACM SIGKDD Conference*. Seattle, WA.

Fellegi, I. and Sunter, A. 1969. A theory for record linkage. *J. Amer. Stat. Assoc. 64*, 328, 1183–1210.

GAMS solvers 2005. http://www.gams.com/solvers/.

Garcia-Molina, H., Ullman, J. D., and Widom, J. 2002. *Database Systems: The Complete Book*. Prentice-Hall, Englewood Cliffs, NJ.

Getoor, L. 2001. Multi-relational data mining using probabilistic relational models: Research summary. In *Proceedings of the 1st Workshop in Multi-Relational Data Mining*.

Gravano, L., Ipeirotis, P., Jagadish, H., Koudas, N., Muthukrishnan, S., and Srivastava, D. 2001. Approximate string joins in a database (almost) for free. In *Proceedings of the VLDB Conference*.

Hernandez, M. and Stolfo, S. 1995. The merge/purge problem for large databases. In *Proceedings of the ACM SIGMOD Conference*. San Jose, CA.

HomePageSearch 2005. http://hpsearch.uni-trier.de.

Jaro, M. 1989. Advances in record-linkage methodology as applied to matching the 1985 census of Tampa, Florida. *J. Amer. Stat. Assoc. 84*, 406.

Jaro, M. 1995. Probabilistic linkage of large public health data files. *Stat. Med. 14*, 5–7 (Mar.–Apr.).

Jin, L., Li, C., and Mehrotra, S. 2003. Efficient record linkage in large data sets. In *Proceedings of the DASFAA Conference*.

Kalashnikov, D. and Mehrotra. 2003. Exploiting relationships for data cleaning. UCI Tech. Rep. TR-RESCUE-03-02.

Kalashnikov, D. V. and Mehrotra, S. 2004. Learning importance of relationships for reference disambiguation. UCI Tech. Rep. TR-RESCUE-04-23.

Kalashnikov, D. V. and Mehrotra, S. 2005. Exploiting relationships for domain-independent data-cleaning. *SIAM SDM (extended version)*, www.ics.uci.edu/~dvk/pub/sdm05.pdf.

Kalashnikov, D. V., Mehrotra, S., and Chen, Z. 2005. Exploiting relationships for domain-independent data-cleaning. In *Proceedings of the SIAM International Conference on Data Mining (SIAM SDM 2005)* (Newport Beach, CA).

Kalashnikov, D. and Prabhakar, S. 2003. Similarity join for low- and high-dimensional data. In *Proceedings of the DASFAA Conference*.

Kalashnikov, D. V. and Prabhakar, S. 2006. Fast similarity join for multi-dimensional data. *Inf. Syst. J.* to appear.

KDSurvey 2003. http://www.kdnuggets.com/polls/2003/data_preparation.htm.

Lee, M., Hsu, W., and Kothari, V. 2004. Cleaning the spurious links in data. *IEEE Intell. Syst.*

LEE, M., LU, H., LING, T., AND KO. 1999. Cleansing data for mining and warehouse. In *Proceedings of the DEXA*.

LI, X., MORIE, P., AND ROTH, D. 2004. Identification and tracing of ambiguous names: Discriminative and generative approaches. In *Proceedings of the AAAI*.

LITTLE, R. AND RUBIN, D. 1986. *Statistical Analysis with Missing Data*. Wiley, New York.

MALETIC, J. AND MARCUS, A. 2000. Data cleansing: Beyond integrity checking. In *Proceedings of the Conference on Information Quality*.

MALIN, B. 2005. Unsupervised name disambiguation via social network similarity. In *Proceedings of the Workshop on Link Analysis, Counterterrorism, and Security*.

MCCALLUM, A. AND WELLNER, B. 2003. Object consolidation by graph partitioning with a conditionally-trained distance metric. In *Proceedings of the KDD Workshop on Data Cleaning, Record Linkage and Object Consolidation*.

MCCALLUM, A. AND WELLNER, B. 2004. Conditional models of identity uncertainty with application to noun coreference. In *Proceedings of the NIPS*.

MCCALLUM, A. K., NIGAM, K., AND UNGAR, L. 2000. Efficient clustering of high-dimensional data sets with application to reference matching. In *Proceedings of the ACM SIGKDD Conference*. Boston, MA.

MONGE, A. E. AND ELKAN, C. 1996. The field matching problem: Algorithms and applications. In *Proceedings of the ACM SIGKDD Conference*. Portland, OR.

MONGE, A. E. AND ELKAN, C. P. 1997. An efficient domain-independent algorithm for detecting approximately duplicate database records. In *Proceedings of the SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery*. Tucson, AZ.

NEWCOMBE, H., KENNEDY, J., AXFORD, S., AND JAMES, A. 1959. Automatic linkage of vital records. *Science 130*, 954–959.

PASULA, H., MARTHI, B., MILCH, B., RUSSELL, S., AND SHPITSER, I. 2002. Identity uncertainty and citation matching. In *Proceedings of the NIPS Conference*.

RISTAD, E. AND YIANILOS, P. 1998. Learning string edit distance. *IEEE Trans. Patt. Anal. Mach. Intell. 20*, 5 (May), 522–532.

SARAWAGI, S. AND BHAMIDIPATY, A. 2002. Interactive deduplication using active learning. In *Proceedings of the ACM SIGKDD Conference*. Alberta, Canada.

SEID, D. AND MEHROTRA, S. 2006. Complex analytical queries over large attributed graph data. Submitted for Publication.

SHAWE-TAYLOR, J. AND CRISTIANNI, N. 2004. *Kernel Methods for Pattern Analysis*. Cambridge University Press, Cambridge, MA.

SINGLA, P. AND DOMINGOS, P. 2004. Multi-relational record linkage. In *Proceedings of the MRDM Workshop*.

TEJADA, S., KNOBLOCK, C. A., AND MINTON, S. 2002. Learning domain-independent string transformation weights for high accuracy object identification. In *Proceedings of the ACM SIGKDD Conference*. Alberta, Canada.

VERYKIOS, V., MOUSTAKIDES, G. V., AND ELFEKY, M. 2003. A Bayesian decision model for cost optimal record matching. *VLDB J. 12*, 28–40.

WHITE, S. AND SMYTH, P. 2003. Algorithms for estimating relative importance in networks. In *Proceedings of the ACM SIGKDD Conference*. Washington, DC.

WIEDERHOLD, G. 2005. The movies dataset. `www-db.stanford.edu/pub/movies/doc.html`.

WINKLER, W. E. 1994. Advanced methods for record linkage. In *Proceedings of the U.S. Bureau of Census*.

WINKLER, W. 1999. The state of record linkage and current research problems. In *Proceedings of the U.S. Bureau of Census, TR99*.

Received January 2005; revised September 2005 and January 2006; accepted February 2006

Online Appendix to: Domain-Independent Data Cleaning via Analysis of Entity-Relationship Graph

DMITRI V. KALASHNIKOV and SHARAD MEHROTRA
University of California, Irvine

A. PROBABILISTIC MODEL

In the main body of the article, we have presented the weight based model (WM) for computing connection strength. In this section of the appendix, we study a different connection strength model, called the *probabilistic model (PM)*. In the probabilistic model, an edge weight is treated not as "weight" but as "probability" that the edge exists.

