Accepted to **SIAM International Conference on Data Mining (SIAM SDM) 2005**

Exploiting relationships for domain-independent data cleaning*

Dmitri V. Kalashnikov Sharad Mehrotra

Computer Science Department
University of California, Irvine
http://www.ics.uci.edu/~dvk/RelDC

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

Additional Key Words and Phrases: relationship-based data cleaning, record linkage, data mining

Contents

1 Introduction **3**

2 Motivation for analyzing relationships **4**

3 Notation and problem definition **6**
- 3.1 References . . . 7
- 3.2 The entity-relationship graph . . . 7
- 3.3 The objective of reference disambiguation . . . 8
- 3.4 Connection Strength and Context Attraction Principle . . . 9

4 The RelDC approach **9**
- 4.1 Computing connection strength . . . 9
  - 4.1.1 First phase of WM: discovering connections . . . 11
  - 4.1.2 Second phase of WM: measuring connection strength . . . 11
- 4.2 Determining equations for option-edge weights . . . 13

*Portions of this work are supported by the NSF under Grants 0331707 and 0331690.

4.3 Determining all weights by solving equations. 13
4.4 Resolving references by interpreting weights. 15

5 Implementations of RelDC **15**
5.1 Iterative and solver implementations of RelDC 15
5.1.1 Solver 15
5.1.2 Iterative 15
5.1.3 Bottleneck of RelDC 16
5.2 Constraining the problem 17
5.3 Depth-first and greedy implementation of AllPaths. 17
5.3.1 Depth-first and greedy algorithms 17
5.3.2 Paths storage 18
5.3.3 Comparing complexity of greedy and depth-first implementations 19
5.4 NBH optimization: utilizing neighborhoods for path pruning. 19
5.5 Storing discovered paths explicitly. 21
5.6 Compatibility of implementations 21
5.7 Computational complexity of RelDC. 21

6 Experimental Results **22**
6.1 Case Study 1: the publications dataset 22
6.1.1 Datasets 22
6.1.2 Accuracy experiments 25
6.1.3 Other experiments 27
6.2 Case Study 2: the movies dataset 30
6.2.1 Dataset 30
6.2.2 Accuracy experiments 32

7 Related Work **34**

8 Conclusion **34**

A Probabilistic model for computing connection strength **37**
A.1 Preliminaries 38
A.2 Independent edge existence 40
A.2.1 General formulae 40
A.2.2 Computing path connection strength in practice 41
A.3 Dependent edge existence 43
A.3.1 Choice nodes on the path 43
A.3.2 Options of the same choice node on the path 44
A.4 Computing the total connection strength. 46

B Solving the NLP problem by bounding option weights **47**

C Alternative WM formulae **48**
C.1 Addressing drawbacks of Equation (4) 48

Abstract

In this paper we address the problem of *reference disambiguation*. Specifically, we consider a situation where entities in the database are referred to using descriptions (e.g., a set of instantiated attributes). The objective of reference disambiguation is to identify the unique entity to which each description corresponds. The key difference between 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 also over synthetic datasets show that analysis of relationships significantly improves quality of the result.

1 Introduction

Recent surveys [4] 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 [34], handling erroneous data [35], record linkage [6, 27, 8], and so on. In this paper we address one such challenge which we refer to as *reference disambiguation*[1].

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

The reference disambiguation problem is related to the problem of *record deduplication* or *record linkage* [27, 8, 6] that often arise when multiple tables (from different data sources) are merged to create a single table. The causes of record linkage and reference disambiguation problems are similar; viz., differences in representations of objects across different data sets, data entry errors, etc. The differences between the two can be intuitively viewed using the relational terminology as follows: while the record linkage problem consists of determining when two records are the same, reference disambiguation corresponds to ensuring that references (i.e., "foreign keys"[2]) in a database point to the correct entities.

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

In this paper, we propose a domain-independent data cleaning approach for reference disambiguation, referred to as Relationship-based Data Cleaning (RelDC), that systematically exploits not only features

[1]The reference disambiguation problem has been previously identified in [32] where it was referred to as *cleaning spurious links*.

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

but also relationships among entities for the purpose of disambiguation. RelDC views the database as a graph of entities that are linked to each other via relationships. It first utilizes a feature based method to identify a set of candidate entities (choices) for a reference to be disambiguated. Graph theoretic techniques are then used to discover and analyze relationships that exist between the entity containing the reference and the set of candidates.

The primary contributions of this paper 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.

The rest of this paper 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 paper.

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.

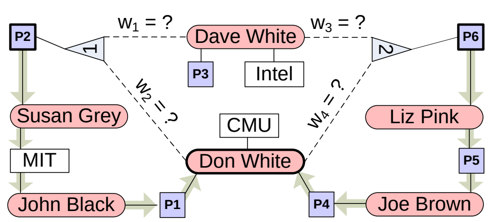

Figure 1: Graph for the publications example

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

$\langle A_1,$ ‘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$\rangle$

Figure 2: *author* records

$\langle P_1,$ ‘Databases ...’, ‘John Black’, ‘Don White’$\rangle$
$\langle P_2,$ ‘Multimedia ...’, ‘Sue Grey’, **‘D. White’**$\rangle$
$\langle P_3,$ ‘Title3 ...’, ‘Dave White’$\rangle$
$\langle P_4,$ ‘Title5 ...’, ‘Don White’, ‘Joe Brown’$\rangle$
$\langle P_5,$ ‘Title6 ...’, ‘Joe Brown’, ‘Liz Pink’$\rangle$
$\langle P_6,$ ‘Title7 ...’, ‘Liz Pink’, **‘D. White’**$\rangle$

Figure 3: *publication* records

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

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

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

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

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

At first glance, the analysis above (used to disambiguate references that could not be resolved using conventional feature-based techniques) may seem domain specific. A general principle emerges if we view the database as a graph of inter-connected entities (modeled as nodes) linked to each other via relationships (modeled as edges). Figure 1 illustrates the entity-relationship graph corresponding to the toy database consisting of *authors* and *papers* records. In the graph, entities containing references are linked to the entities they refer to. For instance, since the reference 'Sue Grey' in P_2 is unambiguously resolved to author 'Susan Grey', paper P_2 is connected by an edge to author A_3. Similarly, paper P_5 is connected to authors A_5 ('Joe Brown') and A_6 ('Liz Pink'). The ambiguity of the references 'D. White' in P_2 and P_6 is captured by linking papers P_2 and P_6 to both 'Dave White' and 'Don White' via two "choice nodes" (labeled '1' and '2' in the figure). These "choice nodes" represent the fact that the reference 'D. White' refers to either one of the entities linked to the choice nodes.

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

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

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

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

Of course, the second question is only important if the answer to the first is yes. 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 Notation and problem definition

In this section we first develop notation and then formally define the problem of reference disambiguation. The notation is summarized in Table 1.

Notation	Meaning
$\mathcal{D}$	the database
$X = \{x_i\}$	the set of all entities in $\mathcal{D}$
X^u	the set of entities that contain uncertain references
r_{ik}	the k-th reference of entity x_i
r^*_{ik}	the to-be-found entity that r_{ik} refers to
S_{ik}	the choice set for r_{ik}
$y_1, y_2, \dots, y_N$	the N elements of choice set S_{ik}; $y_j = y_{ikj}$
$G = (V, E)$	the entity-relationship graph for $\mathcal{D}$
v_i	the vertex in G that corresponds to entity x_i; $v_i = v_{x_i}$
v_{y_j}	the vertex in G that corresponds to entity y_j
v^*_{ik}	the *choice* node for reference r_{ik}
e_j	the edge $e_j = e_{ikj} = (v^*_{ik}, v_{y_j})$ $(j = 1, 2, \dots, N)$
w_j	the weight of edge e_j $(j = 1, 2, \dots, N)$; $w_j = w_{ikj}$
L	the path length limit parameter
$\mathcal{P}_L(u, v)$	the set of all L-short simple paths between nodes u and v in G
$c(u, v)$	the connection strength between nodes u and v in G
$\mathcal{N}_r(v)$	the neighborhood of node v of radius r in graph G

Table 1: Notation

3.1 References

Let $\mathcal{D}$ be the database which contains references that are to be resolved. Let X be the set of all entities[3] in $\mathcal{D}$:

$$X = \{x_1, x_2, \dots, x_{|X|}\}.$$

Each entity x_i consists of a set of m_{x_i} properties $x_i.a_1, x_i.a_2, \dots, x_i.a_{m_{x_i}}$ and of a set of n_{x_i} ($n_{x_i} \geq 0$) references $r_{i1}, r_{i2}, \dots, r_{in_{x_i}}$. Each reference r_{ik} is essentially a description and may itself consist of one or more attributes. For instance, in the example from Section 2, *paper* entities contain *one-attribute* `authorRef` references in the form ⟨*author name*⟩. If, besides author names, author affiliation were also stored in the *paper records*, then `authorRef` references would have consisted of two attributes – ⟨*author name*, *author affiliation*⟩.

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

$$S_{ik} = \{y_{ik1}, y_{ik2}, \dots, y_{ik|S_{ik}|}\}.$$

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

$$S_{ik} = \{y_1, y_2, \dots, y_N\}.$$

Let X^u denote the set of all entities x_i from X that contain at least one uncertain reference:

$$X^u = \{x_i \in X : \exists k \text{ such that } |S_{ik}| > 1\}.$$

Let us rename elements in X such that first $|X^u|$ elements $x_1, x_2, \dots, x_{|X^u|}$ belong to X^u and, consequently, the other $(|X| - |X^u|)$ elements $x_{|X^u|+1}, x_{|X^u|+2}, \dots, x_{|X|}$ do not belong to X^u.

3.2 The entity-relationship graph

RelDC views the resulting database $\mathcal{D}$ as an undirected entity-relationship graph[4] $G = (V, E)$, where V is the set of nodes and E is the set of edges. The set of nodes V is comprised of two sets $V = V^r \cup V^c$: the set of *regular* nodes V^r and the set of *choice* nodes V^c. Each regular node in V^r corresponds to an entity and each edge in E to a relationship.[5] Choice nodes will be defined later. Notation v_{x_i} or v_i denotes the vertex in G that corresponds to entity $x_i \in X$. Note that if entity u contains a reference to entity v, then the nodes in the graph corresponding to u and v are linked since a reference establishes a relationship between the two entities. For instance, `authorRef` reference from paper P to author A corresponds to "A writes P" relationship.

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

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

[5]We will concentrate primarily on binary relationships. Multiway relationships are rare and most of them can be converted to binary relationships [20]. Most of the design models/tools only deal with binary relationships, for instance ODL (Object Definition Language) supports only binary relationships.

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

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

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

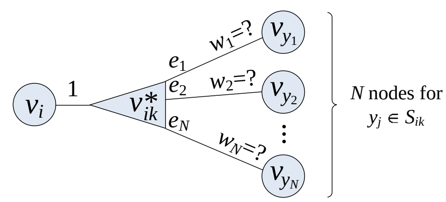

Figure 4: A choice node

3.3 The objective of reference disambiguation

To *resolve* reference r_{ik} means to choose one entity y_j from S_{ik} in order to determine r^*_{ik}. If entity y_j is chosen as the outcome of such a disambiguation, then r_{ik} is said to be *resolved to* y_j or simply *resolved*. Reference r_{ik} is said to be *resolved correctly* if this y_j is r^*_{ik}. Notice, if S_{ik} has just one element y_1 (i.e., $N = 1$), then reference r_{ik} is automatically resolved to y_1. Thus reference r_{ik} is said to be *unresolved* or *uncertain* if it is not resolved yet to any y_j and also $N > 1$.

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

The **goal** of reference disambiguation is to resolve all references as correctly as possible, that is, for each reference r_{ik} to correctly identify r^*_{ik}. We will use notation $Resolve(r_{ik})$ to refer to the procedure which resolves r_{ik}. The *goal* is thus to construct such $Resolve(.)$ which should be as accurate as possible. The *accuracy* of reference disambiguation is the fraction of references being resolved that are resolved correctly.

The **alternative goal** is for each $y_j \in S_{ik}$ to associate weight w_j that reflects the degree of confidence that y_j is r^*_{ik}. For this alternative goal, $Resolve(r_{ik})$ should label each edge e_j with such a weight. Those weights can be **interpreted** later to achieve the main goal: for each r_{ik} try to identify only one y_j as r^*_{ik} correctly. We emphasize this alternative goal since most of the discussion of RelDC approach is devoted to one approach for computing those weights. An interpretation of those weights (in order to try to identify r^*_{ik}) is a small final step of RelDC. Namely, we achieve this by picking y_j such that w_j is the largest among $w_1, w_2, \ldots, w_N$. That is, the outcome of $Resolve(r_{ik})$ is $y_j : w_j = \max_{l=1,2,\ldots,N} w_l$.

3.4 Connection Strength and Context Attraction Principle

As mentioned before, RelDC resolves references based on the context attraction principle that was discussed in Section 2. We now state the principle more formally. Crucial to the principle is the notion of *connection strength* between two entities x_i and y_j (denoted $c(x_i, y_j)$ which captures how strongly x_i and y_j are connected to each other through relationships. Many different approaches can be used to measure $c(x_i, y_j)$ and will be discussed in Section 4. Given the concept of $c(x_i, y_j)$, we can restate the context attraction principle as follows:

Context Attraction Principle: Let r_{ik} be a reference and $y_1, y_2, \ldots, y_N$ be elements of its choice set S_{ik} with corresponding option weights $w_1, w_2, \ldots, w_N$ (recall that $w_1 + w_2 + \cdots + w_N = 1$). The context attraction principle states that for all $l, j \in [1, N]$, if $c_l \geq c_j$ then $w_l \geq w_j$, where $c_l = c(x_i, y_l)$ and $c_j = c(x_i, y_j)$.

4 The RelDC approach

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

RelDC disambiguates references using the following four steps:

1. **Compute connection strengths**. For each reference r_{ik} compute the connection strength $c(x_i, y_j)$ for each $y_j \in S_{ik}$. The result is a set of equations that relate $c(x_i, y_j)$ with the option weights: $c(x_i, y_j) = g_{ij}(\overline{w})$. Here, $\overline{w} = \{w_{ikj} : i = 1, 2, \ldots, |X^u|; k = 1, 2, \ldots, n_{x_i}; j = 1, 2, \ldots, |S_{ik}|\}$ is the set of all option weights in the graph G.
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.** Utilize/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 subsections.

4.1 Computing connection strength

The connection strength $c(u, v)$ between nodes u and v should reflect how strongly nodes u and v are related to each other via relationships in the graph G. 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 subgraph shown in Figure 5 that contains two paths between nodes u and v: $p_a = u \rightarrow a \rightarrow v$ and $p_b = u \rightarrow b \rightarrow v$. Note that in the figure, node b "connects" multiple nodes, not just u and v, whereas node a "connects" only u and v. If Figure 5 were a subgraph of the author-publication graph discussed in Section 2, 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. We require measure of connection strength to be such that of the two distinct paths (or connections) between nodes u and v: $p_a = u \to a \to v$ and $p_b = u \to b \to v$, the strength via p_a be more than that via p_b. Measures such as *path length*, *network flow* do not capture the fact that $c(p_a) > c(p_b)$.

