Reference Reconciliation in Complex Information Spaces

Xin Dong
University of Washington
Seattle, WA 98195
lunadong@cs.washington.edu

Alon Halevy
University of Washington
Seattle, WA 98195
alon@cs.washington.edu

Jayant Madhavan
University of Washington
Seattle, WA 98195
jayant@cs.washington.edu

ABSTRACT

Reference reconciliation is the problem of identifying when different references (i.e., sets of attribute values) in a dataset correspond to the same real-world entity. Most previous literature assumed references to a *single* class that had a fair number of attributes (e.g., research publications). We consider complex information spaces: our references belong to *multiple* related classes and each reference may have very few attribute values. A prime example of such a space is Personal Information Management, where the goal is to provide a coherent view of all the information on one's desktop.

Our reconciliation algorithm has three principal features. First, we exploit the associations between references to design new methods for reference comparison. Second, we propagate information between reconciliation decisions to accumulate positive and negative evidences. Third, we gradually enrich references by merging attribute values. Our experiments show that (1) we considerably improve precision and recall over standard methods on a diverse set of personal information datasets, and (2) there are advantages to using our algorithm even on a standard citation dataset benchmark.

1. INTRODUCTION

One of the major impediments to integrating data from multiple sources, whether by warehousing, virtual integration or web services, is resolving references at the instance level. Data sources have different ways of referring to the *same* real-world entity. Variations in representation arise for multiple reasons: mis-spellings, use of abbreviations, different naming conventions, naming variations over time, and the presence of several values for particular attributes. To join data from multiple sources, and therefore, to do any kind of analysis, we must detect when different references refer to the same real-world entity. This problem is known as *reference reconciliation*.

Reference reconciliation has received significant attention in the literature, and its variations have been referred to as record linkage [37], merge/purge [21], de-duplication [34], hardening soft databases [9], reference matching [27], object identification [35] and identity uncertainty [26]. Most of the previous work considered techniques for reconciling references to a *single* class. Furthermore, typically the data contained many attributes with each instance. However, in practice, many data integration tasks need to tackle complex information spaces where instances of multiple classes and rich relationships between the instances exist, classes may have only few attributes, and references typically have unknown attribute values.

The main motivation for our work comes from the application of Personal Information Management (PIM), and specifically, supporting higher-level browsing of information on one's desktop. The need for better search tools for the desktop was highlighted by systems like SIS [16] and the Google Desktop search tool [19]. However, these systems provide only keyword search to the desktop's contents. The vision of the Personal Memex [5] and recent systems such as Haystack [33] and Semex [13] emphasize the ability to browse personal information by *associations*, representing important semantic relationships between objects. To support such browsing, a PIM system examines data from a variety of sources on the desktop (e.g., mails, contacts, files, spreadsheets) to extract instances of multiple classes: Person, Message, Article, Conference, etc. In addition, the system extracts associations between the instances, such as senderOf, earlyVersionOf, authorOf, and publishedIn, which then provide the basis for browsing. However, since the data sources are heterogeneous and span several years, a real-world object is typically referred to in several different ways. Reconciliation of the above classes of references guarantees that they mesh together seamlessly, and so the PIM system can provide palatable browsing and searching experiences.

Aside from PIM, many reference reconciliation problems involve complex relationships between multiple entities. For example, most of the work on reconciling publications for portals such as Citeseer [7] and Cora [25] has focused solely on the publications. To offer a more powerful portal, we would also like to reconcile authors, institutions, publishers and conferences. In fact, reconciling the other entities can improve the accuracy of publication reconciliation. As another example, reconciling references in online product catalogs is a critical problem in electronic commerce. The reconciliation decision in this context involves the product entities, their manufacturers, suppliers, orders and related products.

To date, most reference reconciliation approaches (see [20, 4] for recent surveys) employ approximate string similarity measures and combine the similarities of multiple attributes. Though we can apply these methods to each type of references in isolation, we miss the rich information carried in the relationships between the instances. Furthermore, the previous techniques assume there are several attributes associated with every reference, and therefore a reasonable amount of information to consider in the reconciliation decision. In some contexts, this does not hold. For example, in PIM, a person reference extracted from an email may contain only an email address, and in Citeseer, a person reference extracted from a citation contains only a name, which may even be abbreviated.

We describe a novel reference reconciliation algorithm that is well suited for complex information spaces and for cases where some of the references may lack information. The key ideas underlying our approach are the following. First, we make extensive use of *context information* (the associations between references) to provide evidence for reconciliation decisions. For example, given two references to persons, we will consider their co-authors and email contacts, to help decide whether to reconcile them. Second, we *propagate* information between reconciliation decisions for different pairs of references. For example, when we decide to reconcile two papers, we obtain additional evidence for reconciling the person references to their authors. This, in turn, can further increase the confidence in reconciling other papers authored by the reconciled persons. Third, we address the lack of information in each reference by *reference enrichment.* For example, when we reconcile two person references, we gather the different representations of the person's name, collect her different email addresses, and enlarge her list of co-authors and email-contacts. This enriched reference can later be reconciled with other references where we previously lacked information for the reconciliation.

Our algorithm is based on propagating reference-similarity decisions in a *dependency graph* whose nodes represent similarities between pairs of references, and whose edges represent the dependencies between the reconciliation decisions. Our framework captures the dependence between different reference pair reconciliations and at the same time retains the flexibility of using established reference comparison techniques. The result is a reconciliation algorithm that improves over the recall of previous techniques without sacrificing precision.

The specific contributions of this paper are the following.

- We describe a novel reference reconciliation algorithm, based on a general framework for propagating information from one reconciliation decision to another. Our algorithm uses context information, similarities computed on related entities, and enriched references. In addition, we show how our algorithm exploits knowledge that two references are guaranteed to be *distinct.*
- We evaluate our reconciliation algorithm on several personal information datasets and on the Cora citation dataset. In the PIM context, the experiments show that our approach significantly improves on standard reference reconciliation techniques. On the Cora dataset, the results show that our approach is comparable to recent adaptive approaches for paper reconciliation, and at the same time produces accurate reconciliation on person and publisher instances.

Person (name, email, *coAuthor, *emailContact)
Article (title, year, pages, *authoredBy, *publishedIn)
Conference (name, year, location)
Journal (name, year, volume, number)

(a) Personal Information Schema

Article a_1=({"Distributed query processing in a relational data base system"}, {"169-180"}, $\{p_1, p_2, p_3\}$, $\{c_1\}$)
a_2=({"Distributed query processing in a relational data base system"}, {"169-180"}, $\{p_4, p_5, p_6\}$, $\{c_2\}$)
Person p_1=({"Robert S. Epstein"}, *null*, $\{p_2, p_3\}$, *null*)
p_2=({"Michael Stonebraker"}, *null*, $\{p_1, p_3\}$, *null*)
p_3=({"Eugene Wong"}, *null*, $\{p_1, p_2\}$, *null*)
p_4=({"Epstein, R.S."}, *null*, $\{p_5, p_6\}$, *null*)
p_5=({"Stonebraker, M."}, *null*, $\{p_4, p_6\}$, *null*)
p_6=({"Wong, E."}, *null*, $\{p_4, p_5\}$, *null*)
p_7=({"Eugene Wong"}, {"eugene@berkeley.edu"}, *null*, $\{p_8\}$)
p_8=(*null*, {"stonebraker@csail.mit.edu"}, *null*, $\{p_7\}$)
p_9=({"mike"}, {"stonebraker@csail.mit.edu"}, *null, null*)
Conf c_1=({"ACM Conference on Management of Data"}, {"1978"}, {"Austin, Texas"})
c_2=({"ACM SIGMOD"}, {"1978"}, *null*)

(b) Raw References

$\{\{a_1, a_2\}, \{p_1, p_4\}, \{p_2, p_5, p_8, p_9\}, \{p_3, p_6, p_7\}, \{c_1, c_2\}\}$

(c) Reconciliation Results

Figure 1: The (a) schema, (b) references, and (c) reconciliation results in Example 1.

Section 2 defines the reconciliation problem and illustrates our approach. Section 3 describes the dependency-graph framework, and Section 4 describes the computation of similarities for each class. Section 5 presents our experiments. Section 6 describes related work and Section 7 concludes.