A.1 Preliminaries

Notation. We will compute probabilities of certain events. Notation $\mathrm{P}(A)$ refers to the probability of event A to occur. We use $E^{\exists}$ to denote event "E exists" for an edge E. Similarly, we use $E^{\nexists}$ for event "E does not exist". Therefore, $\mathrm{P}(E^{\exists})$ refers to the probability that E exists. We will consider situations where the algorithm computes the probability of following (or, 'going along') a specific edge E, usually in the context of a specific path. This probability is denoted as $\mathrm{P}(E^{\rightarrow})$. We will use $dep(e_1, e_2)$ notation as follows: $dep(e_1, e_2) = \texttt{true}$ if and only if events e_1 and e_2 are dependent. Notation $\mathscr{P}$ denote the path being currently considered. Table V summarizes the notation.

The challenge. Figure 45 illustrates an interesting property of graphs with probabilistic edges: each such graph maps on to a family of regular graphs. Figure 45(a) shows a probabilistic graph where three edges are labeled with probability of 0.5. This probabilistic graph maps on to 2^3 regular graphs. For instance, if we assume that none of the three edges is present (the probability of which is 0.5^3) then the graph in 45(a) will be instantiated to the regular graph in Figure 45(b). Figures 45(c) and 45(d) show other two possible instantiations of it, each having the same probability of occurring of 0.5^3.

The challenge in designing algorithms that compute any measure on such probabilistic graphs, including the connection strength measure, comes from the following observation. If a probabilistic graph has n independent edges, that are labeled with non-1 probabilities, then this graph maps into the exponential number (i.e., 2^n) of regular graphs, where the probability of each instantiation is determined by the probability of the corresponding combination of edges to exist. Algorithms that work with probabilistic graphs should be able to account for the fact that some of the edges exist only with certain probabilities. If such

This work was supported in part by NSF grants 0331707, 0331690, and IRI-9703120.

Table V. Notation

Notation	Meaning
$x^{\exists}$	event "x exists" for (edge,path) x
$x^{\nexists}$	event "x does not exist"
$x^{\rightarrow}$	event that corresponds to following x
$dep(e_1, e_2)$	if events e_1 and e_2 are dependent, then $dep(e_1, e_2) = \texttt{true}$, else $\texttt{false}$
$\mathrm{P}(x^{\exists})$	probability that (edge) x exists
$\mathrm{P}(x^{\rightarrow})$	probability of following (going via) x
$\mathscr{P}$	the path being considered
v_i	i-th node on path $\mathscr{P}$
E_i	(v_i, v_{i+1}) edge on path $\mathscr{P}$
E_{ij}	edge labeled with probability p_{ij}
a_{ij}	$a_{ij} = 1$ if edge E_{ij} exists; else $a_{ij} = 0$
$a_{i0} = 1$	dummy variables: $a_{i0} = 1$ (for all i)
$p_{i0} = 1$	dummy variables: $p_{i0} = 1$ (for all i)
$opt(E)$	if edge E is an option-edge, then $opt(E) = \texttt{true}$, else $opt(E) = \texttt{false}$
$choice[E]$	if edge E is an option-edge, then $choice[E]$ is the choice node associated with E
a (vec)	vector, $\mathbf{a} = (a_{10}, a_{11}, \ldots, a_{(k-1)n_{k-1}})$
a (set)	$\mathbf{a} = \{a_{ij} : \text{ for all } i, j\}$
a (var)	at each moment variable **a** is one instantiation of **a** as a vector

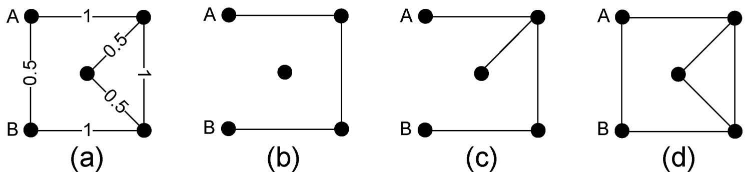

Fig. 45. Probabilistic graph maps to a family of regular graphs.

an algorithm computes a certain measure on a probabilistic graph it should avoid computing it naïvly by computing it on each of 2^n instantiations of this graph separately and then outputting the probabilistic average as the answer. Instead, smart techniques should be designed capable of computing the same answer by applying more efficient methods.

Toy Examples. We will introduce PM by analyzing two examples shown in Figures 46 and 47. Let us consider how to compute the connection strength when edge weights are treated as probabilities that those edges exist. Each figure show a part of a small sample graph with path $\mathscr{P} = A \leftrightarrow B \leftrightarrow C \leftrightarrow D \leftrightarrow E$, which will be of interest to us.

In Figure 46, we assume the events "edge BF is present" and "edge DG is present" are *independent*. The probability of the event "edge BF is present" is 0.8. The probability of the event "edge DG is present" is 0.2. In Figure 47, node F represents a choice node and BF and DF are its option-edges. Events "edge BF exists" and "edge DF exists" are mutually exclusive (and hence strongly

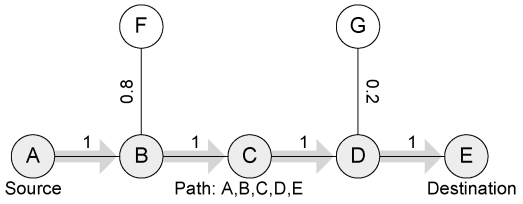

Fig. 46. Toy example: independent case.

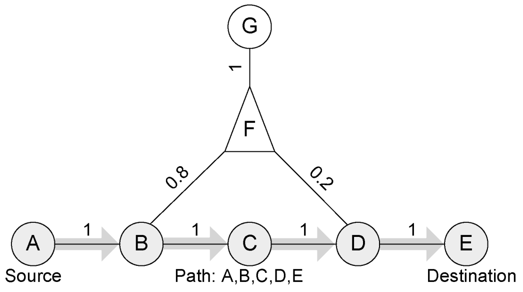

Fig. 47. Toy example: dependent case.

dependent): if one edge is resent, then the other edge must be absent due to the semantics of the choice node.

PM computes the connection strength $c(\mathscr{P})$ of path $\mathscr{P}$ as the probability of following the path $\mathscr{P}$: $c(\mathscr{P}) = \mathrm{P}(\mathscr{P}^{\rightarrow})$. In PM computing $c(\mathscr{P})$ is a two step process. PM first computes the probability $\mathrm{P}(\mathscr{P}^{\exists})$ that path $\mathscr{P}$ exists, then it computes the probability $\mathrm{P}(\mathscr{P}^{\rightarrow}|\mathscr{P}^{\exists})$ of following the path $\mathscr{P}$, given that $\mathscr{P}$ exists. Then PM computes $c(\mathscr{P})$ as $c(\mathscr{P}) = \mathrm{P}(\mathscr{P}^{\rightarrow}) = \mathrm{P}(\mathscr{P}^{\rightarrow}|\mathscr{P}^{\exists})\mathrm{P}(\mathscr{P}^{\exists})$.