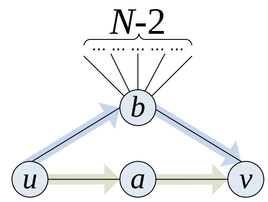

Figure 5: Motivating connection strength formula

A natural way to compute the connection strength $c(u, v)$ between node u and v (which does not suffer from the above described drawback) is to compute it as the probability to reach node v from node u via random walks in graph G. Each step of the random walk is done according to certain probability which is derived from edge labels. Such problems have been studied for graphs in the previous work under Markovian assumptions. For example, White et al. in [46] study the related problem of computing the *relative importance* of given nodes with respect to the set of "root" nodes by generalizing the PageRank algorithm [7]. They view such a graph as a Markov chain where nodes represent states of the Markov chain and probabilities are determined by edge labels.

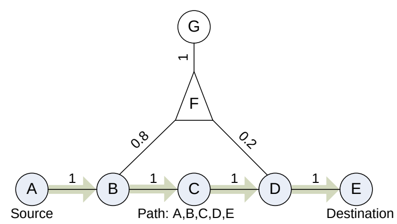

Figure 6: Sample graph.

However, there are several reasons why the existing approaches cannot be applied directly to the problem at hand. The main reason is that the Markovian assumptions do not hold for our graphs. For example, consider paths $G \to F$ and $D \to F$ in Figure 6. In that figure F is a choice node and BF and FD are its mutually exclusive option edges. In general, we can continue $G \to F$ path by following $F \to B$ link, however we cannot continue $D \to F$ path by following the same $F \to B$ link. So, the decision of whether we can or cannot follow $F \to B$ 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.

In Appendix A we have developed a model called the *probabilistic model (PM)* which treats edge weights as probabilities of existence of the edges and correctly computes the probability of reaching a node u starting from a given node v. Since the probabilistic model is somewhat complex (and will be a significant diversion from our main objective of explaining RelDC), in this section we present a *weight-based model (WM)* which is a simplification of PM. 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 path p from u to v is the probability to follow path p in graph G.

Before we formally describe WM formulae, let us show how it works for the small example in Figure 5. To capture the fact that $c(p_a) > c(p_b)$, WM measures $c(p_a)$ and $c(p_b)$ as the probabilities to follow paths p_a and p_b respectively. WM computes those probabilities as follows. 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. So the probability to reach v via path p_b is $\frac{1}{2(N-1)}$. 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 to follow p_a is $\frac{1}{2}$.

In WM, computation of $c(x_i, y_j)$ consists of two phases. The first phase discovers connections between x_i and y_j. The second phase computes/measures the strength in connections discovered by the first phase.

4.1.1 First phase of WM: discovering connections

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

Illegal paths. Not all of the discovered paths are considered when computing $c(x_i, y_j)$ (to resolve reference r_{ik}). Let $e_1, e_2, \ldots, e_N$ be the option-edges associated with the reference r_{ik}.

When resolving r_{ik}, RelDC tries do 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 $\widetilde{G} = G - v^*_{ik}$, see Figure 7. That is, $\widetilde{G}$ is graph G with node v^*_{ik} removed. Also, in general, paths considered when computing $c(x_i, y_j)$ may contain option-edges of some choice nodes. If a path contains an option-edge of a choice node, it should not contain another option-edge of the same choice node because these edges are mutually exclusive. More formally, for any path p if $e_{ikj} \in p$, then $e_{ikl} \notin p$ $(1 \le i \le |X^u|;\ 1 \le k \le n_{x_i};\ 1 \le l \le |S_{ik}|,\ l \ne j)$.

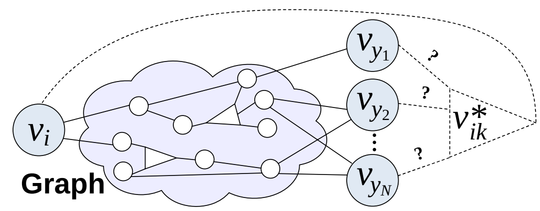

Figure 7: Graph

4.1.2 Second phase of WM: measuring connection strength

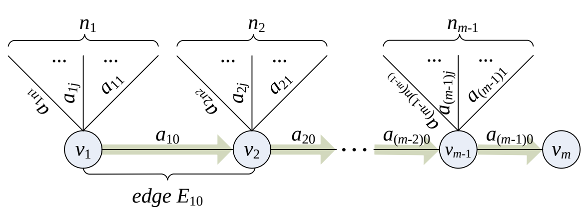

Figure 8: Computing $c(p)$ of path $p = v_1 \to v_2 \to \cdots \to v_m$. Only "possible-to-follow" edges are shown.

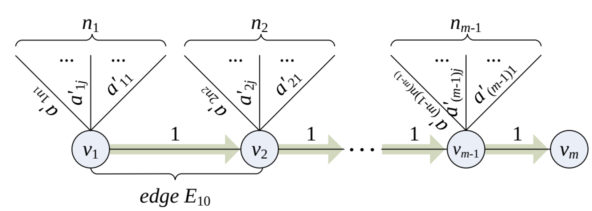

Figure 9: Computing $c(p)$: new labels under assumption that p exists.

In general, each L-short simple path p can be viewed as a sequence of m $(m \le L + 1)$ nodes $\langle v_1, v_2, \ldots, v_m \rangle$ as shown in Figure 8. Figure 8 shows that from node v_i $(i = 1, 2, \ldots, m-1)$ 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 to follow 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 "to follow path p given that p exists".

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

First of all, 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} $(i = 1, 2, \ldots, m-1)$ exists independently from other edges E_{l0} $(l = 1, 2, \ldots, m-1,\ l \neq i)$. Recall that WM is a simplification of PM presented in Appendix A. In Appendix A we show that such an assumption of independence is reasonable.

Next WM computes probability Pr_2 to follow path p given that p exists. If we assume that p exists, then situation will look like that illustrated in Figure 9. In that figure all edges are labeled with weights a'_{ij} which reflect how weights a_{ij} change if we add the assumption that path p exists. For example, $a'_{i0} = 1$ $(i = 1, 2, \ldots, m-1)$ because each edge E_{i0} exists if path p exists. For each a'_{ij}, where $j \neq 0$, either $a'_{ij} = a_{ij}$, or $a'_{ij} = 0$. To understand why a'_{ij} can be zero, consider path p_1 = 'Don White' $\to P_4 \to$ Joe $\to P_5 \to$ Liz $\to P_6 \to$ '2' $\to$ 'Dave White' in Figure 1 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. So, 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 $a'_{ij} = 0$.

WM computes probability Pr_2 "to follow path p given that p exists" as the product of probabilities to follow each edge on p. In WM, the probability to follow an edge is proportional to the weight of the edge. For example, the probability to follow edge E_{10} in Figure 9 is: $\frac{1}{1+a'_{11}+a'_{12}+\cdots+a'_{1n_1}}$. 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 $\mathcal{P}_L(u, v)$:

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

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

Connection strengths in toy database. Let us compute connection strengths c_1, c_2, c_3, and c_4 for the toy database illustrated in Figure 1. Those connection strength are defined as follows: $c_1 = c(P_2, \text{'Dave White'})$, $c_2 = c(P_2, \text{'Don White'})$, $c_3 = c(P_6, \text{'Dave White'})$, and $c_4 = c(P_6, \text{'Don White'})$. Later, those connection strengths will be used to compute option weights w_1, w_2, w_3, and w_4.

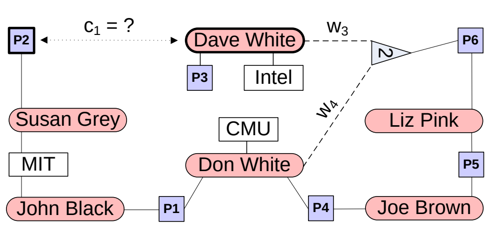

Figure 10: Computing $c_1 = c(P_2, \text{Dave})$: $\widetilde{G} = G -$ '1'.

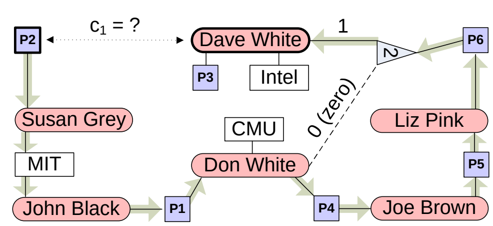

Figure 11: Computing $c_1 = c(P_2, \text{Dave})$: under assumption that path ($P_2 \rightsquigarrow$ 'Dave White') exists. Edge '2' $\to$ 'Dave' exists, therefore edge '2' $\to$ 'Don' does not exist.

Consider first computing $c_1 = c(P_2, \text{'Dave White'})$ 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 $\widetilde{G} = G -$ '1' is shown in Figure 10. The next step is to discover all L-short simple paths in graph $\widetilde{G}$ between P_2 and 'Dave White'. Let us set $L = \infty$, then there is only one such path: $p_1 = P_2 \to$ Susan $\to$ MIT $\to$ John $\to P_1 \to$ Don $\to P_4 \to$ Joe $\to P_5 \to$ Liz $\to P_6 \to$ '2' $\to$ 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 $Pr_1 = w_3$. Now we assume that p_1 exists and compute the probability Pr_2 to follow p_1 given that p_1 exists on the graph shown in Figure 11. That probability is $Pr_2 = \frac{1}{2}$. Thus $c(p_1) = Pr_1 Pr_2 = \frac{w_3}{2}$. The same result can be obtained by directly applying Equation (1). After computing c_2, c_3, and c_4 in a similar fashion we have:

$$\begin{cases} c_1 = c(P_2, \text{'Dave White'}) = \frac{w_3}{2} = c(p_1) \\ c_2 = c(P_2, \text{'Don White'}) = 1 \quad = c(P_2 \to \text{Susan} \to \text{MIT} \to \text{John} \to P_1 \to \text{'Don White'}) \\ c_3 = c(P_6, \text{'Dave White'}) = \frac{w_1}{2} \\ c_4 = c(P_6, \text{'Don White'}) = 1 \end{cases} \tag{3}$$

Notice, the toy database is small and 'MIT' connects only two authors. In more realistic examples, 'MIT' will connect many authors, so connections via 'MIT' will be weak.

4.2 Determining equations for option-edge weights

Given the connection strength measures $c(x_i, y_j)$ for each unresolved reference r_{ik} and its options $y_1, y_2, \dots, y_N$, 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 $c_l \geq c_j$ then $w_l \geq w_j$ is appropriate, where $c_l = c(x_i, y_l)$ and $c_j = c(x_i, y_j)$. In particular, we use the strategy where weights $w_1, w_2, \dots, w_N$ are proportional to the corresponding connection strengths: $w_j c_l = w_l c_j$. Using this strategy and considering that $w_1 + w_2 + \dots + w_N = 1$, weight w_j $(j = 1, 2, \dots, N)$ is computed as:

$$w_j = \begin{cases} \frac{c_j}{c_1 + c_2 + \dots + c_N} & \text{if } (c_1 + c_2 + \dots + c_N) > 0; \\ \frac{1}{N} & \text{if } (c_1 + c_2 + \dots + c_N) = 0. \end{cases} \tag{4}$$

For instance, for the toy database we have:

$$\begin{cases} w_1 = c_1/(c_1 + c_2) = \frac{w_3}{2}/(1 + \frac{w_3}{2}) \\ w_2 = c_2/(c_1 + c_2) = 1/(1 + \frac{w_3}{2}) \\ w_3 = c_3/(c_3 + c_4) = \frac{w_1}{2}/(1 + \frac{w_1}{2}) \\ w_4 = c_4/(c_3 + c_4) = 1/(1 + \frac{w_1}{2}) \end{cases} \tag{5}$$

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.

Solving equations for toy database. Before we discuss how such equations can be solved in general, let us first solve Equations (5) for the toy example. Those equations, given an additional constraint that w_1, w_2, w_3, and w_4 should be in $[0, 1]$, have a unique solution $w_1 = 0$, $w_2 = 1$, $w_3 = 0$, and $w_4 = 1$. Once we have computed the weights, RelDC will interpret these weights to resolve the references. In the toy example, weights $w_1 = 0$, $w_2 = 1$, $w_3 = 0$, and $w_4 = 1$ will lead RelDC to resolve 'D. White' in both P_2 and P_6 to 'Don White'.

General case. In general case, Equations (1), (2), and (4), define each option weight as a function of other option weights: $w_{ikj} = f_{ikj}(\overline{w})$. The exact function for w_{ikj} is determined by Equations (1), (2), and (4), and by the paths that exists between v_i and $v_{y_{ikj}}$ in G. In practice, often $f_{ikj}(\overline{w})$ is constant leading to the equation of the form $w_{ikj} = const$.

$$\begin{cases} w_{ikj} = f_{ikj}(\overline{w}) & (i = 1, 2, \dots, |X^u|; k = 1, 2, \dots, n_{x_i}; j = 1, 2, \dots, |S_{ik}|) \\ 0 \leq w_{ikj} \leq 1 & (i = 1, 2, \dots, |X^u|; k = 1, 2, \dots, n_{x_i}; j = 1, 2, \dots, |S_{ik}|) \end{cases} \tag{6}$$

The goal is to find such a combination of weights w_{ikj} that best "satisfies" System (6). Since System (6), might not have an exact solution, we transform $w_{ikj} = f_{ikj}(\overline{w})$ equations into the form $f_{ikj}(\overline{w}) - \delta_{ikj} \leq w_{ikj} \leq f_{ikj}(\overline{w}) + \delta_{ikj}$. Here variable δ_{ikj}, called *tolerance*, can take on any real nonnegative value. The problem transforms into solving the nonlinear programming problem (NLP) where the constraints are specified by the inequalities above and the objective is to minimize the sum of all δ_{ikj}:

$$\begin{cases} \text{Constraints:} & \\ f_{ikj}(\overline{w}) - \delta_{ikj} \leq w_{ikj} \leq f_{ikj}(\overline{w}) + \delta_{ikj} & (i = 1, 2, \dots, |X^u|; k = 1, 2, \dots, n_{x_i}; j = 1, 2, \dots, |S_{ik}|) \\ 0 \leq w_{ikj} \leq 1 & (i = 1, 2, \dots, |X^u|; k = 1, 2, \dots, n_{x_i}; j = 1, 2, \dots, |S_{ik}|) \\ 0 \leq \delta_{ikj} & (i = 1, 2, \dots, |X^u|; k = 1, 2, \dots, n_{x_i}; j = 1, 2, \dots, |S_{ik}|) \\ & \\ \text{Objective: Minimize} \sum_{i,k,j} \delta_{ikj} & \end{cases} \tag{7}$$

System (7) always has a solution. To show that, it is sufficient to prove that there is at least one solution that satisfied the constraints of System (7). Let us prove that by constructing such a solution. Notice, functions $f_{ikj}(\overline{w})$ (for all i, k, and j) are such that $0 \leq f_{ikj}(\overline{w}) \leq 1$, if $0 \leq w_{ikj} \leq 1$ (for all i, k, j). Thus the following combination: $w_{ikj} = 0$ and $\delta_{ikj} = 1$ (for all i, k, and j) is a solution that satisfies the constraints of System (7), though it does not satisfy the objective in general. The goal, of course, is to find a better solution by requiring that $\sum_{i,k,j} \delta_{ikj}$ is minimized. The pseudo code for the above procedure will be discussed in Section 5.1.1.

