Record Linkage with Uniqueness Constraints and Erroneous Values

Songtao Guo
AT&T Interactive Research
sguo@attinteractive.com

Xin Luna Dong
AT&T Labs-Research
lunadong@research.att.com

Divesh Srivastava
AT&T Labs-Research
divesh@research.att.com

Remi Zajac
AT&T Interactive Research
rzajac@attinteractive.com

ABSTRACT

Many data-management applications require integrating data from a variety of sources, where different sources may refer to the same real-world entity in different ways and some may even provide erroneous data. An important task in this process is to recognize and merge the various references that refer to the same entity. In practice, some attributes satisfy a *uniqueness* constraint—each real-world entity (or most entities) has a unique value for the attribute (*e.g.*, business contact phone, address, and email). Traditional techniques tackle this case by first linking records that are likely to refer to the same real-world entity, and then fusing the linked records and resolving conflicts if any. Such methods can fall short for three reasons: first, erroneous values from sources may prevent correct linking; second, the real world may contain exceptions to the uniqueness constraints and always enforcing uniqueness can miss correct values; third, locally resolving conflicts for linked records may overlook important global evidence.

This paper proposes a novel technique to solve this problem. The key component of our solution is to reduce the problem into a k-partite graph clustering problem and consider in clustering both similarity of attribute values and the sources that associate a pair of values in the same record. Thus, we perform global linkage and fusion simultaneously, and can identify incorrect values and differentiate them from alternative representations of the correct value from the beginning. In addition, we extend our algorithm to be tolerant to a few violations of the uniqueness constraints. Experimental results show accuracy and scalability of our technique.

1. INTRODUCTION

The amount of information produced in the world increases by 30% every year [25] and this rate will only go up. In many domains, such as business, organizations, publications, music, video, movie, sports, travel, vehicle, housing, there exist a large number of data sources and a lot of their data overlap. Different sources can provide information about the same real-world entities; though, they may represent the same attribute value in different ways, and some may even provide erroneous values. An important task in integrating data from various sources is to recognize the various references that refer to the same real-world entity.

Proceedings of the VLDB Endowment, Vol. 3, No. 1

In practice, there are often attributes that satisfy a *uniqueness* constraint, where each real-world entity (or most entities) has a unique value for the attribute; examples include website, contact phone, address, and email address of businesses, cell-phone number, email address, and Facebook account of people, and president and website of organizations, and so on. However, the data may not satisfy the constraints, either because some sources can provide erroneous values, or because there can be a small number of exceptions in the real world. Traditional techniques handle this case in two steps: first, the *record linkage* step (surveyed in [14, 31]) links records that are likely to refer to the same real-world entity, implicitly requiring consistency of the linked records or explicitly enforcing constraints to some extent; then, the *data fusion* step (surveyed in [12]) merges the linked records and decides the correct values for each result entity in the presence of conflicts.

Such techniques have at least three problems, illustrated by Table 1. First, erroneous values may prevent correct linking. In the example, careless linkage may merge the "MS Corp." record from S_{10} with the "Macrosoft" records, as they share phone and address, while failing to merge them with the "MS Corp." records from S_7 and S_8, needless to mention the "Microsoft" records; if we realize that S_{10} confuses between *Microsoft* and *Macrosoft* and provides wrong values, we are more likely to obtain the correct linkage results. Second, such techniques can fall short when exceptions to the uniqueness constraints exist. In the example, enforcing uniqueness can miss the correct number "9400" for *Microsoft.* Third, locally resolving conflicts for linked records may overlook important global evidence. In the example, suppose we have correctly merged all "MS Corp" records with other *Microsoft* records; then the fact that "0500" is provided by more sources for *Macrosoft* provides further evidence that it is incorrect for *Microsoft.*

This paper presents a novel technique to solve the record linkage problem with uniqueness constraints and erroneous values. The key idea in our solution is to merge the linkage step and the fusion step, so we are able to identify incorrect values and differentiate them from alternative representations of the correct value from the beginning, and obtain better linkage results. Another crucial part of our solution is to make global decisions based on which sources associate a pair of values in the same record, so we can obtain better fusion results. Finally, although our solution relies on uniqueness constraints to detect erroneous values, we allow a small number of violations to capture real-world exceptions.

In particular, this paper makes three contributions:

1. We reduce our problem into a k-partite graph clustering problem. Our clustering technique considers both similarity of attribute values and the set of sources that associate a pair

Table 1: Records from 10 sources on 2 businesses. Phone **and** Address **satisfy uniqueness constraints. There exist different representations for the same value (listed in (b)) and erroneous values (in italics).**

(a) Data sources

SOURCE	NAME	PHONE	ADDRESS
S_1	Microsofe Corp.	xxx-1255	1 Microsoft Way
	Microsofe Corp.	xxx-9400	1 Microsoft Way
	Macrosoft Inc.	xxx-0500	2 Sylvan W.
S_2	Microsoft Corp.	xxx-1255	1 Microsoft Way
	Microsofe Corp.	xxx-9400	1 Microsoft Way
	Macrosoft Inc.	xxx-0500	2 Sylvan Way
S_3	Microsoft Corp.	xxx-1255	1 Microsoft Way
	Microsoft Corp.	xxx-9400	1 Microsoft Way
	Macrosoft Inc.	xxx-0500	2 Sylvan Way
S_4	Microsoft Corp.	xxx-1255	1 Microsoft Way
	Microsoft Corp.	xxx-9400	1 Microsoft Way
	Macrosoft Inc.	xxx-0500	2 Sylvan Way
S_5	Microsoft Corp.	xxx-1255	1 Microsoft Way
	Microsoft Corp.	xxx-9400	1 Microsoft Way
	Macrosoft Inc.	xxx-0500	2 Sylvan Way
S_6	Microsoft Corp.	*xxx-2255*	1 Microsoft Way
	Macrosoft Inc.	xxx-0500	2 Sylvan Way
S_7	MS Corp.	xxx-1255	1 Microsoft Way
	Macrosoft Inc.	xxx-0500	2 Sylvan Way
S_8	MS Corp.	xxx-1255	1 Microsoft Way
	Macrosoft Inc.	xxx-0500	2 Sylvan Way
S_9	Macrosoft Inc.	xxx-0500	2 Sylvan Way
S_{10}	MS Corp.	*xxx-0500*	*2 Sylvan Way*

(b) Real-world entities

NAME	PHONE	ADDRESS
Microsoft Corp., Microsofe Corp., MS Corp.	xxx-1255, xxx-9400	1 Microsoft Way
Macrosoft Inc.	xxx-0500	2 Sylvan Way, 2 Sylvan W.

of values in the same record, thus performing global linkage and fusion simultaneously.

2. We consider *soft uniqueness* for capturing possible exceptions of the constraints. We extend our algorithm to distinguish alternative correct values from erroneous ones by analysis of supporting sources.
3. We have conducted extensive experiments on both real-world data and synthetic data, showing the accuracy and scalability of our technique.

Our algorithms can plug in state-of-the-art record-linkage and data-fusion techniques. We can apply various linkage methods to compute similarity of vectors of values for attributes that do not satisfy any uniqueness constraint, and apply various fusion methods to compute a weight for each source based on its accuracy [11].

The rest of the paper is organized as follows. Section 2 defines the problem. Section 3 describes clustering under hard constraints and Section 4 extends it to deal with soft constraints. Section 5 describes experimental results. Section 6 discusses related work and Section 7 concludes. The Appendix describes details of the algorithms, extensions, and experiments on synthetic data.

2. PROBLEM DEFINITION

This section formally defines the problem we solve and how we convert it to a k-partite graph clustering and matching problem.

2.1 Problem definition

Entity Let $\mathcal{E}$ be a set of real-world *entities* in the same domain. Each entity is described by a set of *attributes*, and each entity contains zero, one, or several values for each attribute. Here, we consider atomic values (string, number, date, etc.). We assume a value can have various *representations* (*e.g.*, New York City can be represented as "New York City" or "NYC")[1].

We focus our attention on a special kind of attribute, *uniqueness attributes*, which satisfy the *uniqueness constraint*; that is, each entity has at most one value for the attribute and different entities have different values[2]. We formally define the constraint as follows.

DEFINITION 2.1 (HARD UNIQUENESS CONSTRAINT). *Let $\mathcal{E}$ be a set of entities of domain D and A be an attribute in D. We say there is a* uniqueness constraint *for A w.r.t. $\mathcal{E}$, denoted by $D \leftrightarrow A$, if each entity in $\mathcal{E}$ has a unique value or no value of A.* □

Essentially, a uniqueness attribute is a nullable key. By transitivity, there is a one-to-one relationship between each pair of uniqueness attributes. Among them, we assume existence of an *identifier (key)* attribute, for which each entity has at least one value and the value can identify the entity (*e.g.*, person name and business name).

In reality, although a uniqueness constraint may apply to most entities, there can be a few violations. For example, although most businesses have unique phone numbers, some can have multiple phone numbers and some can share the same phone number with others. We thus define a relaxed version of the constraint.

DEFINITION 2.2 (SOFT UNIQUENESS CONSTRAINT). *Let $\mathcal{E}$ be a set of entities of domain D and A be an attribute in D. A* soft uniqueness constraint *for A w.r.t. $\mathcal{E}$ is denoted by $D \overset{1-p_1}{\underset{1-p_2}{\rightleftarrows}} A$, where p_1 is the upper bound probability of an entity having multiple values for A and p_2 is the upper bound probability of a value of A being shared by multiple entities.* □

As an example, Business $\overset{70\%}{\underset{90\%}{\rightleftarrows}}$ phone means up to 30% businesses have multiple phone numbers and up to 10% phone numbers are shared by multiple businesses. In practice, p_1 and p_2 can be set by domain knowledge. The definition does not limit the number of values each violating entity can contain, or vice versa, which should be decided according to the data.