2. OVERVIEW

We begin by defining the reference reconciliation problem, and then we illustrate our approach with an example from the application of personal information management.

2.1 Problem definition

We model a domain with a schema that includes a set of *classes*, each of which has a set of *attributes*. We distinguish between two kinds of attributes: those whose values are of a simple type such as string and integer, called *atomic* attributes; and those whose values are links to other instances, called *association* attributes.

Ultimately, our goal is to populate the schema such that each instance of a class refers to a single real-world entity, and each real-world entity is represented by at most a single instance. However, what we are given as input are *references* to real-world objects obtained by some extractor program. Each reference partially specifies an instance of a particular class: it has a set of values (possibly empty set) for each attribute of that class.

The reconciliation algorithm tries to partition the set of references in each class, such that each partition corresponds to a single unique real-word entity, and different partitions refer to different entities. We measure the quality of a reconciliation with recall and precision. The recall measures the percentage of correctly reconciled pairs of references over all pairs of references that refer to the same entity, and the precision measures the percentage of correctly reconciled pairs over all reconciled pairs of references.

Example 1. Figure 1(a) shows a subset of a schema for a personal information management application. The schema contains four classes: Person, Article, Conference and Journal, each with a particular set of attributes. The association attributes are denoted with "*". As an example, the Person class has two atomic attributes, name and email, and two association attributes, emailContact and coAuthor, whose values are links to other Person instances. These associations link a Person instance with other Person instances that they have exchanged emails with, or have co-authored with.

Figure 1(b) shows a set of references extracted from a personal dataset. The Article references a_1 and a_2, Person references p_1 to p_6, and Conference references c_1 and c_2 are extracted from two Bibtex items. The other three Person references, p_7 to p_9, are extracted from emails. As an example, the Person reference p_7 is named "Eugene Wong" and has email address "eugene@berkeley.edu". The reference does not include any co-authors, but includes an email contact–Person p_8.

Figure 1(c) shows the ideal reconciliation result for the references in Figure 1(b). □

2.2 Overview of the approach

The goal of our approach is to address several shortcomings of existing reference reconciliation algorithms that preclude their use in applications such as PIM. First, it is often the case that each reference includes very limited information, i.e., contains values for only a few atomic attributes. For example, a Person reference often has values for only one or two atomic attributes. In Example 1, references p_5 and p_8 even do not have any attributes in common. Second, some attributes are multi-valued, so the fact that two attribute values are different does not imply that the two references refer to different real-world objects. For example, two Person references with completely different email addresses may refer to the same person. This phenomenon is especially common in applications where the real-world entities (and hence, the data) evolve over time.

As we show later, the above problems lead to unsatisfactory accuracy of existing reference reconciliation algorithms. The key behind our approach is that we exploit the richness of the information space at hand. We illustrate the main concepts of our algorithm with the example.

Exploiting context information: The main idea underlying our algorithm is to capture and leverage various forms of *context* we can glean about the references, which are not considered by traditional reference reconciliation approaches. For example, we can consider the co-author lists and email-contact lists of person references. In our example, we notice that p_5 has co-authored articles with p_6, and p_8 has email correspondences with p_7. If we decide that p_6 and p_7 are the same person, we obtain additional evidence that may lead us to reconcile p_5 and p_8. Second, we compare values of *different* attributes. For example, the name "Stonebraker, M." and the email address "stonebraker@csail.mit.edu" are closely related: "stonebraker" corresponds to the last name of "Stonebraker, M.". This observation provides positive evidence for merging references p_5 and p_8.

Once we decide to merge two references, we have two mechanisms for leveraging this information: reconciliation propagation and reference enrichment.

Reconciliation propagation: When we reconcile two references, we reconsider reconciling references that are related to them via association attributes. For example, when we detect that the Article references a_1 and a_2 share the same title and similar authors and that they appeared in similar conferences and pages in the proceedings, we decide to reconcile them. Presumably an article has a unique set of authors, so the reconciliation of a_1 and a_2 implies that the authors p_1 and p_4, p_2 and p_5, and p_3 and p_6 should be reconciled respectively. Similarly, we reconcile the conference references c_1 and c_2.

Reference enrichment: When we decide to reconcile two references, we join their attribute values together, thereby enrich the reference. For example, consider the reconciliation of the person references p_5 and p_8. Although "stonebraker@csail.mit.edu" and "Stonebraker, M." are rather similar, this information is insufficient for reconciliation. Similarly, we lack evidence for reconciling p_5 and p_9. However, after we reconcile p_8 and p_9, we can aggregate their information and now know that "mike" and "Stonebraker, M." share the same first name initial, and contact the same person by email correspondence or coauthoring. This additional information will enable us to reconcile p_5, p_8, and p_9.

In summary, our approach obtains better reference reconciliation results by exploiting context information, propagating reconciliation decisions and enriching references. In addition, our algorithm exploits information about two references being *distinct*. To facilitate these mechanisms, we define a graph, called the *dependency graph*, that captures the dependencies of reference similarities and attribute value similarities. In the next section, we describe how we construct the dependency graph and use it for reconciliation.

3. RECONCILIATION ALGORITHM

Our reconciliation algorithm proceeds as follows. First, we construct the dependency graph that captures the relationships between different reconciliation decisions. Then, we iteratively re-compute scores that are assigned to reconciliation decision nodes in the graph until a fixed point is reached. Finally, we compute the transitive closure for the final reconciliation results. Section 3.1 describes the construction of the dependency graph, and Section 3.2 describes how we use the graph for reconciliation. Section 3.3 describes how we enrich references during the reconciliation process, and Section 3.4 describes how our algorithm exploits *negative* information to further improve the accuracy of reconciliation.

3.1 Dependency graph

To reconcile references, we need to compute the similarity for every pair of references of the same class; this similarity is based upon the similarity of their atomic attribute values and that of their association attributes. A node in the graph represents the similarity between a pair of references, and an edge represents the *dependency* between a pair of similarities, e.g., the similarity of a pair of references depends on the similarity of their respective attributes and vice versa. If we change the similarity value of a node, we may need to recompute the similarity of its neighbors. Formally, we define a dependency graph as the following:

DEFINITION 3.1. *Let $\mathcal{R}$ be a set of references. The* dependency graph *for $\mathcal{R}$ is an undirected graph $G = (N, E)$, such that*

- *For each pair of references $r_1, r_2 \in \mathcal{R}$ of the same class, there is a node $m = (r_1, r_2)$ in G.*

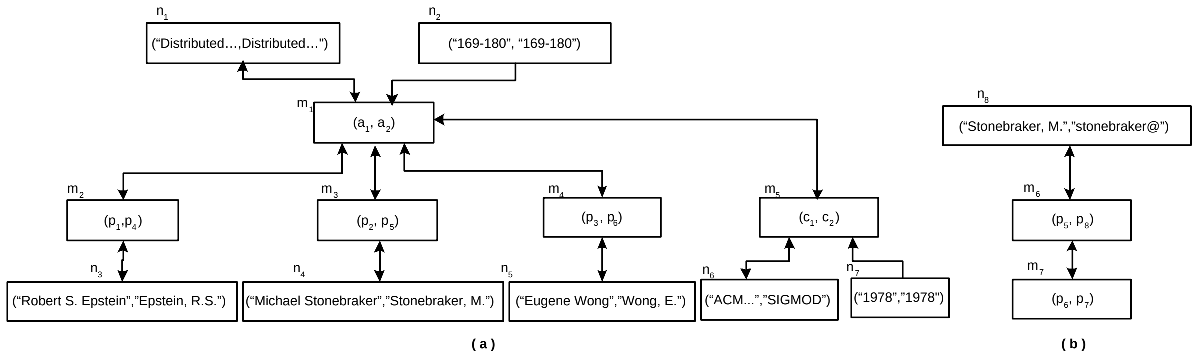
n1
("Distributed...,Distributed...")
n2
("169-180", "169-180")
m1
(a1, a2)
m2
(p1,p4)
m3
(p2, p5)
m4
(p3, p6)
m5
(c1, c2)
n3
("Robert S. Epstein","Epstein, R.S.")
n4
("Michael Stonebraker","Stonebraker, M.")
n5
("Eugene Wong","Wong, E.")
n6
("ACM...","SIGMOD")
n7
("1978","1978")
(a)
n8
("Stonebraker, M.","stonebraker@")
m6
(p5, p8)
m7
(p6, p7)
(b)