Iterative solution. The straightforward approach to solving the resulting NLP problem (7) 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 (7) as well, e.g. in [29] we sketch another approximate algorithm for solving (7) which first computes bounding intervals for all option weights w_{ikj}'s and then employs techniques from [10, 9, 11]. That method is more involved than the iterative solution, which we will present next. The pseudo code for the iterative method is given in Figure 13 in Section 5.1.2.

The iterative method first iterates over each reference r_{ik} ($i = 1, 2, \dots, |X^u|$; $k = 1, 2, \dots, n_{x_i}$) and assigns weight of $\frac{1}{|S_{ik}|}$ to each w_{ikj} ($j = 1, 2, \dots, |S_{ij}|$). It then starts its major iterations in which it first computes $c(x_i, y_{ikj})$ (for all i, k, and j) using Equation (2). After $c(x_i, y_{ikj})$ (for all i, k, and j) are computed, they are used to compute w_{ikj} (for all i, k, and j) using Equation (4). Note that the values of w_{ikj} (for all i, k, and j) will change from $\frac{1}{|S_{ik}|}$ 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 one iteration of the iterative method for the example above. First $w_1 = w_2 = \frac{1}{2}$ and $w_3 = w_4 = \frac{1}{2}$. Next $c_1 = \frac{1}{4}$, $c_2 = 1$, $c_3 = \frac{1}{4}$, and $c_4 = 1$. Finally, $w_1 = \frac{1}{5}$, $w_2 = \frac{4}{5}$, $w_3 = \frac{1}{5}$, and $w_4 = \frac{4}{5}$. If we stop the algorithm at this point and interpret w_j's, then the RelDC's answer is 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 *Iter-RelDC* and denote the implementation that uses a solver as *Solv-RelDC*. For both Iter-RelDC and Solv-RelDC, after the weights are computed, those weights are **interpreted** to produce the final result, as discussed in Section 4. It turned out that the accuracy of Iter-RelDC (with a small number of iterations, such as 10–20) and of Solv-RelDC is practically identical. This is because even though the iterative method does not find the exact weights, those weights 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_{ik} and deciding which entity among $y_1, y_2, \dots, y_N$ from S_{ik} is r^*_{ik}, RelDC chooses such y_j that w_j is the largest among $w_1, w_2, \dots, w_N$. Notice, to resolve r_{ik} we could have also combined w_j weights with feature-based similarities $FBS(x_i, y_j)$ (e.g., as a weighted sum), but we do not study that approach in this paper.

5 Implementations of RelDC

In this section we discuss several implementations of RelDC and present key optimizations. The complete list of optimizations, their taxonomy and analysis are presented in [29]. We will conclude this section with the computational analysis of RelDC.

5.1 Iterative and solver implementations of RelDC

The NLP problem in Equation (7) can be solved iteratively or using a solver. In this section we shall present pseudo code for naïve implementations of Solv-RelDC and Iter-RelDC. In the subsequent sections we shall discuss how to optimize these naïve implementations.

5.1.1 Solver

Figure 12 shows an outline of Solv-RelDC which we have discussed in Section 4. In lines 1–2, if greedy implementation of All-Paths is used (see Section 5.3), the algorithm initializes weights. Initial values of option weights $w_1, w_2, \dots, w_N$ of each choice node v^*_{ik} are assigned such that $w_1 = w_2 = \dots = w_N = \frac{1}{N}$ and $w_1 + w_2 + \dots + w_N = 1$. Lines 3–9 correspond to creating equations for connection strengths $c(x_i, y_{ikj})$ (for all i, k, j) described in Section 4.1: each $c(x_i, y_{ikj})$ is derived based on the simple paths that exist between nodes for x_i and y_{ikj} in the graph. Lines 10–13 correspond to the procedure from Section 4.2 that construct the equations for option weighs w_{ikj} (for all i, k, j). Then, in Line 14, the algorithm takes the NLP problem shown in Equation (7) 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 pseudo code in Figure 13 formalizes the Iter-RelDC procedure described in Section 4.3. Iter-RelDC first initializes weights. Then it iterates recomputing new values for $c(x_i, y_{ikj})$ and w_{ikj} (for all i, k, j). Finally, all the references are resolved by interpreting the weights.

NAÏVE-SOLV-RELDC()
if GRD-RelDC **then**
  INITIALIZE($\overline{w}$)
for $i \leftarrow 1$ **to** $|X^u|$ **do**
  for $k \leftarrow 1$ **to** n_{x_i} **do**
    $\widetilde{G} \leftarrow G - v^*_{ik}$
    for $j \leftarrow 1$ **to** $|S_{ik}|$ **do**
      $\mathcal{P}_L(x_i, y_{ikj}) \leftarrow$ ALL-PATHS($\widetilde{G}, x_i, y_{ikj}, L$)
      $EQ[c(x_i, y_{ikj})] \leftarrow$ EQ-CON-STRENGTH($\overline{w}, \mathcal{P}_L(x_i, y_{ikj})$)
      delete $\mathcal{P}_L(x_i, y_{ikj})$ *//free storage*
for $i \leftarrow 1$ **to** $|X^u|$ **do**
  for $k \leftarrow 1$ **to** n_{x_i} **do**
    for $j \leftarrow 1$ **to** $|S_{ik}|$ **do**
      $EQ[w_{ikj}] \leftarrow$ EQ-ASSIGN-WEIGHT($j, c(x_i, y_{ik1}), c(x_i, y_{ik2}), c(x_i, y_{ik|S_{ik}|})$)
$S \leftarrow$ PREPARE-FOR-SOLVER(all $EQ[w_{ikj}]$)
$\overline{w} \leftarrow$ SOLVE-USING-SOLVER(S)
INTERPRET($\overline{w}$)

INITIALIZE($\overline{w}$)
for $i \leftarrow 1$ **to** $|X^u|$ **do**
  for $k \leftarrow 1$ **to** n_{x_i} **do**
    for $j \leftarrow 1$ **to** $|S_{ik}|$ **do**
      $w_{ikj} \leftarrow \frac{1}{|S_{ik}|}$

Figure 12: NAÏVE-SOLV-RELDC

NAÏVE-ITER-RELDC(N_{iter})
INITIALIZE($\overline{w}$)
MAIN-LOOP($\overline{w}$, N_{iter})
INTERPRET($\overline{w}$)

MAIN-LOOP($\overline{w}$, N_{iter})
for $l \leftarrow 1$ **to** N_{iter} **do**
  for $i \leftarrow 1$ **to** $|X^u|$ **do**
    for $k \leftarrow 1$ **to** n_{x_i} **do**
      $\widetilde{G} \leftarrow G - v^*_{ik}$
      for $j \leftarrow 1$ **to** $|S_{ik}|$ **do**
        $\mathcal{P}_L(x_i, y_{ikj}) \leftarrow$ ALL-PATHS($\widetilde{G}, x_i, y_{ikj}, L$)
        $c(x_i, y_{ikj}) \leftarrow$ CON-STRENGTH($\overline{w}, \mathcal{P}_L(x_i, y_{ikj})$)
        delete $\mathcal{P}_L(x_i, y_{ikj})$ *//free storage*
  for $i \leftarrow 1$ **to** $|X^u|$ **do**
    for $k \leftarrow 1$ **to** n_{x_i} **do**
      for $j \leftarrow 1$ **to** $|S_{ik}|$ **do**
        $w_{ikj} \leftarrow$ ASSIGN-WEIGHT($j, c(x_i, y_{ik1}), c(x_i, y_{ik2}), c(x_i, y_{ik|S_{ik}|})$)

Figure 13: NAÏVE-ITER-RELDC

5.1.3 Bottleneck of RelDC

To optimize RelDC for performance we need to understand where it spend most of its computation time. The most computationally expensive part of both Iter-RelDC and Solv-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 and 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. AllPaths 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 Iter-RelDC. When resolving reference r_{ik}, see Figure 4, Iter-RelDC can use a threshold to prune several y_j's from S_{ik}. If the current weight w_j is too small with respect to the rest of weights $w_1, w_2, \ldots, w_{j-1}, w_{j+1}, \ldots, w_N$, then RelDC will assume y_j cannot be r^*_{ik} and will remove y_j from S_{ik}.

The threshold is computed per choice basis. For v^*_{ik} it is computed as $T = \alpha \cdot \frac{1}{N}$, where α $(0 \leq \alpha < 1)$ is a real number (a fixed parameter).[7] This optimization improves the efficiency since if y_j is removed from S_{ik}, then Iter-RelDC will not recompute $\mathcal{P}_L(x_i, y_j)$, $c(x_i, y_j)$, and w_j 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 type *node_type*1 → *node_type*2 → *node_type*4 → *node_type*1 are of interest. Some of such rules are easy to specify, however it is clear that for a generic framework there should be some method (e.g., a language) for an analyst to specify more complex rules. Our ongoing work addresses the problem of such a language [43].

5.3 Depth-first and greedy implementation of AllPaths.

RelDC utilizes AllPaths procedure to discover all L-short simple paths between two nodes. We have considered two approaches for implementing AllPaths algorithm: the depth-first (DF-AllPaths) and greedy (GRD-AllPaths) shown in Figures 14 and 15 respectively.[9]

The reason for having those two implementations is as follows. The DF-AllPaths is a good choice if skipping of paths is *not allowed*: we shall show that in this case DF-AllPaths is better in terms of time and space complexity than its greedy counterpart. However GRD-AllPaths is a better option if one is interested in fine-tuning the *accuracy vs. performance* trade-off by restricting the running time of the AllPaths algorithm. The reason for this is as follows. If DF-AllPaths is stopped abruptly at some point in the middle of its execution, then certain important paths can still be not discovered. To address this drawback, GRD-AllPaths discovers the most important paths first and least important last.

5.3.1 Depth-first and greedy algorithms

As can be seen from Figures 14 and 15 the depth-first and greedy algorithms are quite similar. The difference between those two is that DF-AllPaths utilizes a stack to account for intermediate paths while GRD-AllPaths utilizes a priority queue. The key in this queue is the connection strengths of intermediate

[7]The typical choices for α in our experiments were 0.0 (i.e., the optimization is not used), 0.2 and 0.3.
[8]This optimization has not been used in our experiments.
[9]All of the optimizations mentioned in this paper can be applied to both of these approaches.

```
DF-ALL-PATHS(G, u, v, L)
1  R ← ∅
2  Push(S, u)
3  while NotEmpty(S) do
4     p ← Pop(S)
5     if LastNode(p) = v then
6        R = R ∪ p
7     else if Length(p) < L then
8        DF-EXPAND-PATH(p, S)
9  return R

DF-EXPAND-PATH(p, S)
1  x ← LastNode(p)
2  for each z ∈ V : (x, z) ∈ E do
3     if ISLEGAL(p → z) then
4        Push(S, p → z)
```

Figure 14: DF-AllPaths

```
GRD-ALL-PATHS(G, u, v, L)
1  R ← ∅
2  Insert(Q, u, 1)
3  while NotEmpty(Q) and STOPCONDITION(.) = false do
4     p ← Get(Q)
5     if LastNode(p) = v then
6        R = R ∪ p
7     else if Length(p) < L then
8        GRD-EXPAND-PATH(p, Q)
9  return R

GRD-EXPAND-PATH(p, Q)
1  x ← LastNode(p)
2  for each z ∈ V : (x, z) ∈ E do
3     if ISLEGAL(p → z) then
4        Insert(Q, p → z, c(p → z))
```

Figure 15: GRD-AllPaths

paths. Also, GRD-AllPaths stops if the stop conditions are met (Line 3 in Figure 15) even if not all paths have been examined yet, whereas DF-AllPaths discovers all paths without skipping any paths.

Both algorithms look for $u \rightsquigarrow v$ paths and start with 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 result set R 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 $p \to z$ is a legal paths, then it inserts this path into the stack (or queue) for further consideration.

The STOPCONDITION() procedure in Line 3 of GRD-AllPaths 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 GRD-ALL-PATHS. For example, thresholds can be used to limit such parameters as 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 R and so on.

Thus GRD-AllPaths discovers most important paths first and least important paths last and can be stopped at a certain point whereas DF-Paths discovers all paths.

5.3.2 Paths storage

When looking for all L-short simple paths of type $u \rightsquigarrow v$, AllPaths maintains several intermediate paths. To store paths compactly and efficiently it uses a data structure called a *paths storage*. DF-ALL-PATHS and GRD-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 of the paths share common prefix which gives an opportunity to save space. For example, to store paths shown in Figure 16 it is not necessary to keep four separate lists shown in Figure 17 of lengths 2, 3, 4, and 4 respectively. It is more efficient to store them as shown in Figure 18 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

$p_1 = u \to 1$	$l_1 = 1 \to u$	$l_1 = 1 \to u$
$p_2 = u \to 1 \to 2$	$l_2 = 2 \to 1 \to u$	$l_2 = 2 \to l_1$
$p_3 = u \to 1 \to 2 \to 3$	$l_3 = 3 \to 2 \to 1 \to u$	$l_3 = 3 \to l_2$
$p_4 = u \to 1 \to 2 \to 4$	$l_4 = 4 \to 2 \to 1 \to u$	$l_4 = 4 \to l_2$
Figure 16: Example of paths.	Figure 17: Separate lists for paths	Figure 18: The paths storage

when the algorithm creates a new intermediate path $p \to z$, the following holds:

1. p is the prefix of $p \to z$
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 AllPaths procedure. The DF-AllPaths and GRD-AllPaths procedures in Figures 14 and 15 are conceptually different only in Lines 2,3 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 GRD-RelDC. But we will analyze only the differences in complexity which arise due to DF-RelDC using a stack and GRD-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 [17].

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 GRD-EXPAND-PATH procedure.

Thus, if the goal is to discover all L-short simple paths *without skipping* any paths, then the DF-AllPaths is expected to show better results than GRD-AllPaths. However, since the greedy version discovers the most important path first, it is a better choice in terms of the *accuracy vs. performance* trade-off than its depth-first counterpart. That is, the greedy version is expected to be better if the execution time of the algorithm needs to be restricted.