Data source Let $\mathcal{S}$ be a set of relational *data sources*. For each entity in $\mathcal{E}$, a source can (but does not necessarily) provide a set of records, which may contain different values of an attribute, or different representations of the same value[3]. Some of the values may not conform to the real world and are *false*; thus, individual sources may violate uniqueness constraints. We assume the schema matching problem has been solved using existing techniques (surveyed in [27]) and so source data are in a uniform schema.

In this paper we solve the following problem: *given a set $\mathcal{S}$ of independent data sources and a set of (hard or soft) uniqueness constraints, identify (1) the set of real-world entities described by $\mathcal{S}$, and (2) discover the* true *values (if any) and different representations of each true value for uniqueness attributes.*

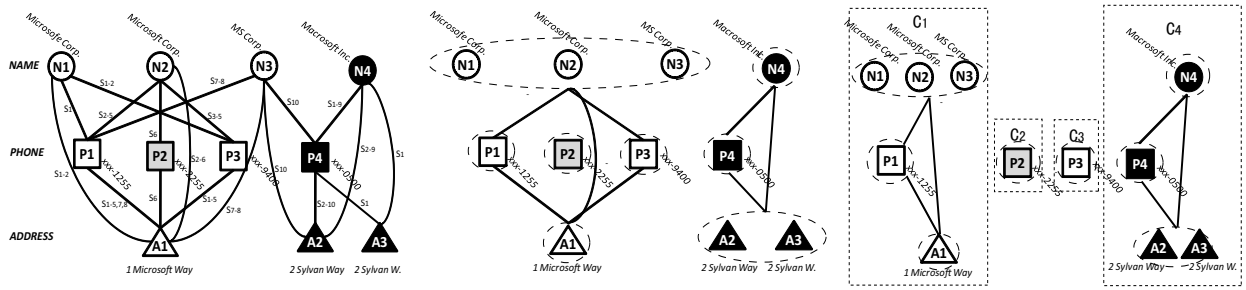
2.2 K-partite graph encoding

Solving our problem requires identifying duplicates by linking various representations of the same value and resolving conflicts by finding the correct value(s). We can thus view this problem as

[1]We assume that one representation represents a single value, which is common in practice, and describe in Appendix C how to relax this assumption.

[2]Our techniques can be easily extended to the case where several attributes jointly satisfy a uniqueness constraint.

[3]If a source provides multiple values or representations for an attribute in one record, we can decompose the record into multiple ones.



(a) 3-Partite graph encoding of the input (b) Encoding of the ideal solution (c) Clustering under hard constraints

Figure 1: Graph encoding for the motivating example. N-nodes, P-nodes, and A-nodes are for names, phones, addresses correspondingly. Nodes with the same shading belong to the same entity. A dashed oval represents a cluster of representations for the same value, and a dashed rectangle represents a cluster of value representations for the same entity.

clustering various representations into values, and *matching* (associating) values that belong to the same entity. To facilitate this process, we define a k-*partite graph encoding* of our problem. We consider only (hard or soft) uniqueness attributes for now and consider other attributes in Appendix C.

DEFINITION 2.3 (K-PARTITE GRAPH ENCODING). *Let $\mathcal{E}$ be a set of entities with k uniqueness attributes $A_1, \ldots, A_k$. Let $\mathcal{S}$ be a set of data sources providing data on $\mathcal{E}$. The* k-partite graph encoding *of $\mathcal{S}$ is an undirected graph $G(\mathcal{S}) = (V_1, \ldots, V_k, E)$, such that*

- *each node in $V_i, i \in [1,k]$, represents a value representation of attribute A_i, provided by a source in $\mathcal{S}$;*
- *each edge $(v_i, v_j) \in E, v_i \in V_i, v_j \in V_j, i, j \in [1,k], i \neq j$, represents existence of a record with value representations v_i and v_j, and is marked with $\bar{S}(v_i, v_j)$, the set of sources that provide such records.* □

As an example, Figure 1(a) shows the 3-partite graph encoding of the data set in Table 1(a). The size of the graph is linear in the size of the input data. We note that although the k-partite graph can lose information on which edges come from the same record, the lost information is not critical (see the full version [18]).

Based on this k-partite graph encoding, we can encode a solution of our problem as follows.

DEFINITION 2.4 (SOLUTION ENCODING). *Let $G(\mathcal{S}) = (V_1, \ldots, V_k, E)$ be a k-partite graph encoding for data sources $\mathcal{S}$ on entities $\mathcal{E}$. A solution encoding has two parts:*

- *for each $i \in [1,k]$, there is a clustering of V_i such that each cluster represents a unique value of A_i;*
- *for each pair of clusters C_i and $C_j, C_i \subseteq V_i, C_j \subseteq V_j, i, j \in [1,k], i \neq j$, there is an edge between C_i and C_j if and only if they belong to the same entity in $\mathcal{E}$.* □

Figure 1(b) shows the encoding of the ideal solution in Table 1(b). In the special case where we consider only hard constraints, for each $i, j \in [1,k], i \neq j$, a cluster in V_i can be connected with at most one cluster in V_j, and vice versa. We can accordingly further cluster all nodes in the k-partite graph into entities (a cluster without a key value does not represent a valid entity). Figure 1(c) shows the clustering under hard constraints on phone and address. In this case, our problem is reduced to a pure clustering problem, where each cluster includes at most a single value, with different representations, for each attribute, so the clustering process conducts linkage and fusion at the same time.

In the rest of our paper, Section 3 describes a clustering algorithm with hard constraints, Section 4 describes an algorithm for soft constraints, and Appendix C describes a few extensions.

3. CLUSTERING W.R.T. HARD CONSTRAINTS

We start from hard constraints, in which case the problem can be reduced to a k-partite graph clustering problem. This section first presents our objective function for clustering, and then describes our clustering algorithm.

3.1 Objective function

An ideal clustering should have a high *cohesion* within each cluster and a low *correlation* between different clusters. Several objective functions have been proposed for clustering taking into consideration cohesion and correlation, such as Davies-Bouldin index [9], Dunn index [13], and Silhouette index [28]. The choice of the index is orthogonal to our techniques; here we adopt the Davies-Bouldin index, which is more stable than the Dunn index and less expensive to compute than the Sihouette index [24, 26].

Formally, given a clustering $\mathcal{C} = \{C_1, \ldots, C_n\}$, its Davies-Bouldin index (DB-index) is defined as follows:

$$\Phi(\mathcal{C}) = \text{Avg}_{i=1}^{n}\big(\max_{j \in [1,n], j \neq i} \frac{d(C_i, C_i) + d(C_j, C_j)}{d(C_i, C_j)}\big), \quad (1)$$

where $d(C_i, C_j)$ denotes the *distance* between C_i and C_j. Note that when $i = j$, the distance is the complement of the cohesion of C_i (C_j); otherwise, the distance is the complement of the correlation between C_i and C_j. Our goal is to obtain a clustering with the *minimum* DB-index, implying high cohesion and low correlation.

Now we consider how to compute cluster distance. Intuitively, if two clusters have a high distance, their values of the same attribute should be very different, and their values of different attributes should be associated by few edges. We thus consider two types of distance: *similarity distance*, denoted by d_S, measuring (complement of) similarity between value representations of the same attribute; and *association distance*, denoted by d_A, measuring (complement of) association between value representations of different attributes. The cluster distance takes their average:

$$d(C_i, C_j) = \frac{d_S(C_i, C_j) + d_A(C_i, C_j)}{2}. \quad (2)$$

We next describe how we compute d_S and d_A.

3.1.1 Similarity distance

For similarity distance d_S, we first compute the distance for each attribute, denoted by $d_S^l, l \in [1,k]$, and then take the average:

$$d_S(C_i, C_j) = \text{Avg}_{l=1}^{k}(d_S^l(C_i, C_j)). \quad (3)$$

Table 2: Similarity matrices for names and addresses. For phone numbers, the similarity is 1 between the same number and 0 otherwise.

(a) Name

	N_1	N_2	N_3	N_4
N_1	1.0	0.95	0.65	0.7
N_2	0.95	1.0	0.65	0.7
N_3	0.65	0.65	1.0	0.4
N_4	0.7	0.7	0.4	1.0

(b) Address

	A_1	A_2	A_3
A_1	1.0	0	0
A_2	0	1.0	0.9
A_3	0	0.9	1.0

For each $l \in [1, k]$, we compute d_S^l by averaging the similarity of each pair of value representations. Formally, let $\bar{R}_i$ (resp. $\bar{R}_j$) be the value representations of A_l in cluster C_i (resp. C_j). Then,

$$d_S^l(C_i, C_j) = 1 - \text{Avg}_{r \in \bar{R}_i, r' \in \bar{R}_j, r \neq r'} \text{sim}(r, r'), \quad (4)$$

where $\text{sim}(r, r')$ is the similarity between two value representations r and r', and its value is between 0 and 1. As special cases, if $i = j$ and $\bar{R}_i$ contains a single representation, $d_S^l(C_i, C_i) = 0$; if $\bar{R}_i$ or $\bar{R}_j$ is empty, we do not consider $d_S^l(C_i, C_i)$ in Eq. (3).

A similarity function (sim) needs to be defined for each attribute. Such function can be string similarity [8], numerical similarity, etc.; the choice is orthogonal to our clustering algorithm.