Figure 2: The dependency graph for references in Figure 1(b). (a) The subgraph for references $a_1, a_2, p_1, p_2, p_3, p_4, p_5, p_6, c_1$ and c_2. The similarity of papers a_1 and a_2 is represented by the node m_1; it is dependent on the similarity of the titles (represented by n_1), the pages (n_2), the authors (m_2, m_3 and m_4), and the conferences (m_5). Note that there does not exist an edge from m_5 to n_7, because the similarity of years is predetermined and is independent of the similarity of any conferences. Also note that there is no node (p_1, p_5), because their name attributes have very different values. (b) The subgraph for references p_5 and p_8. Given that p_5 has coauthored with p_6, and p_8 has email correspondence with p_7, there is a node $m_7 = (p_6, p_7)$, and a bi-directional edge between m_6 and m_7. Note that m_6 does not have a neighbor (p_4, p_7), because p_4 and p_7 do not have any similar attributes and so the node (p_4, p_7) does not exist.

- *For each pair of attribute values a_1 of r_1 and a_2 of r_2 (the two attributes may be of different types), there is a node $n = (a_1, a_2)$ in G and an edge between m and n in E.*
- *Each node has a real-valued* similarity score *(between 0 and 1), denoted interchangeably as $sim(r_1, r_2)$ or $sim(m)$.* □

In what follows, we refer to references and attribute values collectively as *elements*. Note that there is a unique node in the dependency graph for each pair of elements. This uniqueness is crucial for exploiting the dependencies between reconciliation decisions (as we will see in Section 3.2).

Pruning and refining the graph: In practice, building similarity nodes for *all* pairs of elements is unnecessarily wasteful. Hence, we only include in the graph nodes whose references *potentially* refer to the same real-world entity, or whose attribute values are *comparable* (i.e., are of the same attribute, or according to the domain knowledge are of related attributes, such as a name and an email) and similar.

Rather than considering all edges in the graph to be the same, we refine the set of edges in the graph to be of several types, which we will later leverage to gain efficiency. This refinement can be obtained by applying domain knowledge, either specified by domain experts or learned from training data. The first refinement generates a subgraph of the original graph. In the subgraph, there is an edge from node n to m only if the similarity of m truly depends on the similarity of n. We call n an *incoming neighbor* of m, and m an *outgoing neighbor* of n. Note that the subgraph is directed.

The second refinement distinguishes several types of dependencies, as follows. First, we distinguish between *boolean-valued* dependencies and *real-valued* dependencies. If the similarity of a node n depends *only* on whether the references in the node m are reconciled, then we say that m is a *boolean-valued* neighbor of n. In contrast, if the similarity of n depends on the actual similarity value of node m, then we call m a *real-valued* neighbor of n. As an example, given two conference references, c_1 and c_2 in Figure 1(b), their similarity depends on the real similarity value of their names. While it also depends on the similarity of the articles a_1 and a_2, what really matters is whether they are reconciled, and not their actual similarity value (we assume that a single article cannot be published in two different conferences).

We further divide *boolean-valued* neighbors into two categories. If the reconciliation of m's two references implies that the two references in n should also be reconciled, m is called n's *strong-boolean-valued* neighbor. The second case in the earlier example illustrates a strong-boolean-valued neighbor. If the reconciliation of m's references only increases the similarity score of n, but does not directly imply reconciliation, m is called n's *weak-boolean-valued* neighbor. For example, the similarity score of two persons will increase given that they have email correspondence with the same person.

Constructing the graph: We build the graph in two steps. In the first step we consider atomic attributes, in the second step we consider association attributes.

1. For every pair of references r_1 and r_2 of the same class, insert $m = (r_1, r_2)$ with similarity 0. Further,

   (1) For every pair of atomic attribute values a_1 of r_1 and a_2 of r_2, if a_1 and a_2 are comparable, then proceed in two steps.

   Step 1. If $n = (a_1, a_2) \notin G$, we compute the similarity score of a_1 and a_2. If the score indicates that a_1 and a_2 are potentially similar, then insert n with the computed score.

   Step 2. Insert an edge from n to m and an edge from m to n when the corresponding dependency exists.

   (2) If m does not have any neighbors, remove m and its associated edges.

   Note that at this stage we are liberal in identifying potentially similar atomic attribute pairs (we use a relatively low similarity threshold) in order not to lose important nodes in the graph.

2. We now consider references r_1 and r_2, where $m = (r_1, r_2) \in G$. For every pair of associated attribute values a_1 of r_1 and a_2 of r_2, if given the existence of $n = (a_1, a_2)$, there exists dependence between m and n, we do as follows:

- if $a_1 = a_2$, we add the node $n = (a_1, a_1)$ if it does not exist, and add an edge from n to m;
- if $a_1 \neq a_2$ and the node $n = (a_1, a_2)$ exists, add an edge from n to m and an edge from m to n when the corresponding dependency exists.

We require that $m = (r_1, r_2) \in G$ based on the assumption that two references cannot refer to the same entity unless they have *some* similar atomic attributes values. (However, this assumption is not germane to our algorithm.)

As an illustration, the dependency graph for the instances in Figure 1(b) is shown in Figure 2.

3.2 Exploiting the dependency graph

Our algorithm is based on propagating similarity decisions from node to node in the graph. For example, after we decide to reconcile articles a_1 with a_2, we should reconcile their associated conferences c_1 and c_2, and further trigger re-computation of the similarities of other papers that mention the conferences c_1 and c_2, etc. Given that the dependency graph has captured all the dependencies between similarities, it guides the recomputation process, as we now describe. We describe similarity-score functions in Section 4.

We mark the nodes in a dependency graph as *merged, active*, or *inactive*. A node is marked *merged* when its similarity score is above a pre-defined merge-threshold, and it thus represents already reconciled references. A node is marked *active* if we need to reconsider its similarity. The rest are marked *inactive*. The algorithm proceeds as follows, until no node is marked *active*.

1. Initially, all nodes representing the similarity between references are marked *active*. The nodes representing the similarity between atomic attribute values are marked *merged* or *inactive* depending on their associated similarity score.
2. We select one active node at a time and recompute its similarity score. If the new similarity score is above a merge-threshold, we mark the node as *merged*; otherwise, we mark it as *inactive*. In addition, we mark as *active* all its neighbors with similarity scores below 1.

This process is guaranteed to terminate under the following two assumptions. First, we assume that the similarity-score functions for any node are monotonic in the similarity values of its incoming neighbors. Second, for a given node, we activate its neighbors only when its similarity increase is more than a small constant. While it might appear that requiring monotonic similarity functions precludes fixing initial incorrect reconciliation decisions in the presence of later negative evidence, this is not the case. We will show in Section 3.4 how we account for negative evidence.

Implementing the propagation procedure: With the refinement on graph edges, the algorithm does not need to activate all neighbors of a node in the second step, and so can largely reduce similarity re-computation. Specifically, given a node n whose similarity score increases, we do the following instead:

- we mark as *active* all of its outgoing real-valued inactive neighbors whose similarity scores are below 1.
- if we decide to reconcile the references in n, we mark as *active* all of its outgoing boolean-valued neighbors whose similarity scores are below 1.

In addition, a careful choice of the recomputation order can further reduce the number of recomputation and improve the algorithm efficiency. In particular, we employ the following heuristics.

- We compute similarity for a node only if the scores of its incoming value-based neighbors have all been computed, unless there exist mutual dependencies. For example, we compare two articles only after comparing their associated authors and conferences (or journals).
- When a node is merged, we consider its outgoing strong-boolean-valued neighbors first for recomputation.

Our algorithm proceeds by maintaining a queue of active nodes. Initially, the queue contains all reference-similarity nodes, and it satisfies the property that a node always precedes its outgoing real-valued neighbors if there does not exist mutual dependencies. At every iteration, we compute the similarity score for the top node in the queue. If we activate its outgoing real-valued or weak-boolean-valued neighbors, then we insert them at the end of the queue. If we activate its strong-boolean-valued neighbors, we insert them in front of the queue.