5.4 NBH optimization: utilizing neighborhoods for path pruning.

The NBH optimization is the most important performance optimization presented in this paper. It consistently achieves 1–2 orders of magnitude performance improvement under variety of conditions.

The *neighborhood* $\mathcal{N}_r(u)$ of node u of radius r is the set of all the nodes that are reachable from u via at most r edges. Each member of the set is tagged with "the minimum distance to u" information. The intuitive definition presented above can be rephrased formally: for graph $G = (V, E)$, the neighborhood of node u of radius r is defined as the following set of pairs: $\mathcal{N}_r(u) = \{(v, d) : v \in V,\ d = MinDist(v, u),\ d \leq r\}$.

Recall, when resolving reference r_{ik}, the algorithm will need to invoke AllPaths for N pairs – to compute $\mathcal{P}_L(x_i, y_j)$ $(j = 1, 2, \ldots, N)$ see Figure 4. This computation can be optimized by (a) computing neighborhood $\mathcal{N}_r(v_i)$; (b) discovering paths not from v_i to v_{y_j} but in reverse order: from v_{y_j} to v_i; and (c) exploiting $\mathcal{N}_r(v_i)$ to prune certain intermediate paths as explained below, see Figure 21.

When resolving references of entity x_i, the algorithm first computes the neighborhood $\mathcal{N}_r(v_i)$ of v_i of radius r, where $r \leq L$. The neighborhood is computed once per each $x_i \in X^u$ and discarded after x_i is processed. Figure 19 shows an outline of the modified MAIN-LOOP procedure of Iter-RelDC that reflects the changes needed for using the NBH optimization. The new and modified lines are marked in the figure.

MAIN-LOOP($\overline{w}$, N_{iter})
⋮
2 **for** $i \leftarrow 1$ **to** $|X^u|$ **do**
▷2.5 $\mathcal{N}_r(v_i) \leftarrow$ COMPUTE-NBH(v_i, r)
3 **for** $k \leftarrow 1$ **to** n_{x_i} **do**
4 $\widetilde{G} \leftarrow G - v^*_{ik}$
5 **for** $j \leftarrow 1$ **to** $|S_{ik}|$ **do**
▷6 $\mathcal{P}_L(x_i, y_{ikj}) \leftarrow$ ALL-PATHS($\widetilde{G}, x_i, y_{ikj}, L, \mathcal{N}_r(v_i)$)
7 $c(x_i, y_{ikj}) \leftarrow$ CON-STRENGTH($\overline{w}, \mathcal{P}_L(x_i, y_{ikj})$)
8 **delete** $\mathcal{P}_L(x_i, y_{ikj})$ *//free storage*
▷8.5 **delete** $\mathcal{N}_r(v_i)$
⋮

Figure 19: NBH: Modified MAIN-LOOP

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

Figure 20: PRUNE-PATH-NBH()

AllPaths procedure shown in Figures 14 and 15 should be modified as well to be used with NBH. First it should be able to accept an additional parameter $\mathcal{N}_r(v_i)$) 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($p, \mathcal{N}_r(v_i)$) = `false` **then**

This will allow to prune certain paths using the NBH optimization.

The PRUNE-PATH-NBH procedure is provided in Figure 20. It takes advantage of $\mathcal{N}_r(v_i)$ to identify if a given intermediate path $p = v_{y_j} \rightsquigarrow x$ can be pruned or not. First it determines the length m of path p. If m is such that $(m + r) < L$, then it cannot prune p, so it returns `false`. However, if $(m + r) \geq L$, then x must be inside $\mathcal{N}_r(v_i)$. If it is not inside, then path p is pruned, because there cannot be an L-short path $v_{y_j} \overset{p}{\rightsquigarrow} x \overset{p_1}{\rightsquigarrow} v_i$ for any path $p_1 : x \overset{p_1}{\rightsquigarrow} v_i$. If x is inside $\mathcal{N}_r(v_i)$, then the procedure retrieves from $\mathcal{N}_r(v_i)$ the minimum distance d from x to v_i. This distance d should be such that $(m + d) \leq L$: otherwise path p is pruned.

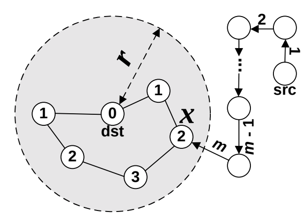

Figure 21: Neighborhood

The NBH optimization can be improved further. Let us introduce a new term – the *actual radius* of neighborhood $\mathcal{N}_r(u)$: $r_{act} = \max_{v:(v,d) \in \mathcal{N}_r(u)}(MinDist(u, v))$. While usually $r_{act} = r$, sometimes[10] $r_{act} < r$. The latter happens when nodes from the neighborhood of u and their incident edges form a cluster which is not connected to the rest of the graph (or this cluster is the whole graph). In this situation $\mathcal{N}_{r_{act}}(u) = \mathcal{N}_l(u)$ for all l ($r_{act} \leq l < \infty$). In other words, we know the neighborhood of u of radius $r = \infty$. Regarding searching all simple paths as described above, 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 20.

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

5.5 Storing discovered paths explicitly.

Once the paths are discovered on the first iteration of Iter-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.

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

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 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 Iter-RelDC, DF-AllPaths, NBH, and the optimization that stores paths. However, certain implementations and optimizations are mutually exclusive. They are as follows:

1. Iter-RelDC vs. Solv-RelDC
2. DF-RelDC vs. GRD-RelDC
3. Solv-RelDC and Storing Paths

Also, there are some compatibility issues of GRD-RelDC with Solv-RelDC. Notice, GRD-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-SOLV-RELDC procedure assign to option-edge weights some initial values.

5.7 Computational complexity of RelDC.

Let us analyze the computational complexity of non-optimized Iter-RelDC with GRD-AllPaths procedure. GRD-AllPaths 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. GRD-AllPaths(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 $c(u, v)$.

GRD-AllPaths 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 there are N_{ref} references that need to be disambiguated, typically $N_{ref} = O(|V|)$. The average cardinality of their choice sets is typically a constant, or $O(|V|)$. Thus, Iter-RelDC will need to compute $c(x_i, y_j)$ for at most $O(|V|^2)$ pairs of (x_i, y_j) per iteration. Therefore the time complexity of an iteration of Iter-RelDC is $O(|V|^2)$ multiplied by the complexity of the GRD-AllPaths procedure, plus the cost to construct all choice 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 GRD-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 [2]. The system runs on a 1.7GHz Pentium machine. We test and compare the following implementations of RelDC:

1. **Iter-RelDC** vs. **Solv-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, Iter-RelDC is assumed by default. Solv-RelDC is applicable only to more restricted problems (e.g., smaller graphs and smaller values of L) than Iter-RelDC. Solv-RelDC is also slower than Iter-RelDC.

2. **WM-RelDC** vs. **PM-RelDC**. The prefixes indicate whether the weight-based model (WM), described in Section 4.1.2, or probabilistic model (PM), described in Appendix A in appendix, has been used for computing connection strengths. By default WM-RelDC is assumed.

3. **DF-RelDC** vs. **GRD-RelDC**. The prefixes specify whether the depth-first (DF) or greedy (GRD) implementation of AllPaths 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-Iter-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[1] and HPSearch[3]. 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.

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

Table 2: Sample content of the *publication* table derived from CiteSeer.

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

Table 3: Sample content of the *author* table derived from HPSearch. Author from CiteSeer not found in HPSearch are also added.

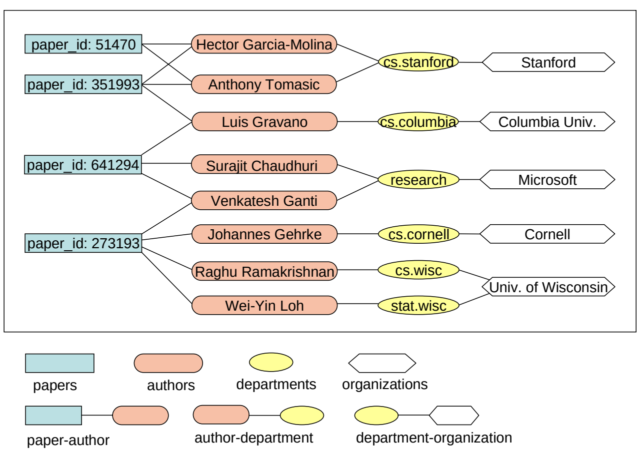

Figure 22: Sample entity-relationship graph for *Publications* dataset. A paper with `paper_id` of, say 51470 can be retrieved from CiteSeer via URL: `http://citeseer.ist.psu.edu/51470.html`

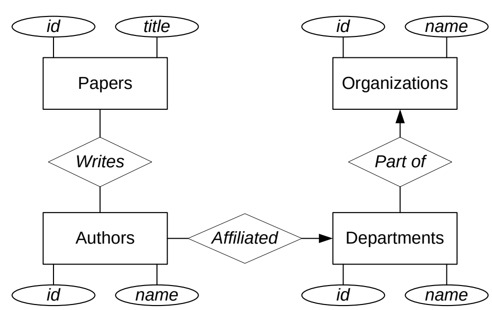

Figure 23: E/R diagram for RealPub

Tables 2 and 3 show sample content of two tables derived from CiteSeer and HPSearch based on which the corresponding entity-relationship graph is constructed for RelDC. Figure 22 shows a sample entity-relationship graph that corresponds to the information in those two tables.

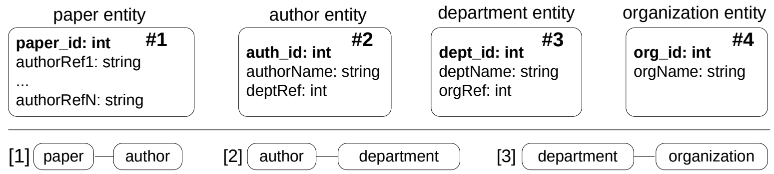

Figure 24: No affiliation in *paper* entities, thus FBS cannot use affiliations.

The various types of entities and relationships present in RealPub are shown in Figure 24. RealPub data consists of 4 *types* of entities: papers (255K), authors (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 (v_A, v_D) edge in the graph. If department D is a part of organization U, then there is (v_D, v_U) edge. If paper P is written by author A, then there is (v_A, v_P) edge. For each of the 573K `authorRef` references, feature-based similarity (FBS) was used to construct its choice 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.

This similarity has been used to construct choice sets for all `authorRef` references. As the result, 86.9% (498K) of all `authorRef` references had choice 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, that 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 our context, *accuracy* is the fraction of all `authorRef` references that are resolved correctly. This definition includes references that have choice sets of cardinality 1.

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 choice set of each such reference is at least two. For 8% of those references there were no $x_i \rightsquigarrow y_j$ 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 have performed 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 experiments, 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 which we describe below.

We first remove from RealPub all the paper entities which 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 choice sets is 1 and there is nothing to resolve. For the rest 18% the problem is more interesting: the cardinality of their choice sets is at least 2. Figure 25 shows the outcome for those 18%.

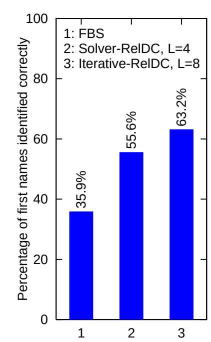

Figure 25: RealPub: Identifying first names

Notice that the reference disambiguation problem tested in the above experiment is of a limited nature – the tasks of identifying the correct first name of the author and 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 Experiment 1 and Experiment 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

[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.

the full first name in an `authorRef`, instead of just the first initial, would have allowed to significantly narrow down the choice 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 mentioned above, 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 Figure 24. 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 SynPub1). SynPub1 uses *uncertainty of type 1* defined as follows. There are $N_{auth} = 1000$ unique authors in SynPub1, but there are only N_{name} ($1 \leq N_{name} \leq N_{auth}$)

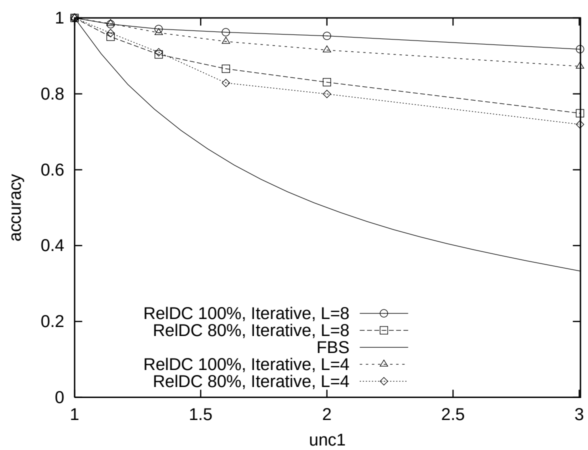

Figure 26: SynPub1: Accuracy vs. unc_1

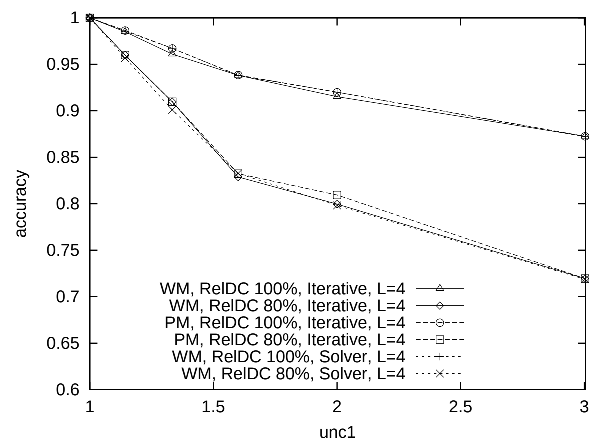

Figure 27: SynPub1: The accuracy results for Solv-RelDC, Iter-RelDC, and Iter-RelDC with PM model are comparable.

unique `authorName`'s. We construct the `authorName` of the author with $\texttt{id} = k$ ($k = 0, 1, \ldots, 999$) 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 $\texttt{id} = 1$ and $\texttt{id} = 751$ 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 large: for any given author there is exactly one another 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 Figures 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-Iter-RelDC, PM-Iter-RelDC, WM-Solv-RelDC implementations are comparable. Figure 28 shows that Iter-RelDC is the fastest implementation among them. The same trend has been observed for all other tested cases. □

Experiment 4 (Accuracy on SynPub2). SynPub2 uses *uncertainty of type 2*. In SynPub2, `authorName`'s

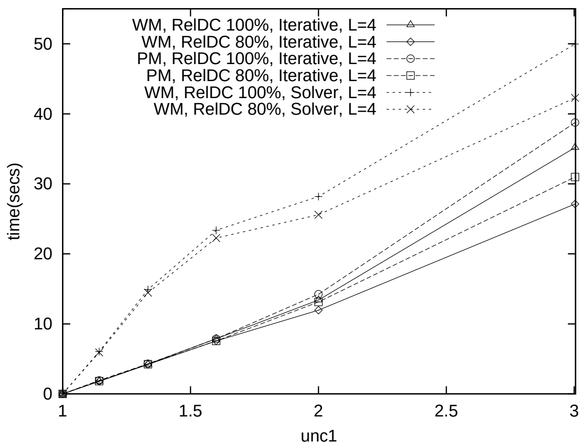

Figure 28: SynPub1: Iter-RelDC is more efficient than (i)Solv-RelDC and (ii)Iter-RelDC with PM model.