EXAMPLE 3.1. *Consider the clustering in Figure 1(c) for the motivating example. Table 2 shows similarity between value representations for each attribute.*

*For cluster C_1, there are three pairs of names, so $d_S^1(C_1, C_1) = 1 - \frac{0.95+0.65+0.65}{3} = 0.25$ (*name*); there is a single phone and address, so $d_S^2(C_1, C_1) = 0$ (*phone*), $d_S^3(C_1, C_1) = 0$ (*address*). Taking the average, $d_S(C_1, C_1) = \frac{0.25+0+0}{3} = 0.083$.*

Between clusters C_1 and C_4, there are again three pairs of names, so $d_S^1(C_1, C_4) = 1 - \frac{0.7+0.7+0.4}{3} = 0.4$; $similarly, d_S^2(C_1, C_4) = 1 - \frac{0}{1} = 1, d_S^3(C_1, C_4) = 1 - \frac{0+0}{2} = 1$. Thus, $d_S(C_1, C_4) = 0.8$. On the other hand, between clusters C_1 and C_2, as C_2 contains only one node P_2, $d_S(C_1, C_2) = d_S^2(C_1, C_2) = 1$. □

3.1.2 Association distance

For association distance d_A, we first compute the distance for each pair of the k attributes, denoted by $d_A^{l,l'}, l, l' \in [1, k], l \neq l'$, and then take the average.

$$d_A(C_i, C_j) = \text{Avg}_{l, l' \in [1,k], l \neq l'} d_A^{l,l'}(C_i, C_j). \quad (5)$$

We next describe how we compute $d_A^{l,l'}$ for each pair of l and l'. When $i = j$, intuitively, the association is represented by the edges between V_l-nodes and $V_{l'}$-nodes in C_i. We can take the fraction of the sources that support any of such edges over all sources that provide V_l- or $V_{l'}$-nodes in C_i. If a source provides several records with various representations of the same entity, we count it only once. Formally, let $\bar{S}^l(C_i)$ (resp. $\bar{S}^{l'}(C_i)$) be the sources that provide a V_l-node (resp. $V_{l'}$-node) in C_i. Let $\bar{S}^{l,l'}(C_i)$ be the sources that support an edge between a V_l-node and a $V_{l'}$-node in C_i. Let $|\bar{S}|$ be the size of set $\bar{S}$. Then, we compute the distance as[4]

$$d_A^{l,l'}(C_i, C_i) = 1 - \frac{|\bar{S}^{l,l'}(C_i)|}{|\bar{S}^l(C_i) \cup \bar{S}^{l'}(C_i)|}. \quad (6)$$

When $i \neq j$, we first compute the association between V_l-nodes in C_i and $V_{l'}$-nodes in C_j, and the association between V_l-nodes in C_j and $V_{l'}$-nodes in C_i. Intuitively, even if only one of the associations is strong, there can be a better clustering (*e.g.*, if the former is strong, moving some of the V_l-nodes in C_i into C_j may obtain a better clustering); thus, we consider the stronger association. If we denote by $\bar{S}^{l,l'}(C_i, C_j)$ the sources that support an edge between a V_l-node in C_i and a $V_{l'}$-node in C_j, we have

$$d_A^{l,l'}(C_i, C_j) = 1 - \max\{\frac{|\bar{S}^{l,l'}(C_i, C_j)|}{|\bar{S}^l(C_i) \cup \bar{S}^{l'}(C_j)|}, \frac{|\bar{S}^{l,l'}(C_j, C_i)|}{|\bar{S}^l(C_j) \cup \bar{S}^{l'}(C_i)|}\}. \quad (7)$$

EXAMPLE 3.2. *Consider the clustering in Figure 1(c) for the motivating example. For cluster C_1, 9 sources ($S_1, \ldots, S_8, S_{10}$) mention at least one node of* name *or* phone*, and 7 sources ($S_1, \ldots, S_5, S_7, S_8$) support associations between* name *and* phone *in C_1; thus, $d_A^{1,2}(C_1, C_1) = 1 - \frac{7}{9} = 0.22$. Similarly, $d_A^{1,3}(C_1, C_1) = 1 - \frac{8}{9} = 0.11$ and $d_A^{2,3}(C_1, C_1) = 1 - \frac{7}{8} = 0.125$. Taking the average, $d_A(C_1, C_1) = 0.153$.*

Consider clusters C_1 and C_4. There is an edge from N_3 to P_4 with the supporter s_{10} (so $|\bar{S}^{1,2}(C_1, C_4)| = 1$), and there is no edge from N_4 to P_1 (so $|\bar{S}^{1,2}(C_4, C_1)| = 0$). Therefore, $d_A^{1,2}(C_1, C_4) = 1 - \frac{1}{10} = 0.9$. Similarly, $d_A^{1,3}(C_1, C_4) = 0.9$ and $d_A^{2,3}(C_1, C_4) = 1$. Taking the average, $d_A(C_1, C_4) = 0.93$.

To compute the DB-index, we find for each cluster its "farthest" cluster and apply Eq. (1): $\Phi = \frac{0.32+0.14+0.32+0.19}{4} = 0.24$. □

3.1.3 Augmentations

We apply two augmentations. First, we discount sources that appear to provide multiple values for an attribute (often due to an inappropriate clustering). Second, we distinguish *logical* similarity between value representations from their appearance similarity (*i.e.*, string similarity). We give details in Appendix A.

3.2 Hill-climbing algorithm

Previous work [17, 29] has shown intractability of clustering in most cases. We now describe an efficient hill-climbing algorithm, CLUSTER (details in Appendix B), that approximates the optimal solution. CLUSTER first generates an initial clustering, then iteratively examines each node and assigns it to the "best" cluster.

Step 1: Initialization. First cluster value representations of each attribute according to their similarity. Then, between clusters of the key attribute and of each non-key-attribute, apply the Hungarian algorithm [23] to find the one-to-one matching with the strongest associations (computed as the sum of the number of supporting sources on each selected edge).

Step 2: Adjustment. For each node N, compute the DB-index of assigning N to each cluster and not changing clusters of other nodes. Assign N to the cluster that minimizes the DB-index.

Step 3: Convergence checking. Repeat Step 2 if the clustering changes.

Note that the initialization and the order in which we adjust the nodes may change the results; however, since the algorithm iterates, we did not observe much difference in our experiments.

EXAMPLE 3.3. *We apply the* CLUSTER *algorithm on the data set in the motivating example. Initially, we cluster N_1 and N_2, and A_2 and A_3, given their high similarity, and obtain a clustering shown in Table 3(a), with DB-index* 0.89.

The first iteration starts with examining node N_1. Moving N_1 to C_1, C_2, or C_4 results in a DB-index of $0.94, 1.16, 0.93$*, respectively, so we keep N_1 in C_3. We then examine the rest of the nodes and decide to move N_2 to C_1 and not move other nodes. The resulting clustering, shown in Table 3(b), has DB-index of* 0.71.

The second iteration moves N_1 to C_1 (Table 3(c)) and decreases the DB-index to 0.45*. Then the algorithm converges.* □

We next formalize several properties of the algorithm.

[4] Instead of counting the number of sources, we can assign a weight to each source according to its accuracy [11], and sum up the weights of the sources.

Table 3: Apply CLUSTER on data sets in the motivating example.

(a) Initial clustering.

	C_1	C_2	C_3	C_4
NAME	N_3		N_1 ,N_2	N_4
PHONE	P_1	P_2	P_3	P_4
ADDRESS	A_1			A_2,A_3

(b) Clustering after the first iteration.

	C_1	C_2	C_3	C_4
NAME	N_2,N_3		N_1	N_4
PHONE	P_1	P_2	P_3	P_4
ADDRESS	A_1			A_2,A_3

(c) Final clustering.

	C_1	C_2	C_3	C_4
NAME	N_1,N_2,N_3			N_4
PHONE	P_1	P_2	P_3	P_4
ADDRESS	A_1			A_2,A_3

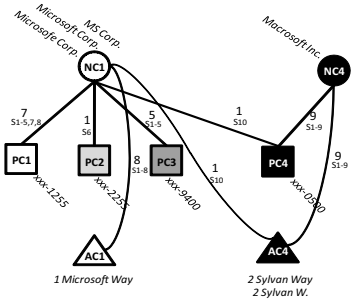


Figure 2: The transformed graph of clustering in Figure 1(c).

THEOREM 3.4. CLUSTER *has the following properties:*

1. CLUSTER *converges.*
2. *Let n be the number of nodes in the input k-partite graph, m be the number of sources, and l be the number of iterations. The time complexity of* CLUSTER *is* $O((2^k + lk)mn^4)$. □

In practice, CLUSTER typically converges in only a few iterations. Also, although it takes exponential time in k, it is not that expensive in practice as k is typically very small. However, the algorithm takes quartic time in n and can be expensive when n is large, we thus pre-process the data as follows.

Pre-processing: We first partition the records such that only records that share similar values and would possibly represent the same entity are put in the same partition. Then, for each partition separately we generate the k-partite graph and apply CLUSTER. Our experiments (Section 5) show that with pre-processing, our algorithm takes linear time in the number of records.

4. MATCHING W.R.T. SOFT CONSTRAINTS

We now describe how we extend our approach to deal with soft constraints by reducing our problem to an optimization problem, and give the solution.

4.1 Soft constraints and objective function

Recall that a soft constraint can be represented as $D \overset{1-p_1}{\underset{1-p_2}{\rightleftharpoons}} A$, meaning that with probability up to p_1 an entity has multiple values for attribute A and with probability up to p_2 a value of A is shared by multiple entities. Formally, let $|A_\kappa|$ be the number of values of the key attribute A_κ (each entity has one and only one key value), and $|\hat{A}_\kappa|$ be the number of A_κ's values that are matched to multiple values of A. We define $|A|$ and $|\hat{A}|$ similarly. Then,

$$0 \leqslant \frac{|\hat{A}_\kappa|}{|A_\kappa|} \leqslant p_1, \quad 0 \leqslant \frac{|\hat{A}|}{|A|} \leqslant p_2. \tag{8}$$

Now our goal is to cluster representations to values for each attribute, and for each soft uniqueness attribute, match (associate) its values with values of the key attribute that belong to the same entity. CLUSTER already clusters representations to values, so we can start from there, merging the clustered nodes, grouping the edges, and computing the *weight* for each edge as the number of supporting sources. The result is essentially a set of bi-partite graphs between key and non-key nodes. As an example, Figure 2 shows the graph transformed from the clustering result in Figure 1(c). Now the problem becomes finding the best matching between A_κ and each soft uniqueness attribute, under conditions (8). We next consider what is the "best" matching between the key attribute A_κ and a particular attribute A.