To illustrate, consider the dependency graph shown in Figure 2(a). Initially, the queue contains nodes $\{m_5, m_4, m_3, m_2, m_1\}$, and nodes n_1, n_2, and n_7 are marked *merged*. We then compute the similarity of m_5, m_4, m_3, m_2, and m_1 in succession. When we decide to merge papers a_1 and a_2, we insert m_2, m_3, m_4, and m_5 back to the front of the queue, so the queue becomes $\{m_5, m_4, m_3, m_2\}$ (the order among these nodes can be arbitrary). Note that n_2 is not added back both because it is not an outgoing neighbor of m_1, and because it already has similarity score 1. Next, we consider m_5 and decide to merge c_1 and c_2, so we insert its strong-boolean-valued neighbor, n_6, in the front of the queue and the queue becomes $\{n_6, m_4, m_3, m_2\}$. This process continues until the queue is empty.

3.3 Enriching the references

Another important aspect of our algorithm is that we *enrich* the references in the process of reconciliation. Specifically, after merging references r_1 and r_2, all the attributes of r_2 can also be considered as those of r_1. For example, if r_1 has email address "stonebraker@csail.mit.edu" and r_2 has email address "stonebraker@mit.edu", the real-world person object actually has both email addresses. Now, when we compute the similarity between r_1 and another reference r_3, we compare both email addresses with the email of r_3, and choose the one with a higher similarity. Note that for the purpose of reconciliation, we do not need to distinguish multiple email addresses and misspelled email addresses.

A naive way to enrich the references would be to run our propagation algorithm, then compute transitive closures and merge the references within the same cluster, and repeat the algorithm. However, with a little bit of care, we can implement enrichment with only local changes to the graph.

Specifically, after we decide to merge references r_1 and r_2, we search for all references r_3, such that there exist nodes $m = (r_1, r_3)$ and $n = (r_2, r_3)$. We proceed to remove n from the graph in the following steps: (1) connect all neighbors (incoming and outgoing neighbors) of n with m while preserving the direction of the edges, (2) remove node n and its associated edges from the dependency graph and from the queue, (3) if m gets new incoming neighbors and is not active in the queue, we insert m at the end of the queue; similarly,

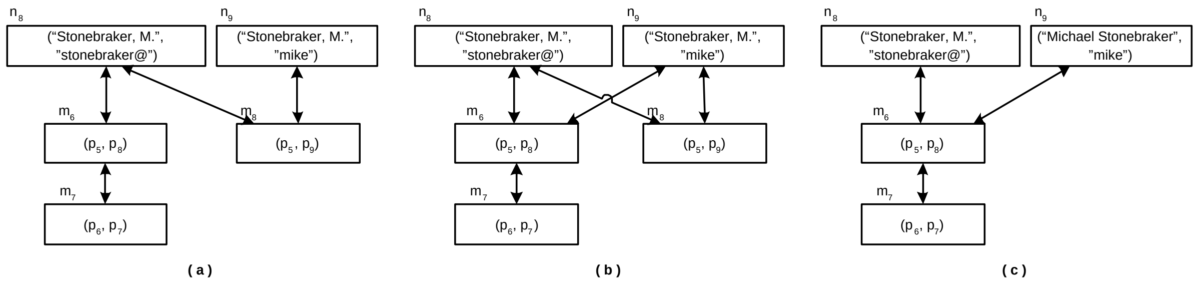
n8
n9
("Stonebraker, M.", "stonebraker@")
("Stonebraker, M.", "mike")
("Michael Stonebraker", "mike")
m6
m8
m7
(p5, p8)
(p5, p9)
(p6, p7)
(a)
(b)
(c)

Figure 3: Enrichment for references in Figure 1(b). (a) The original subgraph representing the similarity of references p_5 and p_8 (node m_6), and the similarity of p_5 and p_9 (node m_8). When we decide to reconcile p_8 and p_9, we need to compute similarity score for only one of m_6 and m_8. (b) In the first step, we connect all of m_8's neighbors with m_6, so n_9 is connected with m_6. Note that n_8 is already connected with m_6 so no change is needed. (c) In the second step, we remove node m_8 and all associated edges. We now have more evidences for reconciling p_5 and p_8.

n's neighbors that get new incoming neighbors and are not active are inserted at the end of the queue. For example, the recomputation of the similarity graph in Figure 2 is shown in Figure 3.

3.4 Enforcing constraints

Up to now, our algorithm has considered only *positive* evidences for similarity computation. In many cases, we may have negative evidences that can contribute to the reconciliation process. As a simple example, if in Figure 1(b), reference p_9 were ("Matt", "stonebraker@csail.mit.edu"), then we would not want to merge p_9 with p_2, which has "Michael Stonebraker" for the name attribute. However, as shown in Figure 3, the algorithm as described so far might reconcile p_8 and p_9 with p_5, and since p_2 is merged with p_5, they will all be merged together when we compute the transitive closure.

Indeed, the above problem is a fundamental one when we generate partitions by computing transitive closures: if we decide to reconcile r_1 with r_2, and r_2 with r_3, then r_1, r_2 and r_3 will be clustered even if we have evidence showing that r_1 is *not* similar to r_3.

We begin to address this problem by considering *constraints*. A constraint is a rule enforcing that two references are guaranteed to be distinct. For example, a constraint may specify that the authors of one paper are distinct from each other. Constraints are typically domain dependent. They can be manually specified by domain experts, or learned from training data or a clean auxiliary source [12]. To enforce constraints, we add one more status to the nodes in the dependency graph: *non-merge*. The two elements in a *non-merge* node are guaranteed to be different and should never be reconciled. Note that a *non-merge* node is different from a non-existing node. The absence of the node (r_1, r_2) indicates that r_1 and r_2 do not have similar attributes; but r_1 can still be reconciled with r_2 if both of them are reconciled with another reference r_3.

To incorporate constraints in our algorithm, we make the following modifications.

1. When constructing the dependency graph we also include nodes whose elements are ensured to be distinct. Such nodes are marked *non-merge* and will never enter the processing queue.
2. A node that has a non-merge incoming real-valued neighbor might be marked as *non-merge* according to the constraints.

```
procedure Reconciliation($\mathcal{R}$) return $\mathcal{P}$
// $\mathcal{R}$ is a reference set, $\mathcal{P}$ is a partitioning over $\mathcal{R}$
  Construct dependency graph $G = (N, E)$;
  Initiate queue q with all reference-similarity nodes that are
     not marked non-merge;
  while (q is not empty)
    Remove the first node $n = (r_1, r_2)$ from q and mark it inactive;
    $oldSim = sim(n)$; Re-compute $sim(n)$;
    if ($sim(n) > oldSim$)
      for each ($n$'s outgoing real-valued neighbor $m$)
        if ($m$ is not active AND $sim(m) < 1$)
          Insert $m$ to the end of q and mark $m$ as active;
    if ($sim(n)$ is above merge-threshold)
      Mark $n$ as merged;
      for each ($n$'s outgoing strong-boolean-valued neighbor $m$)
        if ($m$ is not active AND $sim(m) < 1$)
          Insert $m$ to the front of q and mark $m$ active;
      for each ($n$'s outgoing weak-boolean-valued neighbor $m$)
        if ($m$ is not active AND $sim(m) < 1$)
          Insert $m$ to the end of q and mark $m$ active;
      for each ($r_3$, s.t. $m = (r_1, r_3)$ and $l = (r_2, r_3)$ exist)
        Reconnect all of $l$'s neighbors with $m$;
        Remove $l$ from $G$ and q;
        When appropriate, insert $m$ and $l$'s neighbors to the end
           of $q$ and mark them as active;
  end while
  for each ($l = (r_1, r_2)$ marked non-merge)
    for each ($r_3$, s.t. $m = (r_1, r_3)$ and $n = (r_2, r_3)$ exist)
      if ($sim(m) > sim(n)$) Mark $n$ as non-merge;
      else Mark $m$ as non-merge;
  Compute transitive closure and return $\mathcal{P}$;
```

Figure 4: Algorithm for reference reconciliation

3. After the similarity computation reaches a fixed point, we examine every *non-merge* node $l = (r_1, r_2)$. Let r_3 be another reference such that there exist nodes $m = (r_1, r_3)$ and $n = (r_2, r_3)$, where $sim(m) > sim(n)$. If m is marked *merge*, we mark n as *non-merge*. This step propagates the negative evidence from l to n, so r_1 and r_2 will not be merged with a common reference.