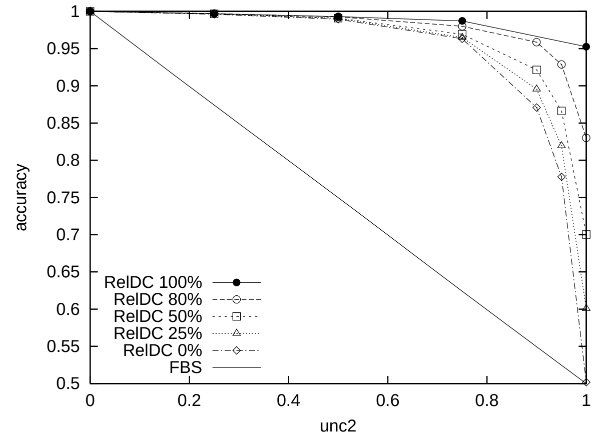

Figure 29: SynPub2: Accuracy vs. unc_2

(in *author* entities) are constructed such that the following holds, see Figure 24. If an `authorRef` reference (in a *paper* entity) is in the format "*first name* + *last name*" then it matches only one (correct) author. 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 then 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 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 what effect the number of relationships and the number of relationship *types* have 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] 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: $unc_1 = 1.75$, $unc_1 = 2.00$,

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

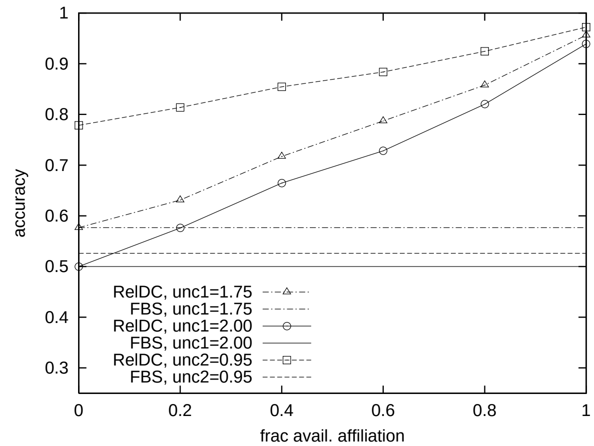

Figure 30: SynPub: Accuracy vs. fraction of available affiliation.

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. □

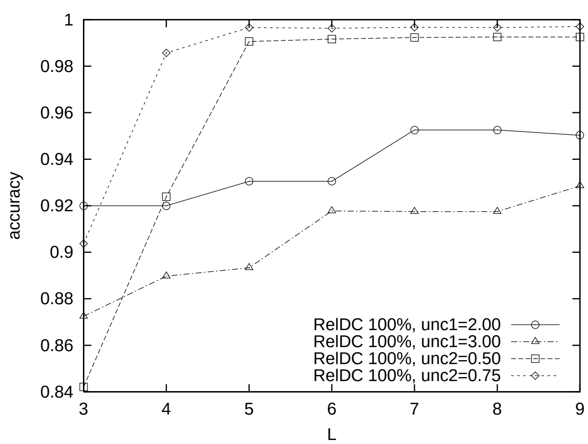

Figure 31: SynPub: Accuracy vs. path length limit

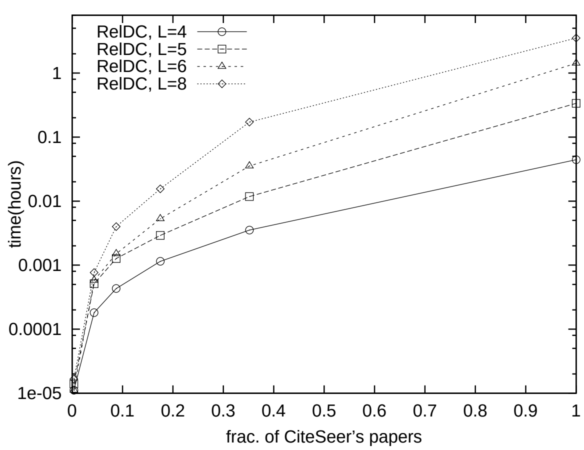

Figure 32: RealPub: Time vs. frac of RealPub's papers

Experiment 7 (The neighborhood optimization). We have developed several optimizations which make RelDC 1–2 orders of magnitude more efficient. Figure 33 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 34 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. □

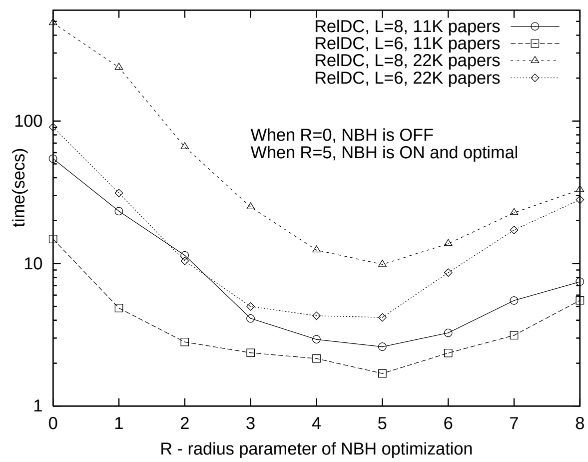

Figure 33: RealPub: Optimizations are crucial.

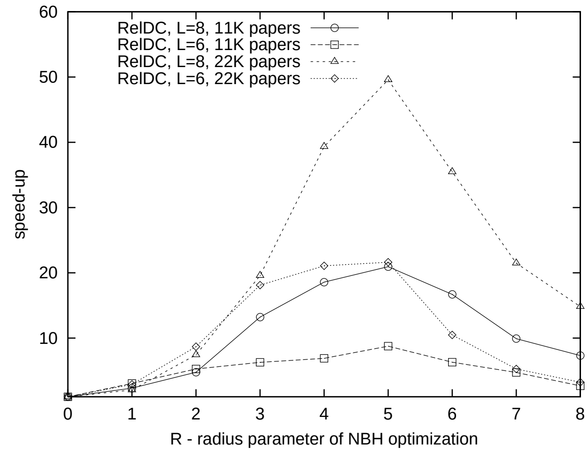

Figure 34: RealPub: speedup achieved by NBH optimization.

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 32 shows the execution time of RelDC as a function of the fraction of papers from RealPub dataset, e.g. 1.0 corresponds to all papers in RealPub (the whole CiteSeer) dataset. □

Experiment 9 (Greedy vs. Depth-first AllPaths implementations). This experiment compares accuracy and performance of greedy and depth-first versions of RelDC. As the name suggests, 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 AllPaths discovers 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.

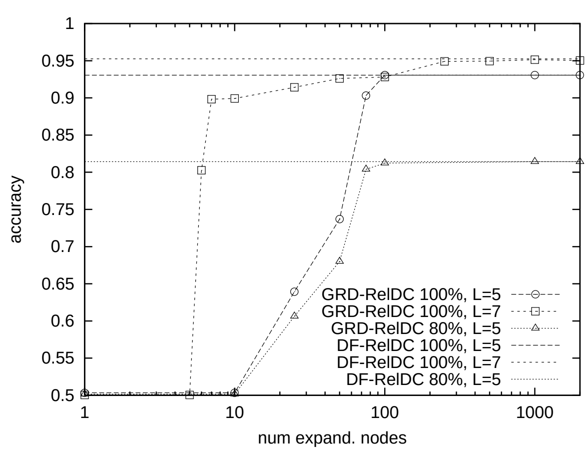

Figure 35: SynPub1: $unc_1 = 2$. Accuracy of GRD-RelDC vs. DF-RelDC.

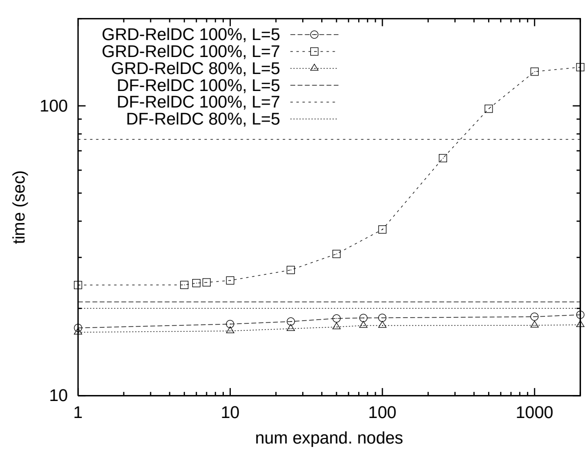

Figure 36: SynPub1: $unc_1 = 2$. Time of GRD-RelDC vs. DF-RelDC.

Figures 35 and 36 study the effect of N_{exp} parameter on the accuracy and efficiency of GRD-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 GRD-RelDC. The AllPaths part of GRD-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 vs. GRD, (2) path length limit $L = 5$ and $L = 7$ and (3) the amount of affiliation 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 GRD-RelDC and then see if the figures corroborate it. We will always assume that **path length is limited** for both DF-RelDC and GRD-RelDC.

If N_{exp} is small then GRD-RelDC should discover only a few paths and its accuracy should be close to that of FBS. If N_{exp} is sufficiently large, then GRD-RelDC should discover the same paths as DF-RelDC. That is, we can compute $m_{ikj} = |\mathcal{P}_L(x_i, y_{ikj})|$, where $|\mathcal{P}_L(x_i, y_{ikj})|$ is the number of paths in $\mathcal{P}_L(x_i, y_{ikj})$. Then if we choose N_{exp} such that $N_{exp} \geq m$, where $m = \max_{i,k,j}(m_{ikj})$, then the set of all paths that GRD-RelDC will discover will be identical to that of DF-RelDC. Thus the accuracy of GRD-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 GRD-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 $L = 5$, GRD-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 GRD-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.

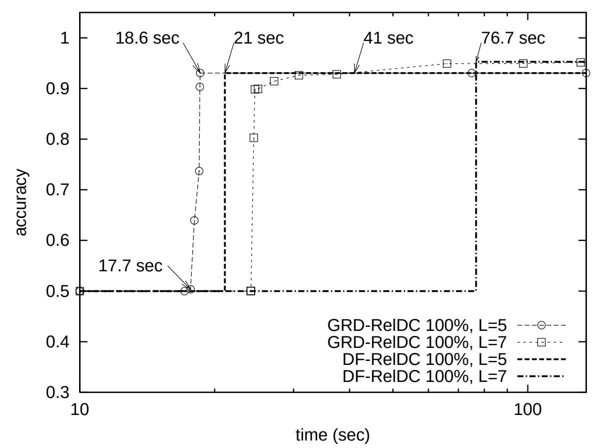

Figure 37: SynPub1: $unc_1 = 2$. Choosing the best among GRD-RelDC $L = 5$, GRD-RelDC $L = 7$, DF-RelDC $L = 5$, DF-RelDC $L = 7$ at each moment in time.

Figure 37 combines Figures 35 and 36. It plots the achieved accuracy by DF- and GRD-RelDC 100% 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 GRD-RelDC with $L = 5$. If one is allowed to spend only $[18.6, 41)$ seconds, it is better to use GRD-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 GRD-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 [47] which has been made popular by the textbook [20]. 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 the RealPub data set. 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*.

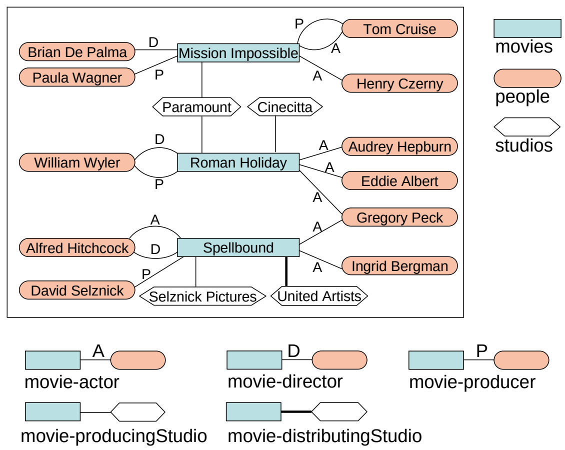

Figure 38: Sample entity-relationship graph for *movies* dataset.

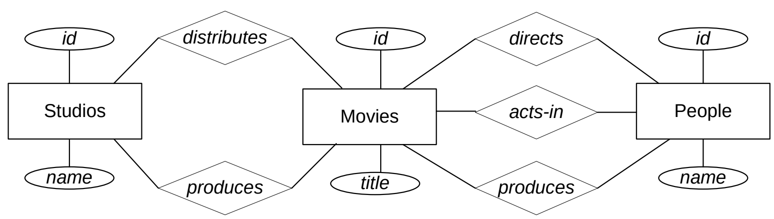

Figure 39: E/R diagram for RealMov.

Stage name	DOW	Name at birth	Gen	DOB	Role	Orig	Notes
Tom Cruise	1981-1989	Thomas Cruise Mapother IV	M	1962	Hero	Am	
B.dePalma	1968-1987	Brian De Palma	M				
Paula Wagner			F			Am	
Henry Czerny			M	1959		Ca	
Wyler	1925-1959	William Wyler		1902		Am	Or(Ge)
Audrey Hepburn	1951-1981	Audrey Hepburn-Ruston	F	1929	pert	Be	
Eddie Albert	1938-1982	Eddie Albert Heimberger	M	1908	honest Joe	Am	
Gregory Peck	1943-1982	Gregory Peck	M	1916	likeable	Am	
Ingrid Bergman	1934-1978	Ingrid Bergman	F	1915	strong beauty	Sw	
Hitchcock	1925-1976	Alfred Hitchcock		1899		Br	Ty(Susp , Nior)
Selznick		David O. Selznick		1902		Ru	Ww(Hitchcock)

Table 4: The *people* table. Some of the notation used: DOW (dates of work), Gen (gender), DOB (date of birth), type (kinds of roles actor played), orig (origin), Am (America).

Figure 38 presents a sample graph for RealMov dataset. Tables 4, 5, 6, and 7 demonstrate sample content of the *people*, *movies*, *studios* and *cast* tables derived from the movies dataset. The sample graph in Figure 38 is constructed from those tables.

ID	Title	Year	Director	Producer	Studio	Color	Genre
BdP30	Mission Impossible	1996	B.dePalma	Tom Cruise, Paula Wagner	Paramount	col	Action
WW67	Roman Holiday	1953	Wyler	Wyler	Cinecitta, Paramount	bnw	Romantic
H42	Spellbound	1945	Hitchcock	Selznick	Selznick Pictures	bnw	Suspect

Table 5: The *movies* table.