Thus, the first step is to compute $\mathrm{P}(\mathscr{P}^{\exists})$. A path exists if each edge on that path exists. For the path $\mathscr{P}$ in Figures 46 and 47, probability $\mathrm{P}(\mathscr{P}^{\exists})$ is equal to $\mathrm{P}(AB^{\exists} \cap BC^{\exists} \cap CD^{\exists} \cap DE^{\exists})$. If the existence of each edge in the path is independent from the existence of other edges, for example, like for the cases shown in Figures 46 and 47, then $\mathrm{P}(\mathscr{P}^{\exists}) = \mathrm{P}(AB^{\exists} \cap BC^{\exists} \cap CD^{\exists} \cap DE^{\exists}) = \mathrm{P}(AB^{\exists})\mathrm{P}(BC^{\exists})\mathrm{P}(CD^{\exists})\mathrm{P}(DE^{\exists}) = 1$.

The second step is to compute the probability $\mathrm{P}(\mathscr{P}^{\rightarrow}|\mathscr{P}^{\exists})$ of following the path $\mathscr{P}$, given that $\mathscr{P}$ exists. Once this probability is computed, we can compute $c(p)$ as $c(\mathscr{P}) = \mathrm{P}(\mathscr{P}^{\rightarrow}) = \mathrm{P}(\mathscr{P}^{\exists})\mathrm{P}(\mathscr{P}^{\rightarrow}|\mathscr{P}^{\exists})$. The probability $\mathrm{P}(\mathscr{P}^{\rightarrow}|\mathscr{P}^{\exists})$ is computed differently for the cases in Figures 46 and 47. This will lead to different values of $c(\mathscr{P})$.

Example A.1.1 (*Independent Edge Existence*). Let us first consider the case where the existence of each edge is independent from the existence of the other edges. In Figure 46, two events "*BF* exists" and "*DG* exists" are independent. The probability of following the path $\mathscr{P}$ is the product of probabilities of following each of the edges on the path: $\mathrm{P}(\mathscr{P}^{\rightarrow}|\mathscr{P}^{\exists}) = \mathrm{P}(AB^{\rightarrow}|\mathscr{P}^{\exists})\mathrm{P}(BC^{\rightarrow}|\mathscr{P}^{\exists})\mathrm{P}(CD^{\rightarrow}|\mathscr{P}^{\exists}) \times \mathrm{P}(DE^{\rightarrow}|\mathscr{P}^{\exists})$. Given path $\mathscr{P}$ exists, the probability of following the edge *AB* in path $\mathscr{P}$ is one. The probability of following the edge *BC* is computed as follows. With probability 0.2 edge *BF* is absent,

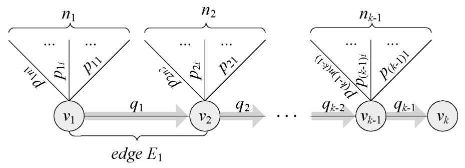

Fig. 48. Independent edge existence. Computing $c(v_1 \leftrightarrow v_2 \leftrightarrow \cdots \leftrightarrow v_k)$. All edges shown in the figure are "possible to follow" edges in the context of the path. Edges that are not possible to follow are not shown.

in which case the probability of following BC is 1. With probability 0.8, edge BF is present, in which case the probability of following BC is $\frac{1}{2}$ – because there are two links, BF and BC, that can be followed. Thus, the total probability of following BC is $0.2 \cdot 1 + 0.8 \cdot \frac{1}{2} = 0.6$. Similarly, the probability of following CD is 1 and the probability of following DE is $0.8 \cdot 1 + 0.2 \cdot \frac{1}{2} = 0.9$. The probability of following the path $\mathscr{P}$, given it exists, is the product of probabilities of following each edge of the path, which is equal to $1 \cdot 0.6 \cdot 1 \cdot 0.9 = 0.54$. Since for the case shown in Figure 46 path $\mathscr{P}$ exists with probability 1, the final probability of following $\mathscr{P}$ is $c(\mathscr{P}) = \mathrm{P}(\mathscr{P}^{\rightarrow}) = 0.54$.

Example A.1.2 (*Dependent Edge Existence*). Let us now consider the case where the existence of an edge can depend on the existence of the other edges. For the case shown in Figure 47 edges BF and DF cannot exist both at the same time. To compute $\mathrm{P}(\mathscr{P}^{\rightarrow}|\mathscr{P}^{\exists})$, we will consider two cases separately: $BF^{\exists}$ and $BF^{\nexists}$. That way we will be able to compute $\mathrm{P}(\mathscr{P}^{\rightarrow}|\mathscr{P}^{\exists})$ as $\mathrm{P}(\mathscr{P}^{\rightarrow}|\mathscr{P}^{\exists}) = \mathrm{P}(BF^{\exists}|p^{\exists})\mathrm{P}(\mathscr{P}^{\rightarrow}|\mathscr{P}^{\exists} \cap BF^{\exists}) + \mathrm{P}(BF^{\nexists}|p^{\exists})\mathrm{P}(\mathscr{P}^{\rightarrow}|\mathscr{P}^{\exists} \cap BF^{\nexists})$.

Let us first assume that $BF^{\exists}$ (i.e., edge BF is present) and then compute $\mathrm{P}(BF^{\exists}|p^{\exists})\mathrm{P}(\mathscr{P}^{\rightarrow}|\mathscr{P}^{\exists} \cap BF^{\exists})$. For the case of Figure 47, if no assumptions about the presence or absence of DF have been made yet, $\mathrm{P}(BF^{\exists}|p^{\exists})$ is simply equal to $\mathrm{P}(BF^{\exists})$, which is equal to 0.8. If BF is present then DF is absent and the probability of following $\mathscr{P}$ is $\mathrm{P}(\mathscr{P}^{\rightarrow}|\mathscr{P}^{\exists} \cap BF^{\exists}) = 1 \cdot \frac{1}{2} \cdot 1 \cdot 1 = \frac{1}{2}$. Now let us consider the second case $BF^{\nexists}$ (and thus $DF^{\exists}$). The probability $\mathrm{P}(BF^{\nexists}|p^{\exists})$ is 0.2. For that case, $\mathrm{P}(\mathscr{P}^{\rightarrow}|\mathscr{P}^{\exists} \cap BF^{\nexists})$ is equal to $1 \cdot 1 \cdot 1 \cdot \frac{1}{2} = \frac{1}{2}$. Thus, $\mathrm{P}(\mathscr{P}^{\rightarrow}|\mathscr{P}^{\exists}) = 0.8 \cdot \frac{1}{2} + 0.2 \cdot \frac{1}{2} = 0.5$. Therefore, $c(\mathscr{P}) = \mathrm{P}(\mathscr{P}^{\rightarrow}) = 0.50$, which is different from that of the previous experiment.

A.2 Independent Edge Existence

Let us consider how to compute path connection strength in general case, assuming the existence of each edge is independent from existence of the other edges.