Typically, we are confident of matching several values of A with a value v_κ of A_κ (or vice versa) only if the edges between these values and v_κ have similar numbers of supporting sources. To capture this intuition, we define the *gap* of support between different selected edges for the same node. We denote by $\bar{M}$ a matching solution with all selected edges, and by $w(e)$ the weight of edge e.

DEFINITION 4.1 (SUPPORT GAP). *Let $G^c(\mathcal{S})$ be the input of the matching problem and let $\bar{M}$ be a matching solution between attributes A_κ and A. Let v be a node for A_κ or A in G. Let $\bar{E}(v) \subseteq \bar{M}$ be the selected edges that are connected with v. The* support gap *for v is defined as Gap$(v) = \max_{e\in\bar{E}(v)} w(e) - \min_{e\in\bar{E}(v)} w(e)$.* □

When we look for the best matching, we aim at maximizing the sum of the weights of selected edges, while minimizing the gap for each node. We thus define the score of a matching $\bar{M}$ as

$$\text{Score}(\bar{M}) = \sum_{(u,v)\in\bar{M}} \frac{w(u,v)}{\text{Gap}(u) + \text{Gap}(v) + \alpha}. \tag{9}$$

Here, α is used to avoid the divide-by-zero error when the gaps are 0 and to smooth the penalty when the gaps are very small. We set α by first computing the standard deviation of the edge weights for each node of A and A_κ, then taking the average for A and for A_κ (separately), and finally summing the results up. This strategy considers the majority difference of weights for each node and works well in our experiments.

A nice property of the score function in Eq. (9) is that if there are two disconnected subgraphs G_1^c and G_2^c of G^c, the matching decision on G_1^c is independent of the matching decision on G_2^c, formalized in the following proposition.

PROPOSITION 4.2 (INDEPENDENCE). *Let G_1^c and G_2^c be two disconnected subgraphs of G^c such that $G_1^c \cup G_2^c = G^c$. Let $\bar{M}_1$ be a matching on G_1^c and $\bar{M}_2$ be a matching on G_2^c. Then, Score$(\bar{M}_1 \cup \bar{M}_2)$ = Score$(\bar{M}_1)$ + Score$(\bar{M}_2)$.* □

EXAMPLE 4.3. *Consider $G^c(\mathcal{S})$ in Figure 2 and we focus on the matching between names and phone numbers. Two matching solutions are shown in Figure 1(b) and Figure 1(c), and they differ in whether to include the edge between NC_1 and P_3. For the matching in Figure 1(c), no node is associated with multiple edges and the gap for each node is 0, so the score is $\frac{7}{0+2.91} + \frac{9}{0+2.91} = 5.49$ (here, $\alpha = 2.91$). For the matching in Figure 1(b), node NC_1 is associated with two edges and the gap is 2. However, including the additional edge is worthwhile as the score is increased to $\frac{7}{2+2.91} + \frac{5}{2+2.91} + \frac{9}{0+2.91} = 5.54$.* □

To summarize, our matching problem is reduced to the following optimization problem `match` between the key attribute and each

soft uniqueness attribute. We note that even if p_1 (p_2) is high, the optimal solution may contain only a few exceptions so the objective function is maximized.

$$\text{maximize} \sum_{(u,v)\in\bar{M}} \frac{w(u,v)}{\text{Gap}(u)+\text{Gap}(v)+\alpha}$$

$$\text{subject to } 0 \leqslant \frac{|\hat{A}_\kappa|}{|A_\kappa|} \leqslant p_1, 0 \leqslant \frac{|\hat{A}|}{|A|} \leqslant p_2.$$

4.2 Two-phase greedy algorithm

Solving a non-linear optimization problem is generally hard [22]; again, we need an efficient algorithm to approximate the optimal solution. Taking a graph $G^c(\mathcal{S})$ involving the key attribute A_κ and a soft uniqueness attribute A, our matching algorithm, MATCH (details in Appendix B) proceeds in two phases: the first phase greedily selects nodes that may match to multiple nodes, and the second phase greedily selects edges to maximize the objective function. Note that Phase 1 is critical: going to Phase 2 directly may select a high-weight edge that introduces a large gap and so prevent later choosing a lower-weight edge that introduces a smaller gap and can increase the score even more.

Phase 1. Node selection. Consider attribute A_κ (similar for A).

- Rank all edges on weight in a descending order.
- Rank the nodes of A_κ by $LS_2(v) = \frac{w_1+w_2}{w_1-w_2+\alpha}$ in a descending order, where w_1 and w_2 are the two highest weights for node v. Select the top $p_1|A_\kappa|$ nodes as candidates and denote the result set by $\bar{V}$. Set degree $\deg(v) = 1$ for $v \notin \bar{V}$.
- For each node $v \in \bar{V}$, start from a subset $\bar{E}(v)$ that contains edges with the top-2 weights and compute a *local score*:

$$\text{LS}(v) = \frac{\sum_{e\in\bar{E}(v)} w(e)}{\max_{e\in\bar{E}(v)} w(e) - \min_{e\in\bar{E}(v)} w(e) + \alpha}. \quad (10)$$

  Progressively add more edges in descending order of the weights until the score does not increase. Set $\deg(v) = |\bar{E}(v)|$.

The local score is defined assuming the same gap for nodes of A. Ideally, we should rank nodes by their local scores; we save the computation by comparing nodes by LS_2 (only top-2 weighted edges) and computing LS only for nodes in the top list.

Phase 2. Edge selection. We next consider edges in the descending order of weights. We greedily select an edge if adding it increases the score without exceeding the degree limit ($\deg(v)$) for both of its nodes, until there exists no such node.

The complexity of this algorithm is $O(e \log e + n \log n + en)$, where e is the number of edges and n is the number of nodes in $G^c(\mathcal{S})$ for the two attributes in consideration.

EXAMPLE 4.4. *We illustrate how we approximate the matching solution for $G^c(\mathcal{S})$ in Figure 2. We consider matching between* name *and* phone. *In the soft constraint for* phone, $p_1 = p_2 = .5$.

In Phase 1, we select NC_1 and PC_4; they are actually the only nodes associated with multiple edges. By computing the local score, we set 2 as the degree for both nodes.

In Phase 2, we greedily choose new edges to add. We first choose the edge between NC_4 and PC_4 and the score is $\frac{9}{0+0+2.91} = 3.09$ ($\alpha = 2.91$). We then choose the edge between NC_1 and PC_1, increasing the score to $\frac{9}{0+0+2.91} + \frac{7}{0+0+2.91} = 5.50$. The third chosen edge is between NC_1 and PC_3, increasing the score to $\frac{9}{0+0+2.91} + \frac{7}{2+0+2.91} + \frac{5}{2+0+2.91} = 5.54$. Adding other edges violates the degree constraints, so we terminate. □

5. EXPERIMENTAL RESULTS

This section describes experimental results on real-world data sources that provide business listings. Appendix D describes additional experiments on synthetic data. Experimental results show high accuracy and scalability of our techniques.

5.1 Experiment settings

Data: We experimented on a set of raw business listings that YellowPages.com obtained from other sources, where we know the provider of each listing. We considered listings (name, phone, address) in two zip codes: 07035 and 07715, and in the San Francisco area. For each zip code, we manually identified the real-world businesses provided by the sources, and verified their phone numbers and addresses by calling the businesses and referring to other resources at YellowPages.com; we used the results as the golden standard. Table 4 shows statistics of the data; the data have a high variety, contain errors, and roughly observe the constraints. The San Francisco data set was used only for scalability test.

Implementations: We implemented our algorithm, referred to as MATCH. MATCH first invokes CLUSTER and then applies matching for soft constraints. We assumed hard constraint on name and soft constraint on phone and address with violation rate 0.3. We pre-computed representation similarity by TF/IDF Jaro-Winkler [8] distance for names and addresses, and by Levenshtein distance [8] for phone numbers. For augmentation, we applied Eq. (12) for multi-value penalty (see Appendix A), and estimated $p(S)$ by observing the data.

We implemented MATCH in Matlab and other components (database connection and similarity computation) in C#. We used a WindowsXP machine with 2.0GHz Intel CPU and 1.5GB of RAM.

Comparison: For comparison, we implemented three traditional linkage and fusion techniques:

- LINK: For each pair of records, compute value similarity for each attribute and take the average. Link two records if the average similarity is at least .85 and consider all linked records as representing one entity. Consider representations that are in the same entity and have a similarity of at least .95 as the same value (this set of thresholds obtained the best results for LINK in most cases in our experiments).
- FUSE: For each key attribute A_κ and non-key attribute A, first compute a weight w for each pair of values $v_{A\kappa}$ and v_A as the number of sources that associate $v_{A\kappa}$ with v_A, then update the weight as[5]

$$w'(v_{A\kappa}, v_A) = \sum_{v'_A} w(v_{A\kappa}, v'_A) \cdot \text{sim}(v_A, v'_A).$$

  Associate each key value with the non-key value with the highest weight (so many-to-one mappings), and consider all associated values as representing one entity.
- LF: Apply LINK, then choose the correct value for each attribute of each entity as the one provided by the largest number of sources.

Measure: We compared generated results with the golden standard and measured quality of the results by *precision* (P), *recall* (R), and *F-measure* (F) on (1) matching of values of different attributes and (2) clustering of values of the same attribute. For matching, we consider each pair of value representations as matched if they are in the same entity, and denote the set of matched pairs by $\bar{G}_M$ for the golden standard and by $\bar{R}_M$ for our results. We define

[5] FUSE adapts the method in [11], but does not consider accuracy of sources and dependence between sources, which is not the focus of this paper.