Name	Full name	City	Country	First	Last	Founder	Successor
Paramount	Paramount Corp.	Los Angeles	USA	1916	1993	W. Hodkinson	Paramount, Viacom
Cinecitta		Rome	Italy	1937			
Selznick	Selznick Pictures	Hollywood	USA	1936	1944		
U.A.	United Artists	Hollywood	USA	1919	1983	Chaplin, Pickford, etc.	MGM-UA

Table 6: The *studios* table.

Movie ID	Actor name
BdP30	Tom Cruise
BdP30	B.dePalma
BdP30	Paula Wagner
BdP30	Henry Czerny
WW67	Wyler
WW67	Audrey Hepburn
WW67	Eddie Albert
WW67	Gregory Peck
H42	Gregory Peck
H42	Ingrid Bergman
H42	Hitchcock
H42	Selznick

Table 7: The *cast* table.

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.

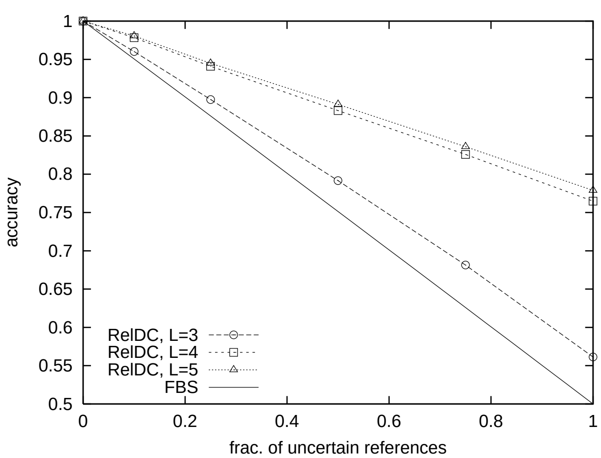

Figure 40: RealMov: disambiguating *director* references. The size of the choice set of each *uncertain* reference is 2.

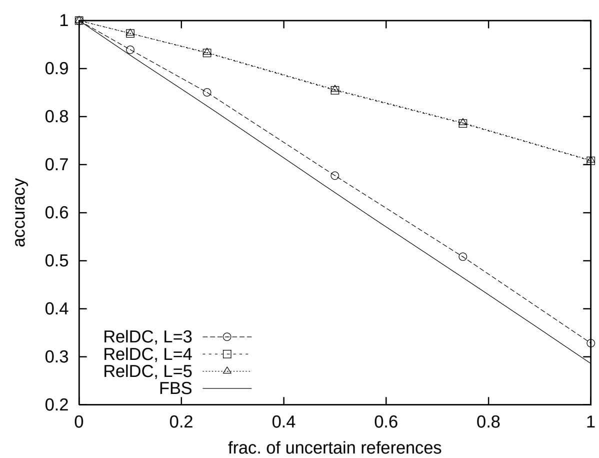

Figure 41: RealMov: disambiguating *director* references. The pmf of sizes of choice sets of *uncertain* references is given in Figure 42.

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 used by other researchers, e.g. [8]. Given the specifics of our problem,

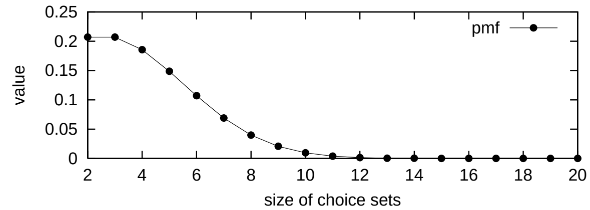

Figure 42: PMF of sizes of choice sets.

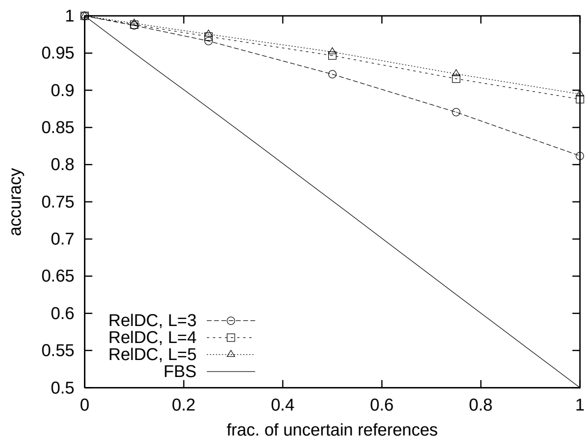

Figure 43: RealMov: disambiguating *studio* references. The size of the choice set of each *uncertain* reference is 2.

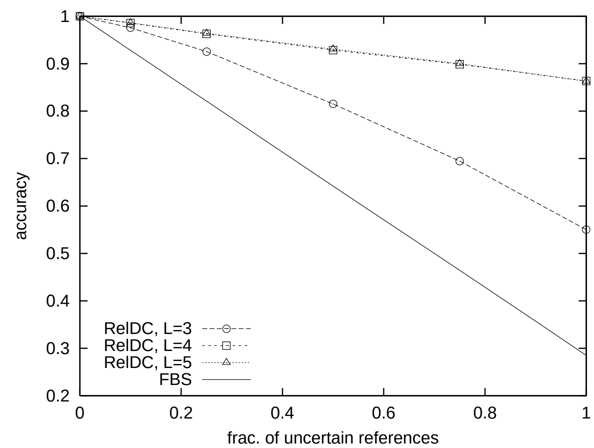

Figure 44: RealMov: disambiguating *studio* references. The pmf of sizes of choice sets of *uncertain* references is given in Figure 42.

to study the accuracy of RelDC we will simulate that we used FBS to determine the choice 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 choice 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 choice 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 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

[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.

of disambiguating *studio* references is even higher. □

7 Related Work

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

Researchers have also explored domain-independent techniques, e.g. [39, 19, 24, 5, 36]. Their work can be viewed as addressing two challenges: (1) improving similarity function, as in [6]; and (2) improving efficiency of linkage, as in [8]. Typically two-level similarity functions are employed to compare two records. First, such a function computes attribute-level similarities by comparing values in the same attributes of two records. Next the function combines the attribute-level similarity measures to compute the overall similarity of two records. A recent trend has been to employ machine learning techniques, e.g. SVM, to learn the best similarity function for a given domain [6]. Many techniques have been proposed to address the efficiency challenge as well: e.g. using specialized indexes [8], sortings, etc.

Those domain-independent techniques deal only with attributes. To the best of our knowledge, RelDC, which was first publicly released in [28], is the first domain-independent data cleaning framework which exploits relationships for cleaning. Recently, in parallel to our work, other researchers have also proposed using relationships for cleaning. In [5] Ananthakrishna et al. employ similarity of directly linked entities, for the case of hierarchical relationships, to solve the record de-duplication challenge. In [32] Lee et al. develop an association-rules mining based method to disambiguate references using similarity of the context attributes: the proposed technique is still an FBS method, but [32] also discusses concept hierarchies which are related to relationships. Getoor et al. in DKDM04 use similarity of attributes of directly linked objects, like in [5], for the purpose of object consolidation. However, the challenge of applying that technique in practice on real-world datasets was identified as future work in that paper. In contrast to the above described techniques, RelDC utilize the CAP principle to automatically discover and analyze relationship chains, thereby establishing a framework that employs systematic relationship analysis for the purpose of cleaning.

8 Conclusion

In this paper we have shown that analysis of inter-object relationships allows to significantly improve the quality of reference disambiguation. We have developed a domain-independent approach, called RelDC, that 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 exists 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 dataset. Extensive empirical analysis on real and synthetic data sets shows that RelDC improves the quality of reference disambiguation beyond the traditional techniques.

As future work we plan to apply similar techniques of relationship analysis to the problem of record linkage. Another research direction is to develop an approach which, given a sample resolved graph, would automatically determine which relationships are irrelevant for a particular disambiguation task [30]. Such an approach would learn the importance of a particular relationship type (and a relationship chain) directly from the data. Yet another direction is to solve the reference disambiguation problem but in different settings. For example, in our publications dataset the set of all authors was available. However, if such a set is not available, the task becomes to not only resolve references but also determine the correct author set.

References

[1] CiteSeer. `http://citeseer.nj.nec.com/cs`.

[2] GAMS/SNOPT solver. `http://www.gams.com/solvers/`.

[3] HomePageSearch. `http://hpsearch.uni-trier.de`.

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

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

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

[7] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *Proc of International World Wide Web Conference*, 1998.

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

[9] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *Proc. of ACM SIGMOD International Conference on Management of Data (ACM SIGMOD'03)*, San Diego, CA, USA, June 9–12 2003.

[10] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Querying imprecise data in moving object environments. *IEEE Transactions on Knowledge and Data Engineering (IEEE TKDE)*, 16(9), Sept. 2004.

[11] R. Cheng, S. Prabhakar, and D. V. Kalashnikov. Querying imprecise data in moving object environments. In *Proc. of the 19th IEEE International Conference on Data Engineering (IEEE ICDE'03)*, Bangalore, India, March 5–8 2003.

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

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

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

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

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

[17] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT Press, 2001.

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

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

[20] H. Garcia-Molina, J. D. Ullman, and J. Widom. *Database systems: the complete book*. Prentice Hall, 2002.

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

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

[23] G. Grimmett and D. Stirzaker. *Probability and random processes*. OXFORD University Press, 2002.

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

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

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

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

[28] D. Kalashnikov and Mehrotra. Exploiting relationships for data cleaning. *TR-RESCUE-03-02*, Nov. 2003.

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

[30] D. V. Kalashnikov and S. Mehrotra. Learning importance of relationships for reference disambiguation. *Submitted for Publication*, Dec. 2004. `http://www.ics.uci.edu/~dvk/RelDC/TR/TR-RESCUE-04-23.pdf`.

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

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

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

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

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

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

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

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

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

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

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

[42] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *Proc. of ACM SIGKDD Conf.*, 2002.

[43] D. Seid and S. Mehrotra. Complex analytical queries over large attributed graph data. *Submitted for Publication*, 2005.

[44] S. Tejada, C. A. Knoblock, and S. Minton. Learning domain-independent string transformation weights for high accuracy object identification. In *Proc. of ACM SIGKDD Conf.*, 2002.

[45] V. Verykios, G.V.Moustakides, and M. Elfeky. A bayesian decision model for cost optimal record matching. *The VLDB Journal*, 12:28–40, 2003.

[46] S. White and P. Smyth. Algorithms for estimating relative importance in networks. In *Proc. of ACM SIGKDD Conf.*, 2003.

[47] G. Wiederhold. The movies dataset. `http://www-db.stanford.edu/pub/movies/doc.html`.

[48] W. Winkler. The state of record linkage and current research problems. In *U.S. Bureau of Census, TR99.*

[49] W. E. Winkler. Advanced methods for record linkage. In *U.S. Bureau of Census*, 1994.

Appendix

A Probabilistic model for computing connection strength

In Section 4.1 we have presented the weight based model (WM) for computing connection strength. In this section 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.

Notation	Meaning
$x^{\exists}$	event "x exists" for (edge,path) x
$x^{\nexists}$	event "x does not exist" for (edge,path) x
$x^{\rightarrow}$	event corresponding to following (edge,path) x
$dep(e_1, e_2)$	if events e_1 and e_2 are independent, then $dep(e_1, e_2) = \texttt{true}$, else $dep(e_1, e_2) = \texttt{false}$
$\mathrm{P}(x^{\exists})$	probability that (edge,path) x exists
$\mathrm{P}(x^{\rightarrow})$	probability to follow (edge,path) 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 and only if edge E_{ij} exists; otherwise $a_{ij} = 0$
$a_{i0} = 1$	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 $opt(E) = \texttt{false}$
v^*_E	if edge E is an option-edge, then v^*_E denotes the choice node associated with E
$\mathbf{a}$, as a vector	$\mathbf{a} = (a_{10}, a_{11}, \ldots, a_{(k-1)n_{k-1}})$
$\mathbf{a}$, as a set	$\mathbf{a} = \{a_{ij} : i = 1, 2, \ldots, k-1; j = 0, 1, \ldots, n_i\}$
$\mathbf{a}$, as a variable	at each moment variable $\mathbf{a}$ is one instantiation of $\mathbf{a}$ as a vector

Table 8: Probabilistic model: Terminology

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 edge E. Similarly, we use $E^{\nexists}$ for event "E does not exist". So, $\mathrm{P}(E^{\exists})$ refers to the probability that E exists. We will consider situations where the algorithm computes the probability to follow (or 'go via') 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 8 summarizes the notation.

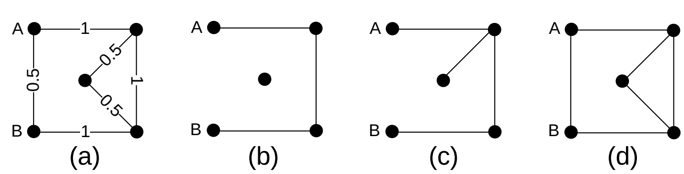

Figure 45: Probabilistic graph maps to a family of regular graphs.

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 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 $\mathscr{P} = A \to B \to C \to D \to E$ which will be of interest to us.

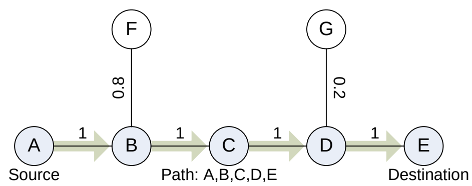

Figure 46: Toy example: independent case

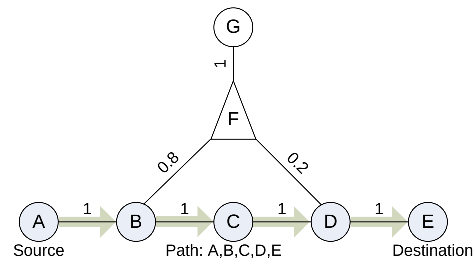

Figure 47: Toy example: dependent case

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 *dependent*): if one edge is present 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 to follow path $\mathscr{P}$: $c(\mathscr{P}) = \mathrm{P}(\mathscr{P}^{\to})$. 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 $\mathrm{P}(\mathscr{P}^{\to}|\mathscr{P}^{\exists})$ to follow $\mathscr{P}$ given that $\mathscr{P}$ exists. Then PM computes $c(\mathscr{P})$ as $c(\mathscr{P}) = \mathrm{P}(\mathscr{P}^{\to}) = \mathrm{P}(\mathscr{P}^{\to}|\mathscr{P}^{\exists})\mathrm{P}(\mathscr{P}^{\exists})$.