A.2.1 *General Formulae.* In general, any path $\mathscr{P}$ can be represented as a sequence of k nodes $\langle v_1, v_2, \ldots, v_k \rangle$ or as a sequence of $(k-1)$ edges $\langle E_1, E_2, \ldots, E_{(k-1)} \rangle$, as illustrated in Figure 48, where $E_i = (v_i, v_{i+1})$ and $\mathrm{P}(E_i^{\exists}) = q_i$, for $i = 1, 2, \ldots, k-1$. We will refer to the edges labeled with probabilities p_{ij} (for all i, j) in this figure as E_{ij}. The goal is to compute the probability of following the path $\mathscr{P}$, which is the measure of the connection strength of the path $\mathscr{P}$:

$$c(\mathscr{P}) = \mathrm{P}(\mathscr{P}^{\rightarrow}) = \mathrm{P}(\mathscr{P}^{\exists})\mathrm{P}(\mathscr{P}^{\rightarrow}|\mathscr{P}^{\exists}). \tag{9}$$

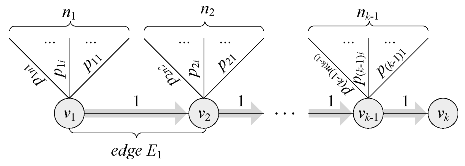

Fig. 49. The case in this figure is similar to that of Figure 48 with an additional assumption that path $\mathscr{P}$ exists.

The probability that $\mathscr{P}$ exists is equivalent to the probability that each of its edges exists:

$$\mathrm{P}(\mathscr{P}^{\exists}) = \mathrm{P}\left(\bigcap_{i=1}^{k-1} E_i^{\exists}\right). \tag{10}$$

Given our assumption of the independence, $\mathrm{P}(\mathscr{P}^{\exists})$ can be computed as

$$\mathrm{P}(\mathscr{P}^{\exists}) = \prod_{i=1}^{k-1} \mathrm{P}(E_i^{\exists}) = \prod_{i=1}^{k-1} q_i. \tag{11}$$

To compute $\mathrm{P}(\mathscr{P}^{\rightarrow})$, we now need to compute $\mathrm{P}(\mathscr{P}^{\rightarrow}|\mathscr{P}^{\exists})$. In turn, to compute $\mathrm{P}(\mathscr{P}^{\rightarrow}|\mathscr{P}^{\exists})$, let us analyze how labels p_{ij} and q_i (for all i, j) in Figure 48 will change, if we assume that $\mathscr{P}$ exists. We will compute the corresponding new labels, $\widetilde{p}_{ij}$ and $\widetilde{q}_i$, and reflect the changes in Figure 49. Since q_i is defined as $q_i = \mathrm{P}(E_i^{\exists})$ and p_{ij} is defined as $p_{ij} = \mathrm{P}(E_{ij}^{\exists})$, the new labels are computed as $\widetilde{q}_i = \mathrm{P}(E_i^{\exists}|p^{\exists}) = 1$ and $\widetilde{p}_{ij} = \mathrm{P}(E_{ij}^{\exists}|p^{\exists})$. Given our assumption of independence, $\widetilde{p}_{ij} = p_{ij}$. The new labeling is shown in Figure 49.

Let us define a variable a_{ij} for each edge E_{ij} (labeled p_{ij}) as follows: $a_{ij} = 1$ if and only if edge E_{ij} exists; otherwise $a_{ij} = 0$. Also, for notational convenience, let us define two sets of dummy variables, a_{i0} and p_{i0}, such that $a_{i0} = 1$ and $p_{i0} = 1$, for $i = 1, 2, \ldots, k-1$.[14] Let $\mathbf{a}$ denote a vector consisting of all a_{ij}'s: $\mathbf{a} = (a_{10}, a_{11}, \ldots, a_{(k-1)n_{k-1}})$. Let $\mathcal{A}$ denote the set of all possible instantiations of $\mathbf{a}$, i.e. $|\mathcal{A}| = 2^{n_1+n_2+\cdots+n_{k-1}}$. Then, probability $\mathrm{P}(\mathscr{P}^{\rightarrow}|\mathscr{P}^{\exists})$ can be computed as

$$\mathrm{P}(\mathscr{P}^{\rightarrow}|\mathscr{P}^{\exists}) = \sum_{\mathbf{a}\in\mathcal{A}} \{\mathrm{P}(\mathscr{P}^{\rightarrow}|\mathbf{a}\cap\mathscr{P}^{\exists})\mathrm{P}(\mathbf{a}|\mathscr{P}^{\exists})\}, \tag{12}$$

where $\mathrm{P}(\mathbf{a}|\mathscr{P}^{\exists})$ is the probability of instantiation $\mathbf{a}$ to occur while assuming $\mathscr{P}^{\exists}$. Given our assumption of independence of probabilities, $\mathrm{P}(\mathbf{a}|\mathscr{P}^{\exists}) = \mathrm{P}(\mathbf{a})$. Probability $\mathrm{P}(\mathbf{a})$ can be computed as

$$\mathrm{P}(\mathbf{a}|\mathscr{P}^{\exists}) = \mathrm{P}(\mathbf{a}) = \prod_{\substack{i=1,2,\ldots,k-1\\ j=0,1,\ldots,n_i}} p_{ij}^{a_{ij}}(1-p_{ij})^{1-a_{ij}}. \tag{13}$$

Probability $\mathrm{P}(\mathscr{P}^{\rightarrow}|\mathbf{a}\cap\mathscr{P}^{\exists})$, which is the probability to go via $\mathscr{P}$ given (1) a

[14]Intuitively (1) $a_{i0} = 1$ corresponds to the fact that edge E_i exists given path $\mathscr{P}$ exists; and (2) $p_{i0} = 1$ corresponds to $p_{i0} = \mathrm{P}(E_i^{\exists}|p^{\exists}) = 1$.

particular instantiation of **a**; and (2) the fact that $\mathscr{P}$ exists, can be computed as

$$P(\mathscr{P}^{\rightarrow}|\mathbf{a} \cap \mathscr{P}^{\exists}) = \prod_{i=1}^{k-1} \frac{1}{1 + \sum_{j=1}^{n_i} a_{ij}} \equiv \prod_{i=1}^{k-1} \frac{1}{\sum_{j=0}^{n_i} a_{ij}}. \tag{14}$$

Thus,

$$P(\mathscr{P}^{\rightarrow}) = \left(\prod_{i=1}^{k-1} q_i\right) \left(\sum_{\mathbf{a}\in\mathcal{A}} \left\{ \left[\prod_{i=1}^{k-1} \frac{1}{\sum_{j=0}^{n_i} a_{ij}}\right] \left[\prod_{ij} p_{ij}^{a_{ij}} (1 - p_{ij})^{1-a_{ij}}\right] \right\}\right). \tag{15}$$

A.2.2 *Computing Path Connection Strength in Practice.* Notice, Equation (15) iterates through all possible instantiations of **a**, which is impossible to compute in practice, given $|\mathcal{A}| = 2^{n_1+n_2+\cdots+n_{k-1}}$. This equation must be simplified to make the computation feasible.