Table 4: Statistics of data. Columns 6-8 show the number of distinct names, phones, addresses for each zip code, and Column 9 shows the number of erroneous phone numbers. The last four columns show percentage of businesses (resp., phones/addresses) that violate a particular constraint and the numbers in the parenthesis are average number of correct phones/addresses (resp., businesses) in the violation cases.

Zip	Business	Source		Record					Constraint violation			
	#Business	#Srcs	#Srcs/business	#Recs	#(dist Ns)	#(dist Ps)	#(dist As)	#(Err Ps)	$N \rightarrow P$	$P \rightarrow N$	$N \rightarrow A$	$A \rightarrow N$
07035	662	15	1∼7	1629	1154	839	735	72	8%(2.6)	.8%(2.7)	2%(2.3)	12.6%(5.1)
07715	149	6	1∼3	266	243	184	55	12	4%(2)	1%(3)	4%(2)	4%(8.5)

Table 5: Accuracy on real-world data sets. The last column shows improvement of MATCH over LINK, and the last two rows average F-measures on matching and clustering.

Zip	Category	Msr	FUSE	LINK	LF	MATCH	*Imp*
07035	N-P match	P	.94	.89	.94	.97	*8.9%*
		R	.80	.88	.83	.91	*3.4%*
		F	.87	.88	.88	.94	*6.0%*
	N-A match	P	.97	.97	.98	.97	*0.1%*
		R	.52	.92	.54	.97	*4.8%*
		F	.68	.95	.70	.97	*2.4%*
	Name cluster	P	.98	.92	.93	.98	*6.8%*
		R	.90	.91	.89	.98	*8.2%*
		F	.94	.92	.91	.98	*7.5%*
07715	N-P match	P	.99	.99	.99	.99	*0.0%*
		R	.93	.94	.94	.93	*-0.5%*
		F	.96	.97	.97	.96	*-0.3%*
	N-A match	P	.93	.95	.93	.98	*3.2%*
		R	.55	.81	.57	.95	*18.0%*
		F	.69	.87	.70	.97	*10.7%*
	Name cluster	P	1.0	.91	.91	1.0	*10.4%*
		R	.96	.75	.75	.96	*28.7%*
		F	.98	.82	.82	.98	*19.8%*
Avg	*Match*	*F*	*.80*	*.92*	*.81*	*.96*	*4.7%*
	Cluster	*F*	*.96*	*.87*	*.86*	*.98*	*13.6%*

$P = \frac{|\bar{G}_M \cap \bar{R}_M|}{|\bar{R}_M|}, R = \frac{|\bar{G}_M \cap \bar{R}_M|}{|\bar{G}_M|}, F = \frac{2PR}{P+R}$. For clustering on an attribute A, we consider each pair of representations of A as clustered if they represent the same value, and denote the set of clustered pairs by $\bar{G}_A$ for the golden standard and by $\bar{R}_A$ for our results. We can compute precision, recall, and F-measure similarly.

5.2 Results

Accuracy: Table 5 compares accuracy of various methods. We observe that MATCH obtains the highest F-measure in most cases. On average, it obtains a F-measure of 0.96 on matching and 0.98 on clustering; on the 07715 data set, it improves over LINK by 11% on name-address matching and by 20% on name clustering. Between LINK and FUSE, FUSE typically obtains a higher precision but a lower recall in matching, as it enforces uniqueness but does not explicitly link various representations of the same value; however, FUSE obtains higher precision and recall in name clustering by enforcing that each business has a single phone number. LF, on the other hand, performs only slightly better than FUSE in matching and similar to LINK in clustering, as it enforces uniqueness but cannot handle exceptions, and does not identify false values from the beginning so can mis-cluster.

Contribution of components: We next studied contribution of different components of our algorithm on performance. We started with the initial clustering in CLUSTER, then progressively added cluster refinement, multi-value penalty, and appearance-similarity based association update (Appendix A). We ran these variants of CLUSTER with and without extension for soft constraints. Figure 3 shows average F-measure on name-phone matching and we observed similar patterns for name-address matching and name clustering. We observe that (1) extension for soft constraints is necessary for real data (improving the F-measure by 6.8%); (2) simply applying clustering without the augmentations improves the results over initial clustering only slightly; (3) when we consider soft constraints, applying multi-value penalty makes significant improvement (by 2.3%) whereas considering appearance-similarity further improves the results (by 1%); however, such augmentations do not help and may even do harm when we ignore soft constraints.

We also experimented with changing the initialization method and the order in which we examine clustering of the nodes in each round, and observed similar results.

Contribution of attributes: Figure 4 compares accuracy of MATCH and its three variants on 07715 on clustering of business names: NPONLY considers associations only between name and phone, NAONLY considers associations only between name and address, NP+NA considers these two associations, and NPA (*i.e.*, MATCH) considers all three associations. This data set has many missing addresses; thus, NAONLY has low precision and recall, and NP+NA improves over NPONLY only slightly. We also observe a big improvement by considering associations between phone and address: NPA increases the F-measure over NP+NA by 5%.

Efficiency and scalability: To show scalability and efficiency of our techniques, we experimented on a sample of 236,306 listings for the San Francisco area. In pre-processing, we put listings into the same partition if they 1) have similar names and addresses, 2) have similar names and the same phone number, or 3) have similar address and the same phone number (threshold=.9). Figure 5 shows the execution time of MATCH for each partition. We observe that for 99% partitions the graph size is less than 11 (nodes). The largest graph has 121 nodes and the maximum execution time for a partition is only 327 seconds. Note that although there is a quartic relationship between execution time and the graph size, execution time also depends on distribution of nodes in the graph; as an example, execution on the largest graph takes only 6 seconds, as the graph contains only 2 phone nodes and 3 address nodes. We observe similar distribution of the graph size when we use different thresholds and different similarity measures.

To test scalability, we randomly divided the whole sample into 10 subsets of the same size. We started with one subset and gradually added more. Figure 6 shows the execution time plotted against the number of records. We observe that with pre-processing, the overall execution time and that for the top-10 partitions grow linearly in the size of the data; thus, our algorithm scales.

6. RELATED WORK

Our work is mainly related to two bodies of work: record linkage and data fusion. Record linkage has been extensively studied in the past (surveyed in [14, 31]). Most of the techniques implicitly assume consistency of records that should be matched and can fall short in presence of erroneous values. Recently, there has been work on linkage with constraints [1, 2, 3, 4, 10, 16, 30, 32]. The considered constraints can be classified into several types [7]: constraints on individual tuples (*e.g.*, only some tuples can participate in linkage), deduplication parameters (*e.g.*, number of real-world entities), pairwise positive and negative examples (*i.e.*, requiring merging or non-merging for certain pairs), forbidden rules (*i.e.*,

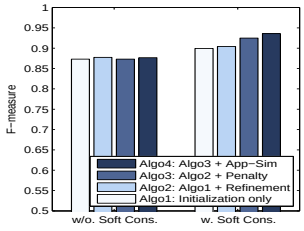


Figure 3: Contribution of components on 07035 data set.

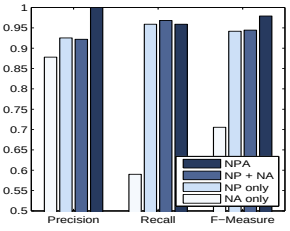


Figure 4: Contribution of associations on 07715 data set.

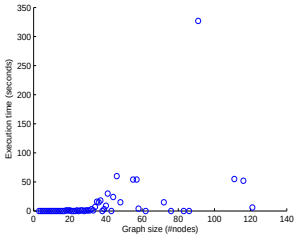


Figure 5: Partitions of the San Francisco data set.

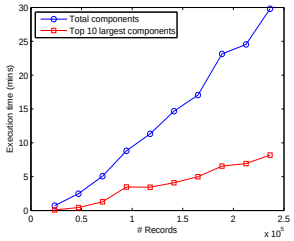


Figure 6: Execution time for the San Francisco data set.

guarantee of distinctness), and group-wise constraints (*i.e.*, condition for aggregation of a group of results). Existing techniques either enforce such constraints strictly [4, 10], or adapt the similarity function to accommodate them [5]. The uniqueness constraint is a kind of forbidden rules; however, instead of blindly enforcing the constraints, we consider possible erroneous values and exceptions.

Data fusion is a new field and studies how to merge linked records and resolve conflicts (surveyed in [12]). Recently, advanced techniques have been proposed to consider accuracy of and dependence between sources in conflict resolution [11].

Our technique is not a simple combination of linkage and fusion, but integrates them seamlessly in k-partite graph clustering. We point out that collective deduplication (surveyed in [21]) applies clustering but clusters records rather than values; [6] clusters values for constraint repair, but does not consider value similarity and associations at meanwhile as we do.

Finally, our work is distinct from data cleaning based on dependency constraints [15], as we need to take various representations into consideration. Our definition of soft constraints is different from relaxed dependency constraints [19, 20] as it is defined on the underlying domain, but not on the provided data.

7. CONCLUSIONS

This paper studies the problem of record linkage in the presence of uniqueness constraints and erroneous values. The key idea of our solution is to integrate linkage and fusion, and apply them in a global fashion. We proposed a k-partite graph clustering algorithm for hard constraints, and extended it to allow for soft constraints. Experiments show high accuracy and scalability of our algorithm.

For future work, we would like to extend our techniques to broader cases with many-to-one relationships between attributes or more general constraints. We would also like to study online record linkage and conflict resolution, which emphasizes efficiency.

8. REFERENCES

[1] J. Aslam, K. Pelekhov, and D. Rus. A practical clustering algorithm for static and dynamic information organization. In *SODA*, 1999.