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, e.g. 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 $\mathrm{P}(\mathscr{P}^{\to}|\mathscr{P}^{\exists})$ to follow path $\mathscr{P}$, given that $\mathscr{P}$ exists. Once this probability is computed, we can compute $c(p)$ as $c(\mathscr{P}) = \mathrm{P}(\mathscr{P}^{\to}) = \mathrm{P}(\mathscr{P}^{\exists})\mathrm{P}(\mathscr{P}^{\to}|\mathscr{P}^{\exists})$. The probability $\mathrm{P}(\mathscr{P}^{\to}|\mathscr{P}^{\exists})$ 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 to follow path $\mathscr{P}$ is the product of probabilities to follow each of the edges on the path: $\mathrm{P}(\mathscr{P}^{\to}|\mathscr{P}^{\exists}) = \mathrm{P}(AB^{\to}|\mathscr{P}^{\exists})\mathrm{P}(BC^{\to}|\mathscr{P}^{\exists})\mathrm{P}(CD^{\to}|\mathscr{P}^{\exists})\mathrm{P}(DE^{\to}|\mathscr{P}^{\exists})$. Given path $\mathscr{P}$ exists, the probability to follow edge AB in path $\mathscr{P}$ is one. The probability to follow edge BC is computed as follows. With probability 0.2 edge BF is absent, in which case the probability to follow BC is 1. With probability 0.8 edge BF is present, in which case the probability to follow BC is $\frac{1}{2}$ – because there are two links, BF and BC, that can be followed. Thus the total probability to follow BC

is $0.2 \cdot 1 + 0.8 \cdot \frac{1}{2} = 0.6$. Similarly, the probability to follow CD is 1 and the probability to follow DE is $0.8 \cdot 1 + 0.2 \cdot \frac{1}{2} = 0.9$. The probability to follow path $\mathscr{P}$, given it exists, is the product of probabilities to follow 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 to follow $\mathscr{P}$ 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 $\mathrm{P}(\mathscr{P}^{\rightarrow}|\mathscr{P}^{\exists}) = \mathrm{P}(BF^{\exists}|\mathscr{P}^{\exists})\mathrm{P}(\mathscr{P}^{\rightarrow}|\mathscr{P}^{\exists} \cap BF^{\exists}) + \mathrm{P}(BF^{\nexists}|\mathscr{P}^{\exists})\mathrm{P}(\mathscr{P}^{\rightarrow}|\mathscr{P}^{\exists} \cap BF^{\nexists})$.

Let us first assume $BF^{\exists}$ (i.e., edge BF is present) and then compute $\mathrm{P}(BF^{\exists}|\mathscr{P}^{\exists})\mathrm{P}(\mathscr{P}^{\rightarrow}|\mathscr{P}^{\exists} \cap BF^{\exists})$. For the case of Figure 47, if no assumptions about the presence or absence of DF have been made yet, $\mathrm{P}(BF^{\exists}|\mathscr{P}^{\exists})$ 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 to follow $\mathscr{P}$ 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 $\mathrm{P}(BF^{\nexists}|\mathscr{P}^{\exists})$ 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 $\mathrm{P}(\mathscr{P}^{\rightarrow}|\mathscr{P}^{\exists}) = 0.8 \cdot \frac{1}{2} + 0.2 \cdot \frac{1}{2} = 0.5$. So $c(\mathscr{P}) = \mathrm{P}(\mathscr{P}^{\rightarrow}) = 0.50$, 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

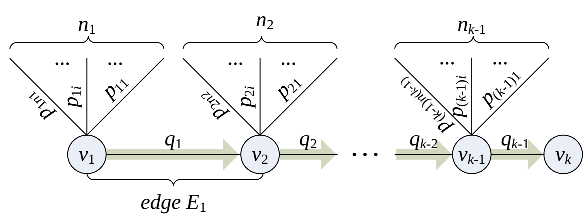

Figure 48: Independent edge existence. Computing $c(v_1 \to v_2 \to \cdots \to v_k)$. All edges shown in the figure are "possible to follow" edges in the context of the path. Edges that are not possible to follow are not shown.

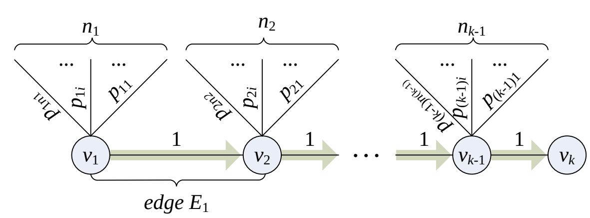

Figure 49: The case in this figure is similar to that of Figure 48 with an additional assumption that path $\mathscr{P}$ exists.

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 $\mathrm{P}(E_i^{\exists}) = q_i$ $(i = 1, 2, \ldots, k-1)$. We will refer to edges labeled with probabilities p_{ij} (for all i, j) in this figure as E_{ij}. The goal is to compute the probability to follow path $\mathscr{P}$, which is the measure of the connection strength of path $\mathscr{P}$:

$$c(\mathscr{P}) = \mathrm{P}(\mathscr{P}^{\rightarrow}) = \mathrm{P}(\mathscr{P}^{\exists})\mathrm{P}(\mathscr{P}^{\rightarrow}|\mathscr{P}^{\exists}). \tag{8}$$

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

$$\mathrm{P}(\mathscr{P}^{\exists}) = \mathrm{P}(\bigcap_{i=1}^{k-1} E_i^{\exists}). \tag{9}$$

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

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

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 $\widetilde{q}_i = \mathrm{P}(E_i^{\exists}|\mathscr{P}^{\exists}) = 1$ and $\widetilde{p}_{ij} = \mathrm{P}(E_{ij}^{\exists}|\mathscr{P}^{\exists})$. 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}: $a_{i0} = 1$ and $p_{i0} = 1$ $(i = 1, 2, \ldots, k-1)$.[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

$$\mathrm{P}(\mathscr{P}^{\rightarrow}|\mathscr{P}^{\exists}) = \sum_{\mathbf{a}\in\mathcal{A}} \left\{ \mathrm{P}(\mathscr{P}^{\rightarrow}|\mathbf{a} \cap \mathscr{P}^{\exists}) \mathrm{P}(\mathbf{a}|\mathscr{P}^{\exists}) \right\}, \tag{11}$$

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

$$\mathrm{P}(\mathbf{a}|\mathscr{P}^{\exists}) = \mathrm{P}(\mathbf{a}) = \prod_{\substack{i=1,2,\ldots,k-1 \\ j=0,1,\ldots,n_i}} p_{ij}^{a_{ij}} (1-p_{ij})^{1-a_{ij}}. \tag{12}$$

Probability $\mathrm{P}(\mathscr{P}^{\rightarrow}|\mathbf{a} \cap \mathscr{P}^{\exists})$, which is the probability to go via $\mathscr{P}$ given (1) a particular instantiation of $\mathbf{a}$; and (2) the fact that $\mathscr{P}$ exists, can be computed as

$$\mathrm{P}(\mathscr{P}^{\rightarrow}|\mathbf{a} \cap \mathscr{P}^{\exists}) = \prod_{i=1}^{k-1} \frac{1}{1 + \sum_{j=1}^{n_i} a_{ij}} \equiv \prod_{i=1}^{k-1} \frac{1}{\sum_{j=0}^{n_i} a_{ij}}. \tag{13}$$

Thus

$$\mathrm{P}(\mathscr{P}^{\rightarrow}) = \left(\prod_{i=1}^{k-1} q_i \right) \left(\sum_{\mathbf{a}\in\mathcal{A}} \left\{ \left[\prod_{i=1}^{k-1} \frac{1}{\sum_{j=0}^{n_i} a_{ij}} \right] \left[\prod_{ij} p_{ij}^{a_{ij}} (1-p_{ij})^{1-a_{ij}} \right] \right\} \right). \tag{14}$$

A.2.2 Computing path connection strength in practice

Notice, Equation (14) iterates through all possible instantiations of $\mathbf{a}$ 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 $\mathbf{P}(\mathscr{P}^{\rightarrow}|\mathscr{P}^{\exists})$ as $\prod_{i=1}^{k-1} \mathbf{P}(E_i^{\rightarrow}|\mathscr{P}^{\exists})$. To achieve the simplification, we will use our assumption of independence of probabilities which allows us to compute $\mathrm{P}(\mathscr{P}^{\rightarrow}|\mathscr{P}^{\exists})$ as the product of the probabilities to follow each individual edge in the path:

$$\mathrm{P}(\mathscr{P}^{\rightarrow}|\mathscr{P}^{\exists}) = \prod_{i=1}^{k-1} \mathrm{P}(E_i^{\rightarrow}|\mathscr{P}^{\exists}). \tag{15}$$

[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 $p_{i0} = \mathrm{P}(E_i^{\exists}|\mathscr{P}^{\exists}) = 1$.

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

$$\mathrm{P}(E_i^{\rightarrow}|\mathscr{P}^{\exists}) = \sum_{\mathbf{a}_i \in \mathcal{A}_i} \left\{ \left[\frac{1}{\sum_{j=0}^{n_i} a_{ij}} \right] \left[\prod_{j=0}^{n_i} p_{ij}^{a_{ij}} (1-p_{ij})^{1-a_{ij}} \right] \right\}. \tag{16}$$

Combining Equations (8), (15) and (16) we have

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

The effect of transformation. Notice, using Equation (14) the algorithm will need to perform $|\mathcal{A}| = 2^{n_1+n_2+\cdots+n_{k-1}}$ iterations – one per each instantiation of $\mathbf{a}$. Using Equation (17) 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 Equation (16) assumes 2^{n_i} iterations will be needed to compute $\mathrm{P}(E_i^{\rightarrow}|\mathscr{P}^{\exists})$.

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 $\tilde{\mathbf{a}}_i$ 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 Equation (16) can be simplified to

$$\mathrm{P}(E_i^{\rightarrow}|\mathscr{P}^{\exists}) = \sum_{\tilde{\mathbf{a}}_i \in \widetilde{\mathcal{A}}_i} \left\{ \left[\frac{1}{m + \sum_{j=0}^{n_i-m} a_{ij}} \right] \left[\prod_{j=0}^{n_i-m} p_{ij}^{a_{ij}} (1-p_{ij})^{1-a_{ij}} \right] \right\}. \tag{18}$$

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

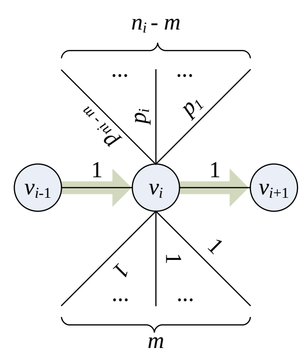

Figure 50: Probability to follow edge $E_i = (v_i, v_{i+1})$

Computing $\mathbf{P}(E_i^{\rightarrow}|\mathscr{P}^{\exists})$ as $\sum_{l=0}^{n_i} \frac{1}{1+l} \mathbf{P}(s_i = l)$. 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. But 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 [23]. 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, i.e. the probability of success in i-th trial is p_i. We can modify Equation (17) to compute $\mathrm{P}(E_i^{\rightarrow}|\mathscr{P}^{\exists})$ as follows:

$$\mathrm{P}(E_i^{\rightarrow}|\mathscr{P}^{\exists}) = \sum_{l=0}^{n_i} \frac{1}{1+l} \mathrm{P}(s_i = l). \tag{19}$$

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* $\mathrm{P}(s_i = l)$ *quickly*, rather than compute it exactly via iterating over all cases when $(s_i = l)$. That is, we would like to avoid computing $\mathrm{P}(s_i = l)$ as

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

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

$$\mathrm{P}(s_i = l) = \frac{\lambda^l e^{-\lambda}}{l!} \left\{ 1 + O\left(\lambda \max_{j=1,2,\ldots,n_i} p_{ij} + \frac{l^2}{\lambda} \max_{j=1,2,\ldots,n_i} p_{ij} \right) \right\}, \text{ where } \lambda = \sum_{j=1}^{n_i} p_{ij}. \tag{20}$$

One can also utilize the following "Monte-Carlo like" method to compute $\mathrm{P}(s_i = l)$. 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 $\mathrm{P}(s_i = l)$ 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 $\mathrm{P}(E_i^{\rightarrow} | \mathscr{P}^{\exists})$ precisely, using iterations. If it is not acceptable (typically, rare case), then try to use Equation (20). 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 $\mathrm{P}(E_{ij}^{\exists}) = p_{ij}$, as follows: $\mu_i = m + \sum_{j=1}^{n_i - m} p_{ij}$. Then since there are $1 + \mu_i$ possible links to follow on average, the probability to follow E_i can be coarsely estimated as

$$\mathrm{P}(E_i^{\rightarrow} | \mathscr{P}^{\exists}) \approx \frac{1}{1 + \mu_i} = \frac{1}{m + \sum_{j=0}^{n_i - m} p_{ij}}. \tag{21}$$

A.3 Dependent edge existence

In this section we discuss how to compute connection strength if occurrence of edges is not independent. 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 $\mathrm{P}(\mathscr{P}^{\rightarrow})$ for those cases.

There are two principal situations we need to address. 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 sub-cases 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 $\mathrm{P}(\mathscr{P}^{\rightarrow}) = \mathrm{P}(\mathscr{P}^{\exists})\mathrm{P}(\mathscr{P}^{\rightarrow} | \mathscr{P}^{\exists})$. When we compute $\mathrm{P}(\mathscr{P}^{\exists})$ 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 Equation (17) to compute the connection strength.

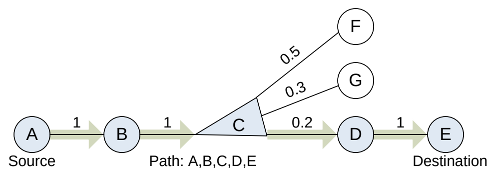

Figure 51: Choice node on the path

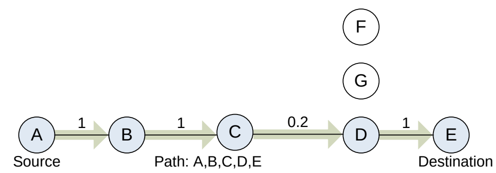

Figure 52: Choice node on the path: removing choice

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 $B \to C \to D$ 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 we have applied the procedure from Section A.3.1 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

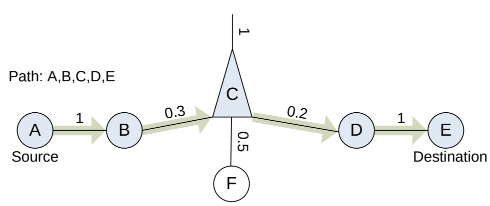

Figure 53: Choice node on the path: illegal path

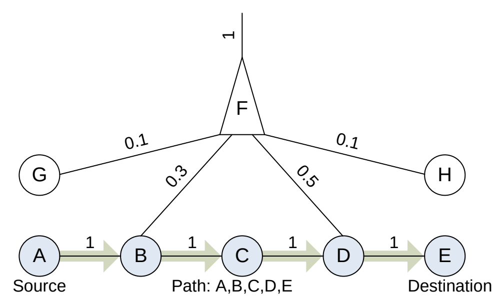

Figure 54: Options of the same choice node on the path

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 Equation (17), but for the general "dependent" case.