Computing $P(\mathscr{P}^{\rightarrow}|\mathscr{P}^{\exists})$ *as* $\prod_{i=1}^{k-1} P(E_i^{\rightarrow}|\mathscr{P}^{\exists})$. To achieve the simplification, we will use our assumption of independence of probabilities, which allows us to compute $P(\mathscr{P}^{\rightarrow}|\mathscr{P}^{\exists})$ as the product of the probabilities of following each individual edge in the path:

$$P(\mathscr{P}^{\rightarrow}|\mathscr{P}^{\exists}) = \prod_{i=1}^{k-1} P(E_i^{\rightarrow}|\mathscr{P}^{\exists}). \tag{16}$$

Let $\mathbf{a}_i$ denote vector $(a_{i0}, a_{i1}, \ldots, a_{in_i})$, that is $\mathbf{a} = (\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_{k-1})$. Let $\mathcal{A}_i$ denote all possible instantiations of $\mathbf{a}_i$. That is, $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2 \times \cdots \times \mathcal{A}_{k-1}$ and $|\mathcal{A}_i| = 2^{n_i}$. Then

$$P(E_i^{\rightarrow}|\mathscr{P}^{\exists}) = \sum_{\mathbf{a}_i\in\mathcal{A}_i} \left\{ \left[\frac{1}{\sum_{j=0}^{n_i} a_{ij}}\right] \left[\prod_{j=0}^{n_i} p_{ij}^{a_{ij}} (1 - p_{ij})^{1-a_{ij}}\right] \right\}. \tag{17}$$

Combining Equations (9), (16), and (17), we have

$$P(\mathscr{P}^{\rightarrow}) = \left(\prod_{i=1}^{k-1} q_i\right) \prod_{i=1}^{k-1} \left(\sum_{\mathbf{a}_i\in\mathcal{A}_i} \left\{ \left[\frac{1}{\sum_{j=0}^{n_i} a_{ij}}\right] \left[\prod_{j=0}^{n_i} p_{ij}^{a_{ij}} (1 - p_{ij})^{1-a_{ij}}\right] \right\}\right). \tag{18}$$

The Effect of Transformation. Notice, using Eq. (15), the algorithm will need to perform $|\mathcal{A}| = 2^{n_1+n_2+\cdots+n_{k-1}}$ iterations – one per each instantiation of **a**. Using Eq. (18), the algorithm will need to perform $|\mathcal{A}_1|+|\mathcal{A}_2|+\cdots+|\mathcal{A}_{k-1}| = 2^{n_1}+2^{n_2}+\cdots+2^{n_{k-1}}$ iterations. Furthermore, each iteration requires less computation. These factors lead to a significant improvement.

Handling Weight-1 Edges. The formula in Eq. (17) assumes 2^{n_i} iterations will be needed to compute $P(E_i^{\rightarrow}|\mathscr{P}^{\exists})$. This formula can be modified further to achieve more efficient computation as follows. In practice, some of the p_{ij}'s, or even all of them, are often equal to 1. Figure 50 shows the case where m $(0 \le m \le n_i)$ edges incident to node v_i are labeled with 1. Let $\widetilde{\mathbf{a}}_i$ denote vector $(a_{i0}, a_{i1}, \ldots, a_{i(n_i-m)})$ and let $\widetilde{\mathcal{A}}_i$ be the set of all possible instantiations of this vector. Then, Eq. (17) can be simplified to

$$P(E_i^{\rightarrow}|\mathscr{P}^{\exists}) = \sum_{\widetilde{\mathbf{a}}_i\in\widetilde{\mathcal{A}}_i} \left\{ \left[\frac{1}{m + \sum_{j=0}^{n_i-m} a_{ij}}\right] \left[\prod_{j=0}^{n_i-m} p_{ij}^{a_{ij}} (1 - p_{ij})^{1-a_{ij}}\right] \right\}. \tag{19}$$

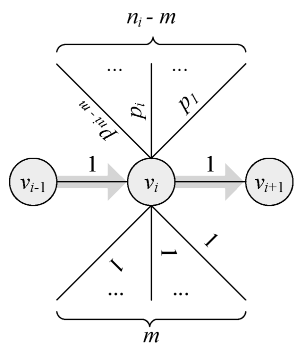

Fig. 50. Probability of following the edge $E_i = (v_i, v_{i+1})$.

The number of iteration is reduced from 2^{n_i} to $2^{n_i - m}$.

Computing $\mathrm{P}(E_i^{\rightarrow} | \mathscr{P}^{\exists})$ *as* $\sum_{\ell=0}^{n_i} \frac{1}{1+\ell} \mathrm{P}(s_i = \ell)$. Performing $2^{n_i - m}$ iterations can still be expensive for the cases when $(n_i - m)$ is large. Next, we discuss several methods to deal with this issue.

Method 1. *Do Not Simplify Further.* In general, the value of $2^{n_i - m}$ can be large. However, for a particular instance of a cleaning problem, it can be that (a) $2^{n_i - m}$ is never large or (b) $2^{n_i - m}$ can be large but bearable and the cases when it is large are infrequent. In those cases, further simplification might not be required.

Method 2. *Estimate Answer Using Results from Poisson Trials Theory.* Let us denote the following sum as s_i: $s_i = \sum_{j=1}^{n_i} a_{ij}$. From a basic probability course, we know that the binomial distribution gives the number of successes in n independent trials where each trial is successful with the same probability p. The binomial distribution can be viewed as a sum of several *i.i.d.* Bernoulli trials. The *Poisson trials* process is similar to the binomial distribution process where trials are still independent but not necessarily identically distributed, that is, the probability of success in the ith trial is p_i. We can modify Eq. (18) to compute $\mathrm{P}(E_i^{\rightarrow} | \mathscr{P}^{\exists})$ as follows:

$$\mathrm{P}(E_i^{\rightarrow} | \mathscr{P}^{\exists}) = \sum_{\ell=0}^{n_i} \frac{1}{1+\ell} \mathrm{P}(s_i = \ell). \tag{20}$$

Notice, for a given i we can treat $a_{i1}, a_{i2}, \ldots, a_{in_i}$ as a sequence of n_i Bernoulli trials with probabilities of success $\mathrm{P}(a_{ij} = 1) = p_{ij}$. One would want to *estimate* $\mathrm{P}(s_i = \ell)$ *quickly*, rather than compute it exactly via iterating over all cases when $(s_i = \ell)$. That is, we would like to avoid computing $\mathrm{P}(s_i = \ell)$ as

$$\mathrm{P}(s_i = \ell) = \sum_{\substack{\mathbf{a}_i \in \mathcal{A}_i \\ s_i = \ell}} \prod_{j=0}^{n_i} p_{ij}^{a_{ij}} (1 - p_{ij})^{1 - a_{ij}}.$$

There are multiple cases when $\mathrm{P}(s_i = \ell)$ can be computed quickly. For example, in certain cases it can be possible to utilize the Poisson trials theory to estimate $\mathrm{P}(s_i = \ell)$. For instance, if each p_{ij} is small, then from the probability theory we know that

$$P(s_i = \ell) = \frac{\lambda^\ell e^{-\lambda}}{\ell!}\left\{1 + O\left(\lambda \max_{j=1,2,\ldots,n_i} p_{ij} + \frac{\ell^2}{\lambda}\max_{j=1,2,\ldots,n_i} p_{ij}\right)\right\}, \text{ where } \lambda = \sum_{j=1}^{n_i} p_{ij}. \quad (21)$$

One can also utilize the following "Monte-Carlo like" method to compute $P(s_i = \ell)$. The idea is to have several runs. During run number m, the method decides by generating a random number ("tossing a coin") if edge E_{ij} is present (variable a_j will be assigned 1) or absent ($a_j = 0$) for this run based on the probability p_{ij}. Then, the sum $S_m = \sum_{j=1}^{n_i} p_{ij}$ is computed for that run. After n runs, the desired probability $P(s_i = \ell)$ is estimated as the number of S_i's which are equal to ℓ, divided by n.