[2] N. Bansal, A. Blum, and S. Chawla. Correlation clustering. *Mach. Learn.*, 56(1-3):89–113, 2004.

[3] I. Bhattacharya and L. Getoor. Collective entity resolution in relational data. *ACM Trans. Knowl. Discov. Data*, 1(1):5, 2007.

[4] M. Bilenko, S. Basu, and R. J. Mooney. Integrating constraints and metric learning in semi-supervised clustering. In *ICML*, 2004.

[5] M. Bilenko and R. J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *SIGKDD*, pages 39–48, 2003.

[6] P. Bohannon, W. Fan, M. Flaster, and R. Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *Sigmod*, 2005.

[7] S. Chaudhuri, A. Das Sarma, V. Ganti, and R. Kaushik. Leveraging aggregate constraints for deduplication. In *SIGMOD*, 2007.

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

[9] D. L. Davies and D. W. Bouldin. A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1(2):224–227, 1979.

[10] X. Dong, A. Y. Halevy, and J. Madhavan. Reference reconciliation in complex information spaces. In *SIGMOD*, pages 85–96, 2005.

[11] X. L. Dong, L. Berti-Equille, and D. Srivastava. Integrating conflicting data: the role of source dependence. *PVLDB*, 2009.

[12] X. L. Dong and F. Naumann. Data fusion–resolving data conflicts for integration. *PVLDB*, 2009.

[13] J. Dunn. Well separated clusters and optimal fuzzy partitions. *Journal of Cybernetics*, 4:95–104, 1974.

[14] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *TKDE*, 19(1):1–16, 2007.

[15] W. Fan. Dependencies revisited for improving data quality. In *PODS*, pages 159–170, 2008.

[16] W. Fan, X. Jia, J. Li, and S. Ma. Reasoning about record matching rules. *PVLDB*, 2(1):757–768, 2009.

[17] T. F. Gonzalez. On the computational complexity of clustering and related problems. *Lecture Notes in Control and Information Sciences*, 38:174–182, 1982.

[18] S. Guo, X. L. Dong, D. Srivastava, and R. Zajac. Record linkage with uniqueness constraints and erroneous values. http://www.research.att.com/∼lunadong/publication/linkage_techReport.pdf.

[19] I. F. Ilyas, V. Markl, P. J. Haas, P. G. Brown, and A. Aboulnaga. Cords: Automatic generation of correlation statistics in db2. In *VLDB*, 2004.

[20] N. Koudas, A. Saha, D. Srivastava, and S. Venkatasubramanian. Metric functional dependencies. In *ICDE*, 2009.

[21] N. Koudas, S. Sarawagi, and D. Srivastava. Record linkage: similarity measures and algorithms. In *SIGMOD*, 2006.

[22] M. W. Krentel. The complexity of optimization problems. *Lecture Notes in Computer Science*, 223, 1986.

[23] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

[24] C. Legany, S. Juhasz, and A. Babos. Cluster validity measurement techniques. In *WSEAS*, pages 388–393, 2006.

[25] P. Lyman, H. R. Varian, K. Swearingen, P. Charles, N. Good, L. L. Jordan, and J. Pal. How much information? 2003. http://www2.sims.berkeley.edu/research/projects/how-much-info-2003/execsum.htm.

[26] S. Petrovic. A comparison between the silhouette index and the davies-bouldin index in labelling ids clusters. In *NORDSEC*, 2006.

[27] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDBJ*, 10(4):334–350, 2001.

[28] P. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Comp. and Applied Math.*, 20(1):53–65, 1987.

[29] J. Sima and S. E. Schaeffer. On the NPCompleteness of some graph cluster measures. *Lecture Notes in Computer Science*, 3831:530–537, 2006.

[30] A. K. H. Tung, R. T. Ng, L. V. S. Lakshmanan, and J. Han. Constraint-based clustering in large databases. In *ICDT*, 2001.

[31] W. Winkler. Overview of record linkage and current research directions. Technical report, Statistical Research Division, U. S. Bureau of the Census, 2006.

[32] D. Zardetto, M. Scannapieco, and T. Catarci. Effective automated object matching. In *ICDE*, pages 757–768, 2010.

APPENDIX

A. AUGMENTATIONS FOR CLUSTERING

We describe two augmentations that further explore evidence from source data and improve performance of clustering.

Multi-value penalty: Under hard constraints, for each entity a source should provide at most a single value for a uniqueness attribute; however, with an incorrect clustering, the source may look like providing multiple values. We capture this negative evidence by discounting sources that provide multiple values for an attribute when we compute intra-cluster association distance. We first assume that no source violates any hard constraint and relax this assumption later.

Let $\bar{S}^{l,\neg l'}(C_i) \subseteq \bar{S}^{l,l'}(C_i)$ be the sources that also support an edge between an A_l-node in C_i and an $A_{l'}$-node outside C_i (similar for $\bar{S}^{\neg l,l'}(C_i)$). Then, we ignore support from sources in these two subsets in association distance:

$$d_A^{l,l'}(C_i, C_i) = 1 - \frac{|\bar{S}^{l,l'}(C_i)| - |\bar{S}^{l,\neg l'}(C_i) \cup \bar{S}^{\neg l,l'}(C_i)|}{|\bar{S}^{l}(C_i) \cup \bar{S}^{l'}(C_i)|}. \quad (11)$$

Now we relax this assumption and denote by $p(S)$ the probability that source S violates a hard constraint on an entity. Then, when we observe a particular violation by S, with only probability $1-p(S)$ that the violation is caused by a wrong clustering. Taking it into account, we revised Eq.(11) to

$$d_A^{l,l'}(C_i, C_i) = 1 - \frac{|\bar{S}^{l,l'}(C_i)| - \sum_{S \in \bar{S}^{l,\neg l'}(C_i) \cup \bar{S}^{\neg l,l'}(C_i)} (1 - p(S))}{|\bar{S}^{l}(C_i) \cup \bar{S}^{l'}(C_i)|}. \quad (12)$$

In practice, we can assign $p(S)$ according to domain knowledge. Alternatively, we can run our algorithm iteratively, computing $p(S)$ according to the clustering result in each round and then using the new values in the next round, until convergence.

EXAMPLE A.1. *Continue with the motivating example and consider the clustering described in Table 6. If we do not apply the multi-value penalty, this clustering is considered as optimal. In fact, the association distance of C'_4 is $\frac{0+0+0}{3} = 0$, and the DB-index is .236, lower than that of the ideal clustering shown in Figure 1(c) (0.241).*

If we apply Eq. (12) and assume $p(S_i) = .2, i \in [1, 10]$, $\bar{S}^1(C'_4) = \bar{S}^2(C'_4) = \bar{S}^{1,2}(C'_4) = \{s_1, \ldots, s_{10}\}$, $\bar{S}^{1,\neg 2}(C'_4) = \{s_1, \ldots, s_8\}$, and $\bar{S}^{\neg 1,2}(C'_4) = \emptyset$, so $d_A^{1,2}(C'_4, C'_4) = 1 - \frac{10-8\times.8}{10} = 0.64$. Similarly, we have $d_A^{1,3}(C'_4, C'_4) = 0.64$, $d_A^{2,3}(C'_4, C'_4) = 0$. The new intra-cluster association distance of C'_4 becomes $d_A(C'_4, C'_4) = 0.43$ and the DB-index is increased to 0.59, higher than that of the ideal solution, 0.45 (also with multi-value penalty). □

Appearance-similarity based association update: Another augmentation for clustering is to update association based on *appearance similarity*. For many attributes, two value representations that look similar can represent completely different values. For example, ***-*1255* and ***-*2255* differ in only one digit but represent two different phone numbers; similarly, *1 Sylvan Way* and *2 Sylvan Way* differ in only one char but represent two different geographical locations on the same street. We thus distinguish *appearance similarity* from *logical similarity*: the former compares the appearance of two representations (*e.g.*, string similarity, numerical similarity), and the latter indicates the likelihood that the two representations represent the same value. The logical similarity can be defined according to some rules. For example, the logical similarity of two phone numbers is 0 (assume normalization to some standard format); as another example, the logical similarity of two addresses is 0 if they have different numbers, and the same as the string similarity otherwise.

Table 6: An alternative clustering for the motivating example.

	C'_1	C'_2	C'_3	C'_4
N-nodes				N_1,N_2,N_3,N_4
P-nodes	P_1	P_2	P_3	P_4
A-nodes	A_1			A_2,A_3

We should use logical similarity in similarity distance to avoid clustering two attributes that look alike but are logically different. However, appearance similarity can help identify mis-spellings and link records with such errors. As an example, consider two records *(Microsofe Corp., xxx-1255)* and *(Microsoft Corp., xxx-2255)*. If we realize the two phone numbers look similar and one of them might be a mis-spelling, we are more likely to link the records.

Our solution is to update the k-partite graph according to appearance similarity. In particular, for each edge (v_i, v_j), we associate each of its support sources with a number indicating the likelihood of support, denoted by $P(S, (v_i, v_j)) \in [0, 1]$. If S provides a record with v_i and v_j, denoted by $(v_i, v_j) \in S$, $P(S, (v_i, v_j)) = 1$; otherwise,

$$P(S, (v_i, v_j)) = \max\{\max_{v'_j \in V_j, (v_i, v'_j) \in S, \text{sim}_L(v_j, v'_j) = 0} \text{sim}_A(v_j, v'_j), \max_{v'_i \in V_i, (v'_i, v_j) \in S, \text{sim}_L(v_i, v'_i) = 0} \text{sim}_A(v_i, v'_i)\}, \quad (13)$$

where we denote by sim_A the appearance similarity and by sim_L the logical similarity.

Now when we compute the intra-cluster association distance, instead of counting the number of sources that support an edge, we sum up their support likelihood.