Note that with the above modification, the algorithm still converges under the assumptions described in Section 3.2.

Figure 4 shows the reconciliation algorithm. As a further optimization, the dependency graph can be pruned at the very beginning using inexpensive reference comparisons, e.g., merging Person references that have the same email address. This preprocessing can significantly reduce the size of the dependency graph and thus improve the efficiency.

4. COMPUTING SIMILARITY SCORES

This section describes the similarity functions used in our algorithm. Given a node $m = (r_1, r_2)$, the similarity function for m takes the similarity scores of m's neighbors as input, and computes a score between 0 and 1.

Our dependency-graph based reconciliation algorithm has the flexibility of using different domain specific similarity functions for each class in the schema. As we explain in Section 6, this is an important advantage of our approach over those based on detailed probabilistic modeling [31, 30]. Note that training data, when available, can be used to learn or tune similarity functions for specific classes.

In what follows we propose a template of similarity functions that proved effective in our experiments. The similarity functions are orthogonal to the dependency graph framework and alternate simple or complex models can also be used. There are tunable parameters in our functions and these can either be learned from training data, or manually set from experience. Note that as we propagate information between reconciliation decisions, we may also propagate incorrect information from an over-estimated similarity value and mislead later similarity computations. Hence, we choose conservative similarity functions and merge-thresholds, in order to obtain a high precision. We obtain improvement in recall mainly by employing a broad collection of evidence.

The components of the similarity function: the similarity score S of a pair of elements is the sum of three components, and is always between 0 and 1: (1) S_{rv}, contributed by real-valued incoming neighbors, (2) S_{sb}, contributed by strong-boolean-valued neighbors, and (3) S_{wb}, contributed by weak-boolean-valued neighbors. When their sum exceeds 1, we set S to 1. Below we discuss each component in detail.

Real-valued neighbors ($\mathbf{S_{rv}}$): Given the similarity values of the real-valued incoming neighbors, we compute the value of S_{rv} between 0 and 1. For example, the similarity function of a person node combines the person-name similarities, person-email similarities, and name-and-email similarities.

Computing S_{rv} is similar in spirit to traditional record-linkage techniques, but with a few important differences. First, to account for the fact that most references do not have values for all attributes, we employ a *set* of similarity functions, rather than a single one. These functions account for cases in which some attributes are missing values (otherwise, we need to assign 0 to the similarity of those attributes, resulting in recall reduction). Second, our similarity functions account for some attributes serving as keys. For example, when two person instances have the same email address, they should be merged even if other attributes are different. Third, our similarity functions take into consideration any possible *non-merge* neighbors. We organize the set of similarity functions as a decision tree, where each branch node represents certain conditions, such as the existence of a similarity value or a *non-merge* neighbor, and each leaf node represents a function for S_{rv}.

Each function for S_{rv} combines the elementary similarity values using a linear combination. Formally, we define

$$S_{rv} = \sum_{i=1}^{n} \lambda_i x_i, \qquad (1)$$

where n is the number of different types of real-valued neighbors (e.g., email similarity, name similarity, etc.), x_i is the score for the elementary similarity of type T_i and λ_i is its corresponding weight.

The final wrinkle in computing S_{rv} is to account for attributes having multiple values (e.g., a person's email). For the similarity node $m = (r_1, r_2)$, we consider all possible attribute values of r_1 and r_2, and compute the similarity for all pairs of attributes that are comparable. If $N_i = \{n_1, \ldots, n_k\}$ is the set of multiple incoming neighbors of some type T_i, we use $MAX\{sim(n_j)|n_j \in N_i\}$ for the similarity value of the type T_i when computing S_{rv} in Equation 1.[1]

Strong-boolean neighbors ($\mathbf{S_{sb}}$): Let m be a similarity node. In principle, if two references in one of m's strong-boolean-valued neighbors are merged, then the two references in m should also be merged, unless they have very different atomic attribute values. For example, when we decide to merge two papers, we can merge their authors with similar names. However, to remain conservative and avoid possible errors caused by noisy information, we only increase the similarity score with a constant β for each merged strong-boolean-valued incoming neighbor. Formally, S_{sb} is:

$$S_{sb} = \begin{cases} \beta \cdot |N_{sb}| & if\ S_{rv} \geq t_{rv} \\ 0 & otherwise \end{cases}$$

where β is a constant, $|N_{sb}|$ is the number of merged strong-boolean-valued incoming neighbors, and t_{rv} is a threshold indicating that two references may possibly refer to the same entity. Given this function, when the initial similarity value of a pair of references, S_{rv}, is not very high (but above the threshold t_{rv}), the two references will be reconciled only if several strong-boolean-valued neighbors are merged. A more sophisticated function can require stricter conditions. For example, we increase the similarity score of two person names only if both names are *full* names.

Weak-boolean neighbors ($\mathbf{S_{wb}}$): Intuitively, the reconciliation of two references in a weak-boolean-valued incoming neighbor of m increases the similarity score of m. For example, we will have higher confidence in reconciling two person references if they not only have similar attribute values, but also contact common people (by email or by co-authorship). However, people's contact lists may vary a lot in length, and it is quite possible that two references refer to the same real-world person but have very different contact lists. Hence, requiring two contact lists to be similar is unnecessarily strict. In our algorithm we count the number of common contacts and increase the similarity score with a constant γ for each common contact. More generally, we define S_{wb} as

$$S_{wb} = \begin{cases} \gamma \cdot |N_{wb}| & if\ S_{rv} \geq t_{rv} \\ 0 & otherwise \end{cases}$$

where γ is a constant, and $|N_{wb}|$ is the number of merged weak-boolean-valued incoming neighbors. Note that the functions S_{wb} and S_{sb} have the same form, but we assign a much higher value to β than to γ. Furthermore, a sophisticated function for S_{wb} can assign a higher reward for the first several merged neighbors and a lower reward for the rest, or consider the relative size of the value set of an associated attribute.

5. EXPERIMENTAL RESULTS

[1]More complex cases can arise, e.g., an attribute value might be a set with each element in the set being multi-valued. Strategies for such cases are beyond the scope of this paper.

Person (name, *coAuthor)
Article (title, pages, *authoredBy, *publishedIn)
Venue (name, year, location)

Figure 5: Schema for references from Cora.

Dataset	#(References)	#(Entities)	#Ref/#Entity
PIM *A*	27367	2731	10.0
PIM *B*	40516	3033	13.4
PIM *C*	18018	2586	7.0
PIM *D*	17534	1639	10.7
Cora	6107	338	18.1

Table 1: Properties of our datasets: the number of extracted references, the number of real-world entities and the reference-to-entity ratio. The average reference-to-entity ratio, 11.8, underscores the importance of reference reconciliation.

We tested our algorithm on two domains: personal information management and publication portal. We now describe a set of experiments that validate the performance of our reconciliation algorithm. The experimental results show that our algorithm obtains high precision and recall in both contexts.

5.1 Datasets

Our first experiment involved four personal datasets. To ensure that we got a variety of references, we chose the owners of the datasets to be in different areas of computer science (database and theory), in different positions (faculty and graduate students), and most importantly, from different countries (including China, India and the USA)[2]. The datasets span several years of computer usage (from 3 to 7), and include references obtained from emails, Latex and Bibtex files, and PDF and Word documents. The references we extracted conform to the schema shown in Figure 1(a), except that conferences and journals were merged into a single Venue class. For each dataset, we manually created the gold standard, i.e., the perfect reconciliation result.