Method 3. *Use Linear Cost Formula.* The third approach is to use a cut-off threshold to decide if the cost of performing $2^{n_i - m}$ iterations is acceptable. If it is acceptable, then compute $P(E_i^{\rightarrow} | \mathscr{P}^{\exists})$ precisely, using iterations. If it is not acceptable (typically, rare case), then try to use Eq. (21). If that fails, use the following (linear-cost) approximation formula. First, compute the expected number of edges μ_i among n_i edges $E_{i1}, E_{i2}, \ldots, E_{in_i}$, where $P(E_{ij}^{\exists}) = p_{ij}$, as follows: $\mu_i = m + \sum_{j=1}^{n_i - m} p_{ij}$. Then, since there are $1 + \mu_i$ possible links that can be followed on average, the probability of following E_i can be coarsely estimated as

$$P(E_i^{\rightarrow} | \mathscr{P}^{\exists}) \approx \frac{1}{1 + \mu_i} = \frac{1}{m + \sum_{j=0}^{n_i - m} p_{ij}}. \quad (22)$$

A.3 Dependent Edge Existence

In this section, we discuss how to compute connection strength if occurrence of edges is dependent. In our model, dependence between two edges arises only when those two edges are option-edges of the same choice node. We next show how to compute $P(\mathscr{P}^{\rightarrow})$ for those cases.

We need to address two principal situations. The first is to handle all choice nodes on the path. The second step is to handle all choice nodes such that a choice node itself is not on the path but at least two of its option nodes are on the path. Next, we address those two cases.

A.3.1 *Choice Nodes on the Path.*

The first case of how to deal with choice nodes on the path is a simple one. There are two subcases in this case illustrated in Figures 51 and 53.

Figure 51 shows a choice node C on the path which has options D, G, and F. Recall, we compute $P(\mathscr{P}^{\rightarrow}) = P(\mathscr{P}^{\exists})P(\mathscr{P}^{\rightarrow} | \mathscr{P}^{\exists})$. When we compute $P(\mathscr{P}^{\exists})$ each edge of path $\mathscr{P}$ should exist. Thus, edge CD must exist, which means edges CG and CF do not exist. Notice, this case is equivalent to the case shown in Figure 52 where (a) edges CG and CF are not there (permanently eliminated from consideration); and (b) node C is just a regular (not a choice) node connected to D via an edge (in this case the edge is labeled 0.2). If we now consider this equivalent case, then we can simply apply Eq. (18) to compute the connection strength.

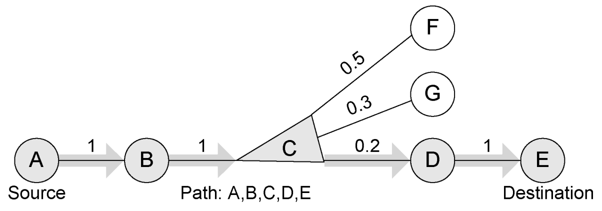

Fig. 51. Choice node on the path.

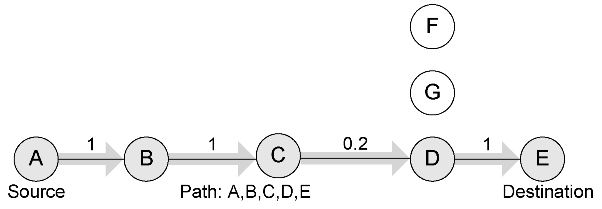

Fig. 52. Choice node on the path: Removing choice.

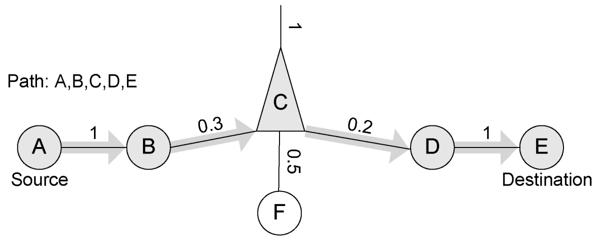

Fig. 53. Choice node on the path: Illegal path.

In general, all choice nodes on the path, can be "eliminated" from the path one by one (or, rather, "replaced with regular nodes") using the procedure above.

Figure 53 shows a choice node C on the path which have options B, F, and D, such that $B \leftrightarrow C \leftrightarrow D$ is a part of the path $\mathscr{P}$. Semantically, edges CB, CF, and CD are mutually exclusive, so path $\mathscr{P}$ cannot exist. Such paths are said to be *illegal* and they are ignored by the algorithm.

A.3.2 *Options of the Same Choice Node on the Path.* Assume now that we have applied the procedure from Section and all choice nodes are "eliminated" from path $\mathscr{P}$. At this point the probability $\mathrm{P}(\mathscr{P}^{\exists})$ can be computed as $\mathrm{P}(\mathscr{P}^{\exists}) = \prod_{i=1}^{k-1} q_i$. The only case that is left to be considered is where a choice node itself is not on the path but at least two of its options are on the path. An example of such a case is illustrated in Figure 54 where choice node F has four options: G, B, D, and H, two of which B and D belong to the path being considered. After choice nodes are eliminated from the path, the goal becomes to create a formula similar to Eq. (18), but for the general "dependent" case.

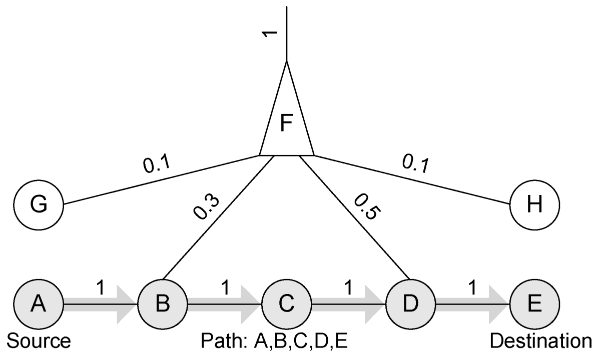

Fig. 54. Options of the same choice node on the path.