EXAMPLE A.2. *Consider the three 2-partite graphs in Figure 7, each with two nodes N_1 and N_2 for name, two nodes P_1 and P_2 for phone, and two (solid-line) edges with the same set of support sources. The logical similarity between N_1 and N_2 (the same as the appearance similarity) is represented by a solid arc, and the appearance similarity between P_1 and P_2 is represented by a dotted arc (the logical similarity is 0). We observe that in (a), both names and phones are highly similar; in (b) and (c), only names or phones are highly similar. We apply association update on the graphs and represent the new edges by dotted lines.*

Obviously, not considering appearance similarity between phones lead to the same clustering for (a) and (b). Instead, if we update the associations by adding the dotted edges in (a), we cluster N_1 and N_2 with P_1. Clustering for (c) shows that even if P_1 and P_2 are highly similar, we shall not cluster N_1 and N_2 if they have low similarity (the clustering in (c) has DB-index 0.88 and clustering N_2 with N_1 increases the DB-index to 2.57). Finally, if we use appearance similarity instead of logical similarity for similarity distance, we will wrongly cluster P_1 and P_2 in (a) and (c), though they represent different phone numbers (values). □

B. ALGORITHM DETAILS

See Algorithm CLUSTER for clustering under hard constraints and Algorithm MATCH for extension to deal with soft constraints.

C. EXTENSIONS

We next describe extensions for some special cases.

Non-uniqueness attribute: In presence of non-uniqueness attributes, we combine them with the identifier to form a *super-identifier*. For

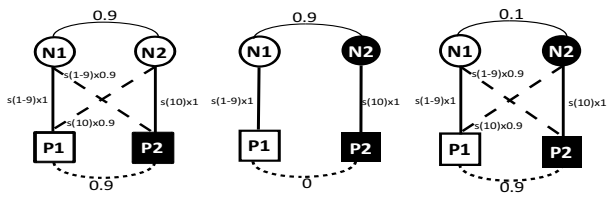


Figure 7: Three 2-partite graphs. Nodes of the same color belong to the same cluster in the clustering results.

Algorithm 1 CLUSTER

Input: k-partite graph $G(\mathcal{S}) = (V_1, \ldots, V_k, E)$.
Output: $\mathcal{C}(G)$, the clustering of G.

$\mathcal{C}^{(0)}(G)$ = INITIALIZE(G); //Generate initial clustering
$t = -1$;
repeat
  $t = t + 1$;
  $\mathcal{C}^{(t+1)}(G) = \mathcal{C}^{(t)}(G)$;
  for $i = 1, k$ **do**
    for all $v \in V_i$ **do**
      for all cluster $C_j \in \mathcal{C}^{(t+1)}(G)$ **do**
        derive $\mathcal{C}_{temp}$ by moving v to C_j;
        $Score(C_j)$ =DAVIESBOULDIN($\mathcal{C}_{temp}$);
      end for
      adjust $\mathcal{C}^{(t+1)}(G)$ by moving v to C_j with the lowest score;
    end for
  end for
until $(\mathcal{C}^{(t)}(G) == \mathcal{C}^{(t+1)}(G))$
return $\mathcal{C}^{(t)}(G)$;

Algorithm 2 MATCH

Input: $G(\mathcal{S})$, the k-partite graph encoding of $\mathcal{S}$.
Output: $\mathcal{E}$, entities provided by $\mathcal{S}$.

$\mathcal{C}(G(\mathcal{S}))$ =CLUSTER($G(\mathcal{S})$);
$G^c(\mathcal{S})$ =TRANSFORM($\mathcal{C}(G(\mathcal{S}))$);
for all soft uniqueness attribute A **do**
  $\bar{E}_A = \{$edges between A and $A_\kappa\}$;
  compute weight of each edge in $\bar{E}_A$;
  rank the edges in $\bar{E}_A$ in a descending order according to edge weight;
  //Phase I
  compute $LS_2(v)$ for each node v of A and A_κ
  select nodes of A with the top $|A|(p_A+\epsilon)$ values of $LS_2(v)$; similar for A_κ;
  compute $\deg(v)$ for each selected node;
  //Phase II
  $\bar{M}_A = \emptyset$;
  for all edge $e \in \bar{E}_A$ **do**
    if adding e to $\bar{M}_A$ does not violate degree constraints and increases score of $\bar{M}_A$ **then**
      add e to $\bar{M}_A$;
    end if
  end for
end for
//Generate entities
$\mathcal{E} = \emptyset$;
for each value v of A_κ in $G^c(\mathcal{S})$ **do**
  add an entity to $\mathcal{E}$ with v and all values matched to v (or clustered w. v for values of hard uniqueness attribute);
end for
return $\mathcal{E}$;

example, we can combine name and category together as a super-identifier, and ignore different values of category by considering (name1, cat1) and (name1, cat2) as different representations of the identifier of a business. We can apply record-linkage techniques for computing similarity of super-identifier values, including using weighted similarity combination, decision tree, etc.; again, the choice is orthogonal to our techniques. Multi-attribute identifier can be handled similarly.

Multi-value representation: In some contexts a representation can represent multiple values (*e.g.*, abbreviated person name). We treat such an attribute as soft uniqueness attribute and such a representation may be matched to multiple entities.

D. EXPERIMENTS ON SYNTHETIC DATA

To understand how CLUSTER and MATCH perform on data of different characteristics, we experimented on synthetic data. We next describe data generation and experimental results for the case where we have only hard constraints and for the case where there also exist soft constraints.

D.1 Clustering w.r.t. hard constraints

D.1.1 Data generation

We assumed there are only hard constraints and considered $m = 4$ entities, each with three attributes: N, the key attribute; P, for which logical similarity between different values is 0; and A, for which logical and appearance similarity are the same (as we describe later, in some experiments we added or removed attributes). For each attribute, we generated (correct) values as follows: we first randomly choose a number in $[0, l]$ for the first entity, where l is the size of the domain; then iteratively generate the next value as $v' = (v + \frac{l \cdot var}{m}) \mod l$, where v is the previous value, and var controls distance between two neighbor values. We set $l = 10M$ and varied var in experiments. Table 7 shows ranges and default values of parameters we used in data generation.

Table 7: Data-generation parameters and their settings.

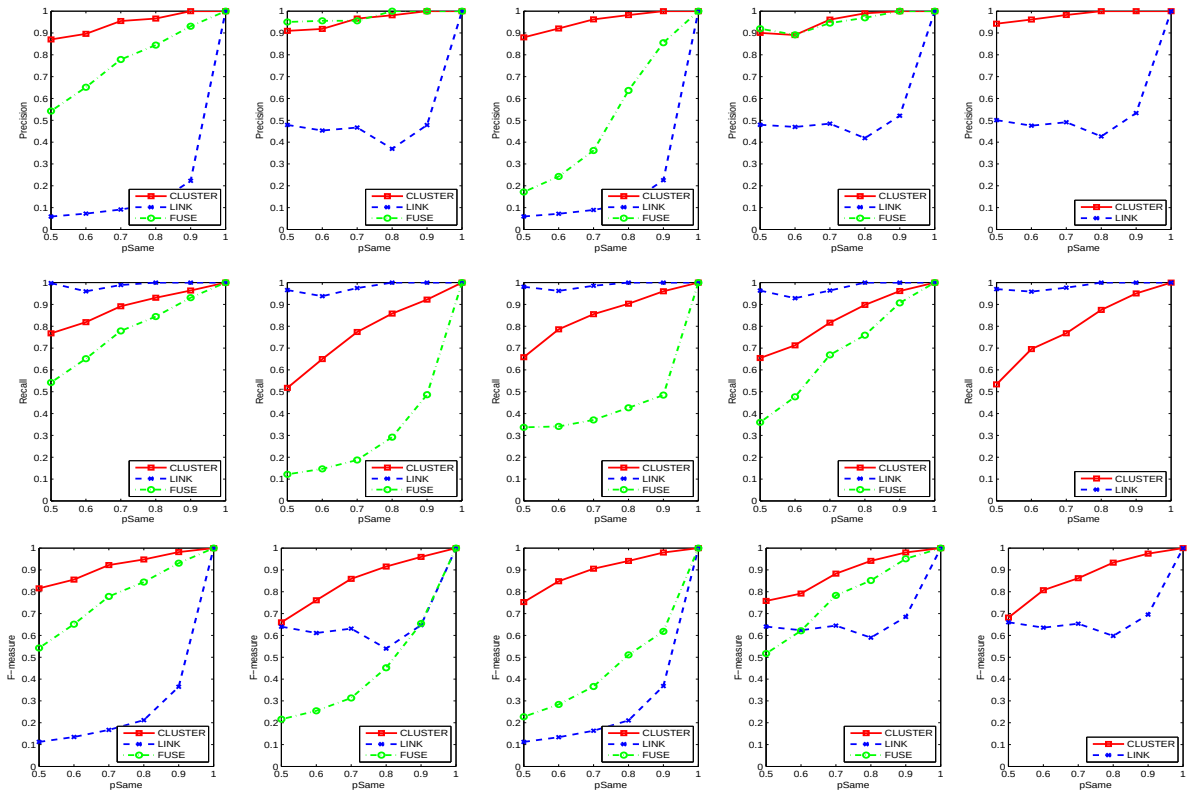
Parameter	var	$pSame$	$pExist$	$noise$
Default	.5	.8	.1	.05
Range	.1-1	.5-1	0-1	0-.1

We generated 20 data sources. For each entity, each source has probability .8 to provide a record. For each attribute of each entity, we perturbed the provided value as follows. Let c be the correct value. With probability $pSame$ a source provides c; otherwise, it provides a wrong value: with probability $pExist$ it provides the correct value of another entity, and otherwise it provides a random value in $[c-l\cdot noise, c+l\cdot noise] \cap [0, l]$. We always set $pExist = 0$ for the key attribute N.