Let us define two sets $\mathbf{f}$ and $\mathbf{d}$ – of 'free' and 'dependent' a_{ij}'s as follows:

$$\begin{aligned} \mathbf{f} &= \{a_{ij} : \forall r, s\ (r \neq i \text{ or } s \neq j) \Rightarrow dep(E^{\exists}_{ij}, E^{\exists}_{rs}) = \texttt{false}\}, \\ \mathbf{d} &= \{a_{ij} : \exists r, s\ (r \neq i \text{ or } s \neq j) : dep(E^{\exists}_{ij}, E^{\exists}_{rs}) = \texttt{true}\}. \end{aligned} \tag{22}$$

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 Equation (17), otherwise we proceed as follows. Similarly to $\mathbf{a}_i$ we can define $\mathbf{f}_i$ and $\mathbf{d}_i$ as follows:

$$\begin{aligned} \mathbf{f}_i &= \{a_{ij} : a_{ij} \in \mathbf{f}, j = 0, 1, \dots, n_i\}, \\ \mathbf{d}_i &= \{a_{ij} : a_{ij} \in \mathbf{d}, j = 1, 2, \dots, n_i\}. \end{aligned} \tag{23}$$

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

$$\mathrm{P}(\mathscr{P}^{\rightarrow}) = \underbrace{\left(\prod_{i=1}^{k-1} q_i\right)}_{\mathrm{P}(\mathscr{P}^{\exists})} \sum_{\mathbf{d}\in\mathcal{D}} \left\{ \underbrace{\left[\prod_{i=1}^{k-1}\left(\sum_{\mathbf{f}_i\in\mathcal{F}_i}\left\{\left[\frac{1}{\sum_{j=0}^{n_i} a_{ij}}\right]\left[\prod_{j:a_{ij}\in\mathbf{f}_i} p_{ij}^{a_{ij}}(1-p_{ij})^{1-a_{ij}}\right]\right\}\right)\right]}_{\Psi(\mathbf{d})} \mathrm{P}(\mathbf{d}) \right\}. \tag{24}$$

Equation (24) iterates over all feasible instantiations of $\mathbf{d}$. $\mathrm{P}(\mathbf{d})$ is the probability of a specific instance of $\mathbf{d}$ to occur. Equation (24) contains term $\sum_{\mathbf{d}\in\mathcal{D}}\{\Psi(\mathbf{d})\mathrm{P}(\mathbf{d})\}$. What this achieves is that a particular instantiation of $\mathbf{d}$ "fixates" a particular combination of all "dependent" edges, and $\mathrm{P}(\mathbf{d})$ corresponds to the probability of that combination. Notice, $\Psi(\mathbf{d})$ directly corresponds to $\mathrm{P}(\mathscr{P}^{\rightarrow}|\mathscr{P}^{\exists})$ part of Equation (17). To compute $\mathrm{P}(\mathscr{P}^{\rightarrow})$ in Equation (24), we only need to specify how to compute $\mathrm{P}(\mathbf{d})$.

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

In other words, we can split set $\mathbf{d}$ into non intersecting 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 $\mathbf{d}_{C_l}$ is a particular instantiation of a_{ij}'s from C_l. Now, to be able to compute Equation (24), we only need to specify how to compute $P(\mathbf{d}_{C_l})$ $(l = 1, 2, \ldots, m)$.

Computing $P(\mathbf{d}_{C_l})$. Figure 55 shows choice node v_l^* with n options $u_1, u_2, \ldots, u_n$. Each (v_l^*, u_j) edge $(j = 1, 2, \ldots, n)$ is labeled with probability p_j. As before, to specify which edge is present and which is

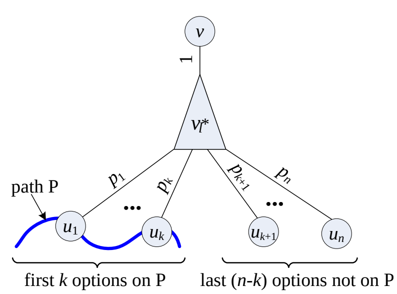

Figure 55: Intra choice dependence.

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 v_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 $P(\mathbf{d}_{C_l})$ 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 $a_1, a_2, \dots, a_k$ is 1. Assume $a_j = 1$ $(1 \le j \le k)$, then $\mathrm{P}(a_j = 1) = p_j$. To summarize:

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

Now we know how to compute $P(\mathbf{d}_{C_l})$ $(l = 1, 2, \dots, m)$, thus we can compute $\mathrm{P}(\mathbf{d})$. Therefore we have specified how to compute path connection strength using Equation (24).

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. This weight will be treated as the probability of the edge to exist.

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 $V = \{v_i : i = 1, 2, \dots, |V|\}$ and E is the set of edges $E = \{E_i : i = 1, 2, \dots, |E|\}$. Let us define a_i as follows: $a_i = 1$ 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 to reach 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

$$\mathrm{P}(\mathcal{G}(u, v)^{\rightarrow}) = \sum_{\mathbf{a} \in \mathcal{A}} \mathrm{P}(\mathcal{G}(u, v)^{\rightarrow} | \mathbf{a}) \mathrm{P}(\mathbf{a}). \tag{25}$$

Notice, when computing $\mathrm{P}(\mathcal{G}(u, v)^{\rightarrow} | \mathbf{a})$ we assume a particular instantiation of $\mathbf{a}$. So 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

$$\mathrm{P}(\mathcal{G}(u, v)^{\rightarrow} | \mathbf{a}) = \sum_{i=1}^{k} \mathrm{P}(\mathscr{P}_i^{\rightarrow} | \mathbf{a}), \tag{26}$$

and

$$\begin{aligned} \mathrm{P}(\mathcal{G}(u, v)^{\rightarrow}) &= \sum_{\mathbf{a} \in \mathcal{A}} \mathrm{P}(\mathcal{G}(u, v)^{\rightarrow} | \mathbf{a}) \mathrm{P}(\mathbf{a}) \\ &= \sum_{\mathbf{a} \in \mathcal{A}} \left[\left(\sum_{i=1}^{k} \mathrm{P}(\mathscr{P}_i^{\rightarrow} | \mathbf{a}) \right) \mathrm{P}(\mathbf{a}) \right] \\ &= \sum_{i=1}^{k} \left[\sum_{\mathbf{a} \in \mathcal{A}} \Big(\mathrm{P}(\mathscr{P}_i^{\rightarrow} | \mathbf{a}) \mathrm{P}(\mathbf{a}) \Big) \right] \\ &= \sum_{i=1}^{k} \mathrm{P}(\mathscr{P}_i^{\rightarrow}). \end{aligned} \tag{27}$$

Equation (27) shows that the total connection strength is the sum of the connection strength of all L-short simple paths.

B Solving the NLP problem by bounding option weights

The iterative technique for solving (7) from Section 4.3 works well in practice, but it is not the only approximate method to solve (7). Next we sketch another interesting method of solving it.

Let us motive the technique with the help of an example. Consider Equations (5) for the toy database. The savvy reader could have noticed that for this particular example it is actually possible to skip the step of computing the answer to (5) and proceed to interpreting weights as follows. Consider the numerator and denominator of equation $w_1 = \frac{w_3}{2} / \left(1 + \frac{w_3}{2}\right)$. Given that all weights are real numbers from $[0, 1]$ interval, we can bound the numerator $\frac{w_3}{2}$: $\frac{w_3}{2} \in \left[0, \frac{1}{2}\right]$. Similarly, we can bound the denominator $\left(1 + \frac{w_3}{2}\right) \in \left[1, 1\frac{1}{2}\right]$. Therefore, we can assign to w_1 the lower bound $w_1^{\vdash} = 0/\left(1\frac{1}{2}\right) = 0$, and the upper bound $w_1^{\dashv} = \frac{1}{2}/1 = \frac{1}{2}$. So, $w_1 \in \left[w_1^{\vdash}, w_1^{\dashv}\right] = \left[0, \frac{1}{2}\right]$. Similarly we can compute that $w_2 \in \left[\frac{2}{3}, 1\right]$, $w_3 \in \left[0, \frac{1}{2}\right]$, and $w_4 \in \left[\frac{2}{3}, 1\right]$. Thus, knowing these bounds, it is possible to determine even without solving (5) that if (5) has a solution, then this solution is such that $w_1 < w_2$ and $w_3 < w_4$. For our toy example the later information is sufficient to interpret weights: it will lead us to the same conclusion as if the system was solved exactly – that both 'D. White' references refer to 'Don White'.

In general, we can quickly compute the bounding interval $\left[w_{ikj}^{\vdash}, w_{ikj}^{\dashv}\right]$ for each w_{ikj}, e.g. by analyzing numerators and denominators.[15] The bounding interval of each w_{ikj} is, in general, determined by other option weights, so it might shrink later – for example, if we at some later point determine the exact values of some of those option weights. Let us consider some reference r_{ik} and bounds $\left[w_1^{\vdash}, w_1^{\dashv}\right], \left[w_2^{\vdash}, w_2^{\dashv}\right], \ldots, \left[w_N^{\vdash}, w_N^{\dashv}\right]$ of weights $w_1, w_2, \ldots, w_N$ of its option edges, see Figure 4. Clearly, if there exists $w_j^{\vdash}$ such that $w_j^{\vdash} > w_l^{\dashv}$ $(l = 1, 2, \ldots, N, l \neq j)$, then w_j is guaranteed to be greater than any w_l $(l = 1, 2, \ldots, N, l \neq j)$ and consequently the weight-interpreting procedure is guaranteed to pick y_j as r_{ik}^*. So, given this fact, we can compute the following set of (yet unresolved) references

$$R = \left\{r_{ik} : \exists w_{ikj} \text{ such that } w_{ikj}^{\vdash} > w_{ikl}^{\dashv} \;\; (l = 1, 2, \ldots, N, l \neq j)\right\}$$

Notice, we know how to resolve each reference in this set. So, we resolve each reference $r_{ik} \in R$ to the corresponding y_j and assign weight of 1 to w_j and weight of 0 to w_l $(l = 1, 2, \ldots, N, l \neq j)$. This assignment of values to option weights w_{ikj}'s can shrink bounding intervals of the other option weights, so that we can recompute the set R again and apply the procedure again until either all references are resolved or R is the empty set.

The above procedure can still leave some of the references unresolved due to overlap of bounding intervals. The procedure was useful for providing intuition behind the generic method of solving references using bounding intervals which we will describe next.

Recall that resolving reference r_{ik} translates into determining which w_j among $w_1, w_2, \ldots, w_N$ has the maximum value. To achieve this goal notice that for each option weight w_j we can determine its bounds $\left[w_j^{\vdash}, w_j^{\dashv}\right]$. We can treat each w_j as a random variable with its probability density function (pdf) defined on $\left[w_j^{\vdash}, w_j^{\dashv}\right]$. To be concrete, let us assume that each w_j $(j = 1, 2, \ldots, N)$ is uniformly distributed on $\left[w_j^{\vdash}, w_j^{\dashv}\right]$. Now, we can compute the *probability* p_j that given w_j $(j = 1, 2, \ldots, N)$ has the maximum value, among $w_1, w_2, \ldots, w_N$, given their bounding intervals and pdfs. Methods for efficient computation of such probabilities for arbitrary bounding intervals and arbitrary pdfs have been studied extensively in [10, 9, 11]. Therefore, with each reference r_{ik} we can associate two values $\langle j_{ik}, p_{ik}\rangle$ defined as follows: $p_{ik} = \max_{l=1,2,\ldots,N} p_l$, and $j_{ik} = j : p_j = p_{ik}$. That is, for reference r_{ik}, entity $y_{j_{ik}}$ has the highest probability (p_{ik}) to be r_{ik}^*, where p_{ik} is computed based on the current bounding intervals and pdfs. Notice, for each reference r_{ik} from set R defined above, the value of p_{ik} is always 1.0 (e.i., 100%).

[15] More precise bounds can be computed (quickly as well) by various methods. For instance, many terms of the equations being considered are in the form of $\frac{w_j}{c+w_1+w_2+\cdots w_n}$, where $1 \leq j \leq n$, and c is a non-negative constant. We can bound those by noticing that $\frac{w_j}{c+w_1+w_2+\cdots w_n} \in \left[0, \frac{1}{c+1}\right]$, given that $0 \leq w_l \leq 1$ $(l = 1, 2, \ldots, n)$.

The generic algorithm can employ those j_{ik}'s and p_{ik}'s to solve (7) in a variety of ways. For example, it can always maintain the value p_{max} of the maximum of all p_{ik}. It each step it can resolve each reference r_{ik} for which $p_{ik} = p_{max}$ to the corresponding $y_{ikj_{ik}}$ and assign weight of 1 to $w_{ikj_{ik}}$ and weight of 0 to w_{ikl} $(l = 1, 2, \ldots, |S_{ik}|, l \neq j_{ik})$. Notice, this weight assignment can shrink certain bounding intervals and therefore change j_{ik}'s and p_{ik}'s. The algorithm proceeds until no unresolved references are left. Since at each step the algorithm resolves at least one reference, the algorithm is guaranteed to terminate.

C Alternative WM formulae

C.1 Addressing drawbacks of Equation (4)

One could argue that the formula in Equation (4) does not address properly the situation illustrated in Figure 56. In the example in Figure 56, when disambiguating references r_{ik} the choice set for this reference S_{ik} has three elements y_1, y_2, and y_3.

In Figure 56(a) the connection strengths $c_j = c(x_i, y_j)$ $(j = 1, 2, 3)$ 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^*_{ik} is y_1 or y_2 and found insubstantial evidence that r^*_{ik} is y_3. However Equation (4) 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^*_{ik}.

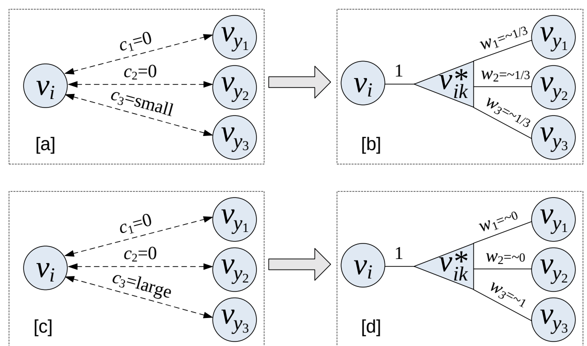

Figure 56: Motivation for *Normalization method 2*

One can argue that in such a situation, since the evidence that r^*_{ik} 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 Equation (4) and achieve the desired weight assignment as follows. We will assume that since y_1, y_2, and y_3 are in the choice set S_{ik} of reference r_{ik} (whereas other entities are not in the choice set), in such situations there is always a very small default connection strength α between each x_i and y_j. That is, Equation (4) is modified and the weights are assigned as follows:

$$w_j = \frac{(c_j + \alpha)}{\sum_{l=1}^{N}(c_l + \alpha)}. \tag{28}$$

where α is a small positive weight: $\alpha \in \mathbb{R}^+$. Equation (28) corrects the mentioned drawbacks of Equation (4).