Let us define two sets, **f** and **d**, of 'free' and 'dependent' a_{ij}'s as:

$$\begin{aligned} \mathbf{f} &= \{a_{ij} : \forall r, s\ (r \neq i \text{ or } s \neq j) \Rightarrow dep(E^{\exists}_{ij}, E^{\exists}_{rs}) = \texttt{false}\}, \\ \mathbf{d} &= \{a_{ij} : \exists r, s\ (r \neq i \text{ or } s \neq j) : dep(E^{\exists}_{ij}, E^{\exists}_{rs}) = \texttt{true}\}. \end{aligned} \tag{23}$$

Notice, $\mathbf{a} = \mathbf{f} \cup \mathbf{d}$ and $\mathbf{f} \cap \mathbf{d} = \emptyset$. If $\mathbf{d} = \emptyset$, then there is no dependence and the solution is given by Eq. (18). To handle the case where $\mathbf{d} \neq \emptyset$, let us define $\mathbf{f}_i$ and $\mathbf{d}_i$ as:

$$\begin{aligned} \mathbf{f}_i &= \{a_{ij} : a_{ij} \in \mathbf{f}, j = 0, 1, \ldots, n_i\}, \\ \mathbf{d}_i &= \{a_{ij} : a_{ij} \in \mathbf{d}, j = 1, 2, \ldots, n_i\}. \end{aligned} \tag{24}$$

Notice, $\mathbf{a}_i = \mathbf{f}_i \cup \mathbf{d}_i$ and $\mathbf{f}_i \cap \mathbf{d}_i = \emptyset$. Let us define $\mathcal{D}$ as the set of all possible instantiations of **d**, and $\mathcal{F}_i$ as the set of all possible instantiations of $\mathbf{f}_i$. Then

$$\mathrm{P}(\mathscr{P}^{\rightarrow}) = \underbrace{\left(\prod_{i=1}^{k-1} q_i\right)}_{\mathrm{P}(\mathscr{P}^{\exists})} \sum_{\mathbf{d} \in \mathcal{D}} \left\{ \underbrace{\left[\prod_{i=1}^{k-1} \left(\sum_{\mathbf{f}_i \in \mathcal{F}_i} \left\{ \left[\frac{1}{\sum_{j=0}^{n_i} a_{ij}}\right] \left[\prod_{j : a_{ij} \in \mathbf{f}_i} p_{ij}^{a_{ij}} (1 - p_{ij})^{1 - a_{ij}}\right] \right\} \right)\right]}_{\Psi(\mathbf{d})} \mathrm{P}(\mathbf{d}) \right\}. \tag{25}$$

Eq. (25) iterates over all feasible instantiations of **d**. P(**d**) is the probability of a specific instance of **d** to occur. Eq. (25) contains term $\sum_{\mathbf{d} \in \mathcal{D}} \{\Psi(\mathbf{d})\mathrm{P}(\mathbf{d})\}$. What this achieves is that a particular instantiation of **d** "fixates" a particular combination of all "dependent" edges, and P(**d**) corresponds to the probability of that combination. Notice, $\Psi(\mathbf{d})$ directly corresponds to $\mathrm{P}(\mathscr{P}^{\rightarrow} | \mathscr{P}^{\exists})$ part of Eq. (18). To compute $\mathrm{P}(\mathscr{P}^{\rightarrow})$ in Eq. (25), we only need to specify how to compute P(**d**).

Computing P(**d**). Recall, we now consider the cases where a_{ij} is in **d** only because there is (at least one) another $a_{rs} \in \mathbf{d}$ such that $dep(E^{\exists}_{ij}, E^{\exists}_{rs}) = \texttt{true}$ and $choice[E_{ij}] = choice[E_{rs}]$, where $choice[E_{ij}]$ is the choice node associated with E_{ij}. Figure 47 illustrates an example of such a case. Therefore, for each $a_{ij} \in \mathbf{d}$, we can identify choice node $r_\ell = choice[E_{ij}]$ and compute set $C_\ell = \{a_{rs} \in \mathbf{d} : choice[E_{rs}] = r_\ell\}$. Then, for any two distinct elements $a_{ij} \in C_\ell$ and a_{rs} the following holds: $dep(E^{\exists}_{ij}, E^{\exists}_{rs}) = \texttt{true}$ if and only if $a_{rs} \in C_\ell$.

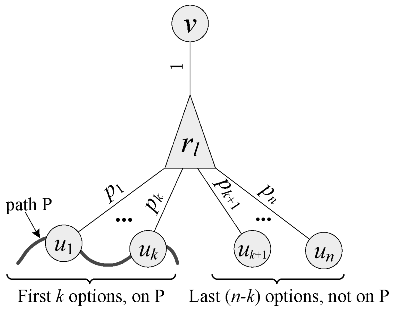

Fig. 55. Intra choice dependence.

In other words, we can split set $\mathbf{d}$ into nonintersecting subsets $\mathbf{d} = C_1 \cup C_2 \cup \cdots \cup C_m$. The existence of each edge E_{ij} such that a_{ij} is in one of those sets C_ℓ depends only on the existence of those edges E_{rs}'s whose a_{rs} is in C_ℓ as well. Therefore, $\mathrm{P}(\mathbf{d})$ can be computed as $\mathrm{P}(\mathbf{d}) = \mathrm{P}(\mathbf{d}_{C_1})\mathrm{P}(\mathbf{d}_{C_2}) \times \cdots \times \mathrm{P}(\mathbf{d}_{C_m})$, where $\mathbf{d}_{C_\ell}$ is a particular instantiation of a_{ij}'s from C_ℓ. Now, to be able to compute Eq. (25), we only need to specify how to compute $P(\mathbf{d}_{C_\ell})$ for $\ell = 1, 2, \ldots, m$.

Computing $P(\mathbf{d}_{C_\ell})$. Figure 55 shows choice node r_ℓ with n options $u_1, u_2, \ldots, u_n$. Each edge (r_ℓ, u_j) for $j = 1, 2, \ldots, n$ is labeled with probability p_j.

As before, to specify which edge is present and which is absent, each option edge has variable a_j associated with it. Variable $a_j = 1$ if and only if the edge labeled with p_j is present, otherwise $a_j = 0$. That is, $\mathrm{P}(a_j = 1) = p_j$ and $p_1 + p_2 + \cdots + p_n = 1$.

Let us assume, without loss of generality, that the first k $(2 \le k \le n)$ options $u_1, u_2, \ldots, u_k$ of r_ℓ belong to path $\mathscr{P}$ while the other $(n - k)$ options $u_{k+1}, u_{k+2}, \ldots, u_n$ do not belong to $\mathscr{P}$, as shown in Figure 55. In the context of Figure 55, computing $P(\mathbf{d}_{C_\ell})$ is equivalent to computing the probability a particular instantiation of vector $(a_1, a_2, \ldots, a_k)$ to occur.

Notice, only one a_i among $a_1, a_2, \ldots, a_k, a_{k+1}, a_{k+2}, \ldots, a_n$ can be 1, the rest are zeroes. First, let us compute the probability of instantiation $a_1 = a_2 = \cdots = a_k = 0$. For that case, one of $a_{k+1}, a_{k+2}, \ldots, a_n$ should be equal to 1. Thus, $\mathrm{P}(a_1 = a_2 = \cdots = a_k = 0) = p_{k+1} + p_{k+2} + \cdots + p_n$.