Our data generation naturally implies the golden standard. For each parameter setting, we ran the experiment 10 times and reported average precision, recall, and F-measure. We computed the similarity between two values v and v' by $1 - \frac{|v-v'|}{l}$. Note that according to our data generation, the similarity between values is quite high: when $var = .5$ (default), the similarity between two neighbor correct values is already .875. Our algorithm thus sets similarity that is below .95 to 0. We applied Eq. (11) for multi-value penalty.

D.1.2 Results

Overall performance: Figure 8 shows results when we vary $pSame$ for all attributes. We have three observations. First, CLUSTER performs well and consistently better than the other methods: when $pSame = .8$ (default), it has F-measures of above .9 for matching and above .92 for clustering; when $pSame = .5$ and so the records for each entity vary highly, CLUSTER still obtains a



(a) N-P matching (b) N-A matching (c) A-P matching (d) N clustering (e) A clustering

Figure 8: Accuracy with schema (N, P, A). CLUSTER obtains the best results in most cases. FUSE does not cluster A-values, so we do not plot its accuracy on A-clustering.

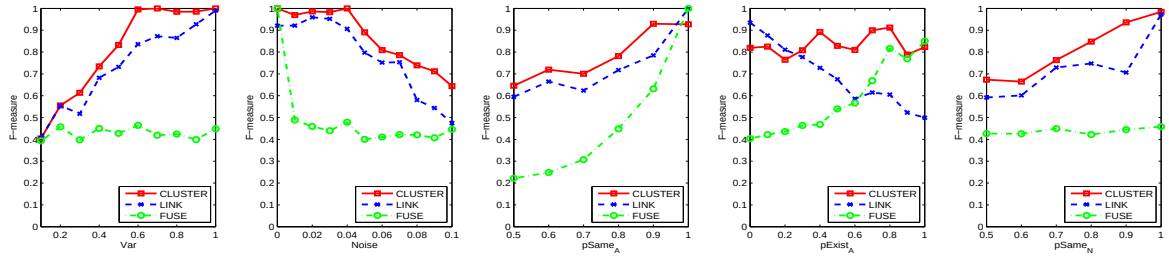


Figure 9: Effect of parameters on matching (N-A). CLUSTER obtains the best results in most settings.

F-measure of .60 on average, 37% higher than LINK and 133% higher than FUSE. Second, CLUSTER performs especially well with respect to the P attribute, whose different values are considered logically different: when $pSame = .5$, the F-measure of N-P matching is .81, whereas that of N-A matching is .66. Third, LINK typically obtains the lowest precision as it does not identify false values, and FUSE typically obtains the lowest recall, as it does not consider similarity of N-values.

Effect of parameters: To examine effect of data-generation parameters on the performance, we experimented on data sets with only two attributes (N, A). Figure 9 shows F-measure of matching when we vary different parameters and we observed very similar pattern on clustering.

Again, CLUSTER obtains the best results in most cases, whereas FUSE, which does not explicitly cluster representations, obtains the worst results. In particular, we consider the parameters in three groups. First, var and $noise$ control similarity of generated values (correct or perturbed), so has a big effect on accuracy of CLUSTER and LINK: the higher var and the lower $noise$, the better results. Note that LINK is more sensitive to $noise$, as it does not identify false values and so is more likely to cluster an erroneous record with records for other entities when $noise$ is high. Second, $pSame_A$ and $pExist_A$ control variety and correctness of A-values. We observe that CLUSTER is quite stable w.r.t. them as it considers both errors and variety of representations; LINK performs worse when $pExist_A$ is high (so more errors); FUSE performs worse when $pExist_A$ is low (so higher variety of representations), and is sensitive to $pSame_A$ (indeed, FUSE is only affected by these two parameters). We note that CLUSTER does not obtain a F-measure of 1 when $pSame_A = 1$, because it may mis-cluster names that look very alike. Third, $pSame_N$ controls variety of N-values (recall that $pExist_N = 0$); the higher $pSame_N$, the higher F-measure of CLUSTER and LINK.

Contribution of different components: We conducted similar experiments as on real-world data for further understanding how different components contribute to our performance. We experimented

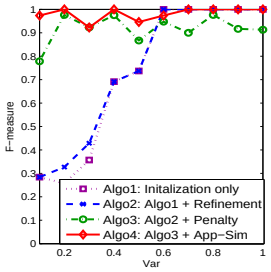


Figure 10: Contribution of components on matching (N-P).

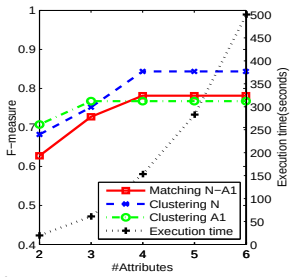


Figure 11: Contribution of attributes.

on (N, P), as the logical similarity and appearance similarity differ for P. Figure 10 shows F-measure of matching when we vary var and we observed similar pattern for clustering. We observe that (1) simply applying clustering without the augmentations improves the results over initial clustering only slightly, if any; (2) applying multi-value penalty makes significant improvement when var is small and increases robustness with respect to var; and (3) considering appearance-similarity can further improve the results (by 8% on average).

Effect of attributes: We also examined how the number of uniqueness attributes can affect our performance. We started with data of schema (N, A) and then added more uniqueness attributes whose logical and appearance similarity are the same. Figure 11 shows the F-measure of matching and clustering. We observe clear improvement when the number of uniqueness attributes increases until it reaches 4; after that, adding more uniqueness attributes has only slight benefit. Finally, our algorithm finishes in 8.35 minutes when there are 6 uniqueness attributes; this is adequate given that linkage and fusion are often conducted offline.

D.2 Considering soft constraints

D.2.1 Data generation

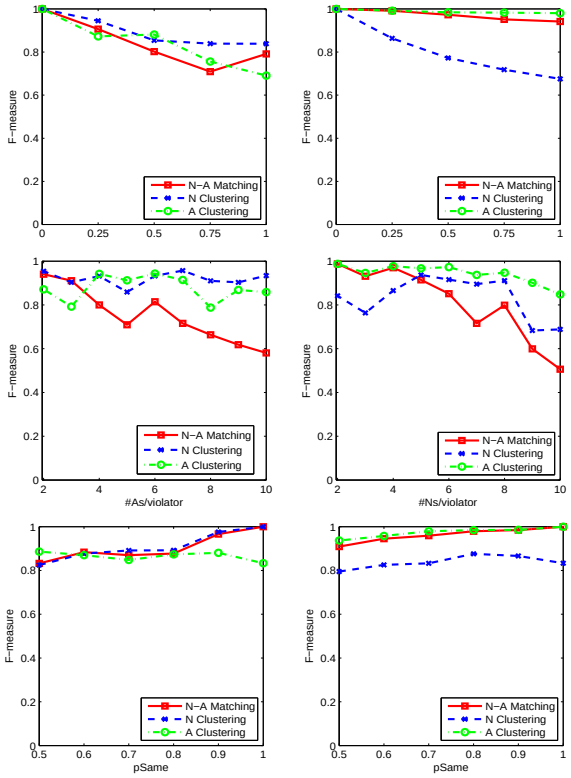
We considered $m = 4$ entities, each with two attributes: N (the key) and A. We considered violations of uniqueness constraints on each side and control them using two sets of parameters: violation rate ($p_{N\to A}$ and $p_{A\to N}$) and number of associated values per violator ($\#A$s/N-violator and $\#N$s/A-violator). We ranged the former from 0 to 1 and the latter from 2 to 10; we set their default values to .25 and 2 respectively. We generated the standard (true) values and source data as described in Section D.1.1 and used the default values in Table 7 for other parameters.

Our algorithm computes the similarity between perturbed values of the same standard value as before, but sets it to 0 between perturbations of different standard values (as we can have up to 13 standard values for each attribute, two neighbor values can be very close). We applied Eq. (12) for multi-value penalty and set $p(S) = .9$ for each source (using .75 obtained similar results).

D.2.2 Results

Figure 12 shows the performance of MATCH when we varied different parameters. We have the following observations.

- To a certain extent, MATCH handles soft constraints quite well. For $N \to A$ violations (similar for $A \to N$ violations), the F-measures of both matching and clustering are above .8 when $p_{N\to A}$ is up to .5 and $\#A$s/N-violator is 2, and above .7 when $p_{A\to N}$ is .25 and $\#A$s/N-violator is up to 5.
- MATCH is fairly tolerant to the number of violations and the number of values each violator is associated with. For $N \to A$ violations (similar for $A \to N$), when $p_{N\to A}$ increases to 1 and $\#A$s/N-violator is 2, we still obtain a F-measure of above .6 in matching and in clustering. When $\#A$s/N-violator is increased to 10, the F-measure of clustering is above .8 and that of matching is around .6. Note that inaccurate clustering of A-values is mainly because of incorrectly clustering different A-values of the same entity; also note that the drop in matching performance is mainly because some perturbed values of an N-violator are not merged with the standard value, so not matched with the multiple A-values and cause a big penalty on recall. Finally, although a lower $pSame$ can lead to a lower F-measure, the difference is very small.
- Because we first consider N-nodes and then A-nodes in clustering, we obtained better results with $A \to N$ violations than with $N \to A$ violations; but if we change the order, the pattern of the results also switch.



(a) $N \to A$ violation (b) $A \to N$ violation

Figure 12: Performance of MATCH in presence of soft constraints. Our algorithm handles violations quite well.

D.3 Summary

We summarize observations from experiments as follows.

1. Our techniques significantly improve over traditional techniques on a variety of data sets and are less sensitive on the variety of representations and erroneous values.
2. Applying multi-value penalty is critical, and appearance-similarity based association update further improves our algorithm.
3. The accuracy of our techniques increases with the number of uniqueness attributes.
4. Our algorithm handles soft constrains well and is fairly stable.
5. Our algorithm scales well.