In order to demonstrate the applicability of our algorithm in a more conventional setting, our second experiment uses the subset of the Cora dataset provided by McCallum [11] and used previously in [8, 3, 30]. This dataset is a collection of 1295 different citations to 112 computer science research papers from the Cora Computer Science Research Paper Engine. We extracted references of types Person, Article and Venue from the citations, according to the schema shown in Figure 5. The papers in the dataset are already hand-labeled, while we had to label the persons and venues. The properties of our datasets are summarized in Table 1.

5.2 Experimental methodology

As in many other reference reconciliation works, we measured the overall performance with precision and recall, and reported *F-measure*, defined as

$$F - measure = \frac{2 \cdot prec \cdot recall}{prec + recall}$$

Observe that this penalizes results more for incorrect reconciliation for *popular* entities, i.e., entities with more distinct references. This is as required in the PIM context, where popular entities are browsed more often and errors in their reconciliation cause more inconvenience.

[2]Names and email addresses of persons from these countries have very different characteristics.

Table 2: Average precision, recall and F-measure for each class of references: DepGraph equals or outperforms IndepDec in all classes.

Class	IndepDec		DepGraph	
	Prec/Recall	F-msre	Prec/Recall	F-msre
Person	0.967/0.926	0.946	0.995/0.976	0.986
Article	0.997/0.977	0.987	0.999/0.976	0.987
Venue	0.935/0.790	0.856	0.987/0.937	0.961

Table 3: DepGraph outperforms IndepDec in reconciling Person references when only the email or paper subsets are considered and when the full datasets are considered.

Dataset	IndepDec		DepGraph	
	Prec/Recall	F-msre	Prec/Recall	F-msre
Full	0.967/0.926	0.946	0.995/0.976	0.986
PArticle	0.999/0.761	0.864	0.997/0.994	0.996
PEmail	0.999/0.905	0.950	0.995/0.974	0.984

We employed the same set of similarity functions and thresholds for *all* datasets. We manually set the thresholds and parameters and we used the same ones in *all* our experiments. Specifically, we set the merge-threshold to 0.85 for all reference similarities, and to 1 for all attribute similarities. We set $\beta = 0.1, \gamma = 0.05$ for all classes, except that we set $\beta = 0.2$ for Venue. We set $t_{rv} = 0.7$ for Person and Article references and $t_{rv} = 0.1$ for Venue references. We omit the detailed settings for S_{rv} since they vary from one class to another, but they are available in [14]. We note that as we chose the thresholds and parameters to be conservative, the results were insensitive to small perturbations in their values.

In our experiments, we refer to our algorithm as DepGraph. On the PIM data, we compared DepGraph with a candidate standard reference reconciliation approach, called IndepDec (it roughly corresponds to approaches such as [21, 27]). To contrast the two algorithms on the class Person, IndepDec compares person names and emails independently and combines the results for reference similarity without exploiting the dependencies between individual reconciliation decisions. DepGraph, in addition, compares the names with the email accounts, considers the articles authored by the persons, counts the common people appearing in the coauthor or email-contact lists, applies reconciliation propagation and reference enrichment, and enforces constraints. We use the same similarity functions and thresholds for IndepDec and DepGraph.

For the Cora dataset, we compared the results of DepGraph with the results reported in papers that proposed state-of-the-art reference reconciliation approaches.

5.3 Reference reconciliation for personal data

Table 2 shows the average, over the four datasets, precision and recall for articles, persons and venues (conferences and journals). DepGraph obtained higher precision and recall for both person and venue references. Specifically, it improved the recall for venue references by 18.6%, and for person references by 5.4%. Note that this 5.4% is in fact substantial: as we shall see in Table 4, it corresponds to a decrease in hundreds of partitions. Our algorithm does not

Table 4: Performance for different PIM datasets: DepGraph outperforms IndepDec for the Person class in each dataset. This shows that our algorithm is robust to variations in the nature of references.

PIM dataset	IndepDec			DepGraph		
#(Persons)/#(Refs)	Prec/Recall	F-measure	#(Par)	Prec/Recall	F-measure	#(Par)
A (1750/24076)	0.999/0.741	0.851	3159	0.999/0.999	0.999	1873
B (1989/36359)	0.974/0.998	0.986	2154	0.999/0.999	0.999	2068
C (1570/15160)	0.999/0.967	0.983	1660	0.982/0.987	0.985	1596
D (1518/17199)	0.894/0.998	0.943	1579	0.999/0.920	0.958	1546

improve the results for articles. This is because the article references are obtained from a set of bibtex files that are typically very well curated by the user. From Table 2 we also observe that while IndepDec may obtain either a low precision or a low recall in some cases, DepGraph typically obtains both high precision and high recall. Furthermore, as we will explain below, the improvement of DepGraph over IndepDec is most pronounced on the datasets in which the references have little information, or there is a greater variety in the references to the same real-world entity.

We now examine person references, which are associated with rich context information and therefore provide the most opportunities for performance improvement. We divided each dataset into two subsets: one containing person references extracted only from emails (PEmail) and the other containing person references extracted from articles and other non-email sources (PArticle). Table 3 shows the average precision and recall of the two approaches on the whole dataset and each subset of data. The DepGraph approach improved the recall by 30.7% on the article datasets, by 7.6% on the email datasets, and by 5.4% on the full datasets. It obtained significant recall increase on the article datasets by exploiting the associations between persons and articles, which compensate for the lack of information for each person reference in itself (each reference contains only the person name). The high precision and recall on the PEmail subset suggest that our algorithm has value also in information spaces that include a single class of references, but with rich associations between the references.

Table 4 shows the performance on each individual PIM dataset. DepGraph obtained higher F-measures and generated much fewer person reference partitions on all datasets. It improved the recall by 34.8% on dataset A, which has the highest variety in the presentations of individual person entities. Note that DepGraph reduced the recall on dataset D. The main reason is that the owner of the dataset changed her last name and also her email account (at the same email server) when she got married, so after enforcing the constraint, DepGraph divided her references into two partitions with similar sizes. Since the owner is typically the most popular entity in the dataset, dividing her references into two partitions leads to large loss in recall. However, we observe that the other references in her dataset were better reconciled: DepGraph obtained a much higher precision, and reduced the number of partitions by 33. In contrast, two other dataset owners also have name changing issues. DepGraph successfully merged their references because they continued to use the same email addresses after the name changes. One minor point is that the precision on dataset C is lower than others. The owner of the dataset is Chinese and her Chinese friends typically have short names with significant overlap, which makes reconciliation more difficult. For such a dataset, we did not find a set of similarity functions or heuristics that improve recall without sacrificing precision. Except the above, we did not find distinguishable performance difference between the datasets.

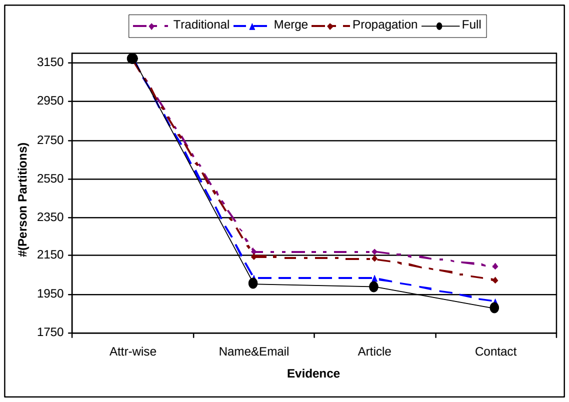

Figure 6: Each type of evidence and each algorithmic feature progressively contributes to more reconciliation on Person references in PIM A (see Table 5 for details). The top-left most point represents IndepDec while the bottom-right most point represents DepGraph.

Component Contributions

We now analyze the contribution of different components of the algorithm. We conducted experiments on dataset A, which has the highest variety and most room for improvement (DepGraph improved the recall from 0.741 to 0.999). This dataset contains 24076 Person references, and they refer to 1750 real-world persons.

Our analysis is along two orthogonal dimensions. Along one dimension, we analyzed the contribution of different types of evidence. We started with *Attr-wise* that compares person references by their names and their emails respectively. Then, we considered *Name&Email* that compares names against email addresses. Next, we considered *Article* that exploits the association between persons and their articles – two persons are likely to be the same, if the articles they authored are similar. Finally, we considered *Contact* that exploits common email-contacts and co-authors – persons with a similar set of email-contacts and co-authors are likely to be similar. Each of the above variations considers new evidence in addition to that of the earlier. We considered *Contact* last because it is likely to perform the best when some person references have already been reconciled, and thus the contact lists can be merged and enriched.