The second case is when one of $a_1, a_2, \ldots, a_k$ is 1. Assume that $a_j = 1$, where $1 \le j \le k$, then $\mathrm{P}(a_j = 1) = p_j$. To summarize:

$$\mathrm{P}(a_1, a_2, \ldots, a_k) = \begin{cases} p_j & \text{if } \exists j\ (1 \le j \le k) : a_j = 1; \\ p_{k+1} + p_{k+2} + \cdots + p_n & \text{otherwise.} \end{cases}$$

Now we know how to compute $P(\mathbf{d}_{C_\ell})$ for $\ell = 1, 2, \ldots, m$, thus we can compute $\mathrm{P}(\mathbf{d})$. Therefore, we have specified how to compute path connection strength using Eq. (25).

A.4 Computing the Total Connection Strength

The connection strength between nodes u and v is computed as a sum of connection strengths of all simple paths between u and v: $c(u, v) = \sum_{\mathscr{P} \in \mathcal{P}_L(u,v)} c(\mathscr{P})$.

Based on this connection strength, the weight of the corresponding edge will be determined.

Let us give the motivation of why the *summation* of individual simple paths is performed. We associate the connection strength between two nodes u and v with probability of reaching v from u via only L-short simple paths. Let us name those simple paths $\mathscr{P}_1, \mathscr{P}_2, \dots, \mathscr{P}_k$. Let us call $\mathcal{G}(u, v)$ the subgraph comprised of the union of those paths: $\mathcal{G}(u, v) = \mathscr{P}_1 \cup \mathscr{P}_2 \cup \cdots \cup \mathscr{P}_k$. Subgraph $\mathcal{G}(u, v)$ is a subgraph of the complete graph $G = (V, E)$, where V is the set of vertices $V = \{v_1, v_2, \dots, v_{|V|}\}$ and E is the set of edges $E = \{E_1, E_2, \dots, E_{|E|}\}$. Let us define a_i as: $a_i = 1$ if and only if edge E_i is present, otherwise $a_i = 0$. Let $\mathbf{a}$ denote vector $(a_1, a_2, \dots, a_{|E|})$ and let $\mathcal{A}$ be the set of all possible instantiations of $\mathbf{a}$.

We need to compute the probability of reaching v from u via subgraph $\mathrm{P}(\mathcal{G}(u, v)^{\rightarrow})$, which we treat as the measure of the connection strength. We can represent $\mathrm{P}(\mathcal{G}(u, v)^{\rightarrow})$ as

$$\mathrm{P}(\mathcal{G}(u, v)^{\rightarrow}) = \sum_{\mathbf{a} \in \mathcal{A}} \mathrm{P}(\mathcal{G}(u, v)^{\rightarrow} | \mathbf{a}) \mathrm{P}(\mathbf{a}). \tag{26}$$

Notice, when computing $\mathrm{P}(\mathcal{G}(u, v)^{\rightarrow} | \mathbf{a})$ we assume a particular instantiation of $\mathbf{a}$. Therefore, the complete knowledge of which edges are present and which are absent is available, as if all the edges were "fixed". Assuming one particular instantiation of $\mathbf{a}$, there is no dependence among edge existence events any longer: each edge is either present with 100% probability or absent with 100% probability. Thus,

$$\mathrm{P}(\mathcal{G}(u, v)^{\rightarrow} | \mathbf{a}) = \sum_{i=1}^{k} \mathrm{P}(\mathscr{P}_i^{\rightarrow} | \mathbf{a}), \tag{27}$$

and

$$\begin{aligned} \mathrm{P}(\mathcal{G}(u, v)^{\rightarrow}) &= \sum_{\mathbf{a} \in \mathcal{A}} \mathrm{P}(\mathcal{G}(u, v)^{\rightarrow} | \mathbf{a}) \mathrm{P}(\mathbf{a}) \\ &= \sum_{\mathbf{a} \in \mathcal{A}} \left[\left(\sum_{i=1}^{k} \mathrm{P}(\mathscr{P}_i^{\rightarrow} | \mathbf{a}) \right) \mathrm{P}(\mathbf{a}) \right] \\ &= \sum_{i=1}^{k} \left[\sum_{\mathbf{a} \in \mathcal{A}} (\mathrm{P}(\mathscr{P}_i^{\rightarrow} | \mathbf{a}) \mathrm{P}(\mathbf{a})) \right] \\ &= \sum_{i=1}^{k} \mathrm{P}(\mathscr{P}_i^{\rightarrow}). \end{aligned} \tag{28}$$

Equation (28) shows that the total connection strength is the sum of the connection strength of all L-short simple paths.

B. ALTERNATIVE WM FORMULAE

One could argue that the original WM formulae, covered in the main body of this article, does not address properly the situation illustrated in Figure 56. In the example in Figure 56, when disambiguating references r the option set for this reference S_r has three elements y_1, y_2, and y_3. In Figure 56(a), the

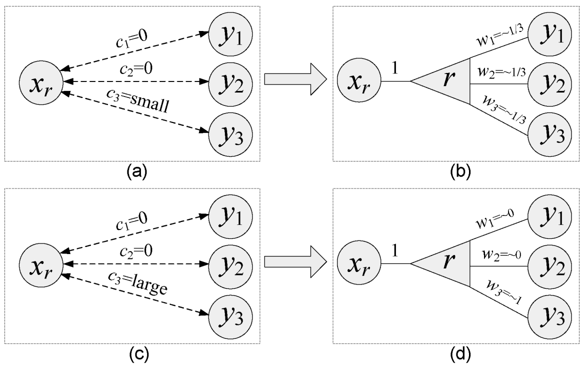

Fig. 56. Motivation for *Normalization method 2*.

connection strengths $c_j = c(x_r, y_j)$ for $j = 1, 2, 3$ are as follows: $c_1 = 0$, $c_2 = 0$, and c_3 is a nonnegative value which is small. That is, RelDC has not been able to find any evidence that r^* is y_1 or y_2 and found insubstantial evidence that r^* is y_3. However, the original WM formulae will compute $w_1 = 0$, $w_2 = 0$, and $w_3 = 1$, one interpretation of which might be that the algorithm is 100% confident y_3 is r^*.

One can argue that in such a situation, since the evidence that r^* is y_3 is very weak, w_1, w_2, and w_3 should be roughly equal. That is, their values should be close to $\frac{1}{3}$ in this case, as shown in Figure 56(b), and w_3 should be slightly greater than w_1 and w_2.

Figure 56(c) is similar to Figure 56(a), except for c_3 is large with respect to other connection strengths in the system. Following the same logic, weights w_1 and w_2 should be close to zero. Weight w_3 should be close to 1, as in Figure 56(d).

We can correct those issues with the WM formulae and achieve the desired weight assignment as follows. We will assume that since y_1, y_2, and y_3 are in the option set S_r of reference r (whereas other entities are not in the option set), in such situations there is always a very small default connection strength α between each x_r and y_j. That is, the weights should be assigned as follows:

$$w_j = \frac{(c_j + \alpha)}{\sum_{\ell=1}^{N}(c_{r\ell} + \alpha)}. \tag{29}$$

where α is a small positive weight. Equation (29) corrects the mentioned drawbacks of the WM formulae.