Along another dimension, we examined the independent contributions of reconciliation propagation and reference enrichment. We considered four modes:

Table 5: The number of Person reference partitions obtained by different variations of the algorithm on PIM dataset *A*. For each mode, the last column shows the improvement in recall (measured as the percentage reduction in the difference between the number of result partitions and the number of real-world entities) from *Attr-wise* to *Contact*, i.e., by considering all the additional available evidence. For each evidence variation, the last row shows the recall improvement by reconciliation propagation and reference enrichment. The bottom-right cell shows the overall recall improvement of DepGraph over IndepDec.

Mode	*Attr-wise*	*Name&Email*	*Article*	*Contact*	Reduction(%)
Traditional	3159	2169	2169	2096	75.4
Propagation	3159	2146	2135	2022	80.7
Merge	3169	2036	2036	1910	88.7
Full	3169	2002	1990	1873	91.3
Reduction(%)	-	39.9	42.7	64.6	91.3

- Full: apply both reconciliation propagation and reference enrichment.
- Propagation: apply only reconciliation propagation.
- Merge: apply only reference enrichment.
- Traditional: apply neither.

We observed very similar precision for the different variations and modes of the algorithm, and so we focus on recall. To demonstrate the difference in recall, we counted the number of person entities returned by each approach (i.e., the number of resulting partitions). Since the precision is about the same in all cases, this count is proportional to the recall and is shown in Figure 6. Table 5 shows the recall improvement in terms of the percentage of reduction in the difference between the number of person partitions returned by the reconciliation algorithm and the number of real-world person entities. Note that *Attr-wise* in the Traditional mode is equivalent to the IndepDec approach, and *Contact* in the Full mode is equivalent to the DepGraph approach. The IndepDec approach reconciled the 24076 references into 3159 instances, while the DepGraph approach reconciled them into 1873 instances, reducing the difference between the result number of references and the real number of references by 91.3% (from 1409 to 125).

Among the different evidence variations, *Name&Email* dramatically improved the recall. It helped greatly in reconciling the references extracted from Latex and Bibtex files with the references extracted from emails. It also helped reconcile different email accounts of a person. *Contact* also significantly increased the recall. In the full mode, it successfully reduced an extra 117 person partitions.

Among the different modes, the Full mode obtained the highest recall, and the Traditional mode got the lowest recall. Even when we considered all types of evidence, the Traditional mode generated 2096 instances. The gap between the resulting number of partitions and the real number of partitions is a factor of 2.81 times the corresponding gap to the Full mode. We observed that Merge performed much better than Propagation independent of the types of evidence being used. One important reason is that when the data has a high variety, i.e., one person has several different name presentations and email addresses, reference enrichment effectively accumulates these evidences for more informed reconciliation decisions. Another reason is that reference enrichment merges the contact lists that are originally scattered across different references of the same person, and thus significantly enhances *Contact*. However, the FULL mode does significantly better than either, thus demonstrates their combined utility.

Table 6: The precision, recall, the number of real-word entities that are involved in erroneous reconciliations (false-positives), and the graph size in terms of the number of nodes, show that considering constraints can significantly improve precision while maintaining recall without blowing up the graph.

Method	Prec/Recall	#(Entities with false-positives	#(Nodes)
DepGraph	0.999/0.9994	13	692030
Non-Constraint	0.947/0.9996	61	590438

Finally, we observed that reconciliation propagation and reference enrichment require abundant evidence to be effective. For *Attr-wise*, the four modes obtained very similar results. As we considered more evidence, the difference gradually grew. Also, observe that *Article* does not provide any benefit for Merge, but Propagation is able to reconcile the authors of the reconciled articles and derive some benefit. On the other hand, *Contact* provides significant benefit for Merge compared to for Propagation because of the consolidation of email-contact and co-author lists.

Effect of Constraints

We now examine the effect of constraint enforcement. We compared with the Non-Constraint approach, which does not consider any constraint or negative evidence. In contrast, DepGraph enforces the following three conditions:

1. Authors of a paper are distinct persons.
2. Two persons with the same first name but completely different last name, or with the same last name but completely different first name, are distinct persons unless they share the same email address.
3. A person has a unique account on an email server.

Table 6 shows the precision and recall of each approach, along with the number of real-world Person instances involved in false positives. The DepGraph approach obtained very high precision. Among the 1750 real-world instances, only 13 were incorrectly reconciled, of which 5 are mailing lists; hence, only 4 pairs of real person instances were incorrectly reconciled. The Non-constraint approach has much lower precision, where 61 instances were incorrectly matched. We also observed that while considering constraints added more nodes in the dependency graph, a care-

Table 7: Precision, recall and F-measure in the Cora dataset: DepGraph obtains a higher F-measure for all types of references.

Class	IndepDec		DepGraph	
	Prec/Recall	F-msre	Prec/Recall	F-msre
Person	0.994/0.985	0.989	1/0.987	0.993
Article	0.985/0.913	0.948	0.985/0.924	0.954
Venue	0.982/0.362	0.529	0.837/0.714	0.771

ful choice of constraints did not necessarily blow up the graph.

5.4 The Cora Dataset

Table 7 shows the precision, recall and F-measure for DepGraph and IndepDec on the Cora dataset. We observed a large improvement of F-measure on Venue references and an improvement on Article and Person references. We note that the Cora dataset is very noisy; for example, citations of the same paper may mention different venues. On such a dataset, the effect of the propagation from article nodes to venue nodes was two-fold. On the one hand, it helped reconcile a large number of venues and improve the recall on venue references, and this in turn improved the recall on article references. On the other hand, it incorrectly reconciled the many different venues mentioned in citations to the same article, and thus reduced the precision.

Finally, we compared our results with other reported experimental results on the same benchmark dataset. [30] reported a 0.842/0.909 precision/recall on their collective record linkage approach; [3] reported a 0.867 F-measure on their adaptive approach; and [8] reported a 0.99/0.925 precision/recall on their approach. Because our algorithms handle key attributes in a different way (two references are reconciled if they agree on key values), both IndepDec and DepGraph obtained high precision and recall. Nevertheless, the strength of dependency graph further improved the result and made it comparable to those of the above adaptive approaches, even without using any training data.

6. RELATED WORK

The problem of reference reconciliation, originally defined by Newcombe, et al. [29], was first formalized by Fellegi and Sunter [17]. A large number of approaches that have been since proposed [3, 21, 27, 36, 35] use some variant of the original Felligi-Sunter model. Reconciliation typically proceeds in three steps: first, a vector of similarity scores is computed for individual reference pairs by comparing their attributes; second, based on this vector, each reference pair is compared as either a match or non-match; and finally, a transitive closure is computed over matching pairs to determine the final partition of references. Attribute comparison is done by using either generic string similarity measures (see [10, 4] for a comprehensive comparison and [6, 35, 3] for recent adaptive approaches) or some domain-specific measures (e.g., geographic locality in [1]). The classification of candidate reference pairs into match and non-match pairs is done through a variety of methods: (a) rule-based methods [21, 24, 18, 22] that allow human experts to specify matching rules declaratively; (b) unsupervised learning methods [36] that employ the Expectation-Maximization (EM) algorithm to estimate the importance of different components of the similarity vector; (c) supervised learning methods [32, 8, 35, 34, 4] that use labeled examples to train a probabilistic model, such as a decision tree, Bayesian network, or SVM, and later apply the model to identify matching pairs; and finally, (d) methods that prune erroneous matching pairs by validating their merged properties using knowledge in secondary sources [12, 15, 28].

Our approach departs from the basic three-step model in that our dependency graph effectively models relations between reconciliation decisions. There is a continuous feedback loop between computing similarities and matching decisions. Importantly, our framework retains the flexibility of employing the different established techniques described above for computing attribute and reference similarities.

Most prior work has treated reference reconciliation as a single class problem, and it is typically assumed that attribute values, albeit noisy ones, are known for all the references. This model breaks down in complex information spaces such as PIM, where there are multiple classes and individual references not only can have missing attribute values, but also can have multiple valid attribute values. We are able to offset this complexity by exploiting associations between references to design new similarity measures, and to learn from matching decisions across multiple classes.

The idea of capturing the dependencies between reconciliation decisions has recently been explored in the data mining and machine learning community. In [31], a complex generative model is proposed that captures dependencies between various classes and attributes and also possible errors during reference extraction. In [30], a dependency model is proposed that propagates reconciliation decisions through shared attribute values. Both the above approaches entail learning a global detailed probabilistic model from training data, and having the entire reconciliation process guided by that probabilistic model. In complex information spaces that contain multiple classes and complex associations between the classes, learning such a model is impractical. In contrast, our approach provides a mechanism for exploiting influences between reconciliation decisions, and it allows applying different domain-specific models (either heuristic or learned) for particular classes of references.

In [2, 23], associations are used to compute similarities and relate reconciliation decisions. Their proposed heuristics are just a subset of the many heuristics we use. Further, we are considering a much more complex domain.

In prior work [15], we proposed an approach in which references are reconciled by a sequence of comparison and matching steps with different similarity measures being used in different steps. Merging between steps was used to increase information about individual references. However, the classes are treated in isolation and associations between classes are not exploited. [13] described an algorithm for person reconciliation in PIM, and sketched a high-level model for relating reconciliation decisions between multiple classes, but no details or experimental validation was provided.

The use of negative information was proposed in [12] to validate individual reconciliation decisions. Our framework exploits the dependency graph to propagate such information for additional benefit.

Finally, several work [21, 27, 6, 22] has addressed the computational cost of reference reconciliation. We follow the spirit of the canopy mechanism [27] to reduce the size of our dependency graph. We insert into the graph only atomic attribute value pairs and reference pairs that have some potential to be similar.

7. CONCLUSIONS AND FUTURE WORK

Many applications that require reference reconciliation, such as personal information management systems, research paper citation portals and product catalog integration, are based on information spaces that involve multiple classes and rich relationships between instances. Thus far, reference reconciliation has been studied only in the context of reconciling references of a single class. This paper fills in this gap by proposing a generic framework that effectively exploits the rich information present in the associations between references, and reconciles references of multiple classes at one time. Specifically, to make more informed reconciliation decisions, our framework influences later similarity computation with early reconciliation decisions, and enriches references by instant merging. We apply our algorithm to PIM and Cora datasets, and our experimental results show that it obtains high precision and recall in both applications. While we emphasize applications with multiple classes of objects, our experiments on email-address reconciliation show that our algorithm also benefits record-linkage tasks that match only a single class of objects.

We plan to explore several directions in future work. First, we will consider an efficient incremental reconciliation approach, applied when new references are inserted to an already-reconciled dataset. Second, we will study learning techniques based on our framework. Specifically, we will consider how to use user feedback to adjust similarity functions and improve future reconciliation results.

Acknowledgments

We thank Anhai Doan, Pedro Domingos, Parag, and the anonymous reviewers for their insightful comments. We also thank Ema Nemes, Parag, and Jun Zhang for their invaluable help in the creation of our evaluation benchmark. The research was funded by NSF CAREER Grant 9985114, NSF ITR Grant 0205635, and DARPA Contract 03-000225.

8. REFERENCES

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

[2] I. Bhattacharya and L. Getoor. Iterative record linkage for cleaning and integration. In *DMKD*, 2004.

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

[4] M. Bilenko, R. Mooney, W. Cohen, P. Ravikumar, and S. Fienberg. Adaptive name matching in information integration. *IEEE Intelligent Systems Special Issue on Information Integration on the Web*, September 2003.

[5] V. Bush. As we may think. *The Atlantic Monthly*, 1945.

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

[7] Computer and information science papers citeseer publications researchindex. http://citeseer.ist.psu.edu/.

[8] W. Cohen and J. Richman. Learning to match and cluster large high-dimensional data sets for data integration, 2002.

[9] W. W. Cohen, H. Kautz, and D. McAllester. Hardening soft information sources. In *SIGKDD*, 2000.

[10] W. W. Cohen, P. Ravikumar, and S. E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *IIWEB*, pages 73–78, 2003.

[11] http://www.cs.umass.edu/~mccallum/data/cora-refs.tar.gz.

[12] A. Doan, Y. Lu, Y. Lee, and J. Han. Object matching for information integration: a profiler-based approach. In *IIWeb*, 2003.

[13] X. Dong and A. Halevy. A Platform for Personal Information Management and Integration. In *Proc. of CIDR*, 2005.

[14] X. Dong, A. Halevy, and J. Madhavan. Reference Reconciliation in Complex Information Spaces. Technical Report 2005-03-04, Univ. of Washington, 2005.

[15] X. Dong, A. Halevy, E. Nemes, S. Sigurdsson, and P. Domingos. Semex: Toward on-the-fly personal information integration. In *IIWeb*, 2004.

[16] S. Dumais, E. Cutrell, J. Cadiz, G. Jancke, R. Sarin, and D. C. Robbins. Stuff i've seen: A system for personal information retrieval and re-use. In *SIGIR*, 2003.

[17] I. P. Fellegi and A. B. Sunter. A theory for record linkage. In *Journal of the American Statistical Association*, 1969.

[18] H. Galhardas, D. Florescu, D. Shasha, E. Simon, and C.-A. Saita. Declarative data cleaning: language, model, and algorithms. In *VLDB*, pages 371–380, 2001.

[19] Google. http://desktop.google.com/, 2004.

[20] L. Gu, R. Baxter, D. Vickers, and C. Rainsford. Record linkage: current practice and future directions. http://www.act.cmis.csiro.au/rohanb/PAPERS/record_linkage.pdf.

[21] M. A. Hernandez and S. J. Stolfo. The merge/purge problem for large databases. In *SIGMOD*, 1995.

[22] L. Jin, C. Li, and S. Mehrotra. Efficient Record Linkage in Large Data Sets. In *DASFAA*, 2003.

[23] D. V. Kalashnikov, S. Mehrotra, and Z. Chen. Exploiting relationships for domain-independent data cleaning. In *SIAM Data Mining (SDM)*, 2005.

[24] M. L. Lee, T. W. Ling, and W. L. Low. Intelliclean: a knowledge-based intelligent data cleaner. In *SIGKDD*, pages 290–294, 2000.

[25] A. McCallum, K. Nigam, J. Rennie, and K. Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, 2000.

[26] A. McCallum and B. Wellner. Toward conditional models of identity uncertainty with application to proper noun coreference. In *IIWEB*, 2003.

[27] A. K. McCallum, K. Nigam, and L. H. Ungar. Efficient Clustering of High-Dimensional Data Sets with Application to Reference Matching. In *SIGKDD*, 2000.

[28] M. Michalowski, S. Thakkar, and C. A. Knoblock. Exploiting secondary sources for unsupervised record linkage. In *IIWeb*, 2004.

[29] H. Newcombe, J. Kennedy, S. Axford, and A. James. Automatic linkage of vital records. In *Science 130 (1959), no. 3381*, pages 954–959, 1959.

[30] Parag and P. Domingos. Multi-relational record linkage. In *MRDM*, 2004.

[31] H. Pasula, B. Marthi, B. Milch, S. Russell, and I. Shpitser. Identity uncertainty and citation matching. In *NIPS*, 2002.

[32] J. C. Pinheiro and D. X. Sun. Methods for linking and mining massive heterogeneous databases. In *SIGKDD*, 1998.

[33] D. Quan, D. Huynh, and D. R. Karger. Haystack: A platform for authoring end user semantic web applications. In *ISWC*, 2003.

[34] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *SIGKDD*, 2002.

[35] S. Tejada, C. Knoblock, and S. Minton. Learning domain-independent string transformation weights for high accuracy object identification. In *SIGKDD*, 2002.

[36] W. E. Winkler. Using the em algorithm for weight computation in the fellegi-sunter model of record linkage. In *Section on Survey Research Methods*, 1988.

[37] W. E. Winkler. The state of record linkage and current research problems. Technical report, U.S. Bureau of the Census, Wachington, DC, 1999.