Collective Entity Resolution in Relational Data

INDRAJIT BHATTACHARYA and LISE GETOOR
University of Maryland, College Park

Many databases contain uncertain and imprecise references to real-world entities. The absence of identifiers for the underlying entities often results in a database which contains multiple references to the same entity. This can lead not only to data redundancy, but also inaccuracies in query processing and knowledge extraction. These problems can be alleviated through the use of *entity resolution*. Entity resolution involves discovering the underlying entities and mapping each database reference to these entities. Traditionally, entities are resolved using pairwise similarity over the attributes of references. However, there is often additional relational information in the data. Specifically, references to different entities may cooccur. In these cases, collective entity resolution, in which entities for cooccurring references are determined jointly rather than independently, can improve entity resolution accuracy. We propose a novel relational clustering algorithm that uses both attribute and relational information for determining the underlying domain entities, and we give an efficient implementation. We investigate the impact that different relational similarity measures have on entity resolution quality. We evaluate our collective entity resolution algorithm on multiple real-world databases. We show that it improves entity resolution performance over both attribute-based baselines and over algorithms that consider relational information but do not resolve entities collectively. In addition, we perform detailed experiments on synthetically generated data to identify data characteristics that favor collective relational resolution over purely attribute-based algorithms.

Categories and Subject Descriptors: H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Clustering*; H.2.8 [**Database Management**]: Database Applications—*Data mining*

General Terms: Algorithms

Additional Key Words and Phrases: Entity resolution, graph clustering, record linkage, data cleaning

ACM Reference Format:

Bhattacharya, I. and Getoor, L. 2007. Collective entity resolution in relational data. ACM Trans. Knowl. Discov. Data, 1, 1, Article 5 (March 2007), 36 pages. DOI = 10.1145/1217299.1217304 http://doi.acm.org/10.1145/1217299.1217304

This work was supported by the National Science Foundation, NSF no. 0423845 and NSF no. 0438866 with additional support from the ITIC KDD Program.
Author's address: University of Maryland, College Park, MD; email: getoor@cs.umd.edu.

1. INTRODUCTION

In many applications, there are a variety of ways of referring to the same underlying real-world entity. For example in a census database, J. Doe, Jonathan Doe, and Jon Doe may all refer to the same person. Additionally, in many domains, references to different entities often cooccur in the data. For example, the same database may also have records showing that Jonathan Doe is married to Jeanette Doe and has dependents James Doe and Jason Doe, Jon Doe is married to Jean Doe, and J. Doe has dependents Jim Doe, Jason Doe and Jackie Doe. Such relationships between entity references are best represented as a graph, which we refer to as the *reference graph*, where the nodes are the entity references and edges (or often hyperedges) in the graph indicate references which cooccur. The problem is, for any real-world entity, there may be multiple references to it, and, accordingly, there is more than one node in the reference graph corresponding to that entity.

Thus an important first step in any graph mining algorithm is transforming such a reference graph into an *entity graph*, where nodes are the entities themselves and edges are among entities. Given a collection of references to entities, we would like to a) determine the collection of true underlying entities and b) correctly map the entity references in the collection to these entities. Figure 1(a) shows the reference graph from our census example and Figure 1(b) shows the entity graph after the references in the reference graph have been resolved. Even in this simple example, the entity graph is much smaller than the reference graph and consists of a single connected component. This provides a much more accurate picture of the underlying domain structure than the collection of disconnected subgraphs in the reference graph.

Entity resolution is a common problem that comes in different guises (and is given different names) in many computer science domains. Examples include computer vision, where we need to figure out when regions in two different images refer to the same underlying object (the correspondence problem); natural language processing when we would like to determine which noun phrases refer to the same underlying entity (coreference resolution); and databases, where, when merging two databases or cleaning a database, we would like to determine when two tuple records are referring to the same real-world object (deduplication and data integration). Deduplication [Hernández and Stolfo 1995; Monge and Elkan 1996] is important for both accurate analysis, for example, determining the number of customers, and for cost-effectiveness, for example, removing duplicates from mailing lists. In information integration, determining approximate joins [Cohen 2000] is important for consolidating information from multiple sources; most often there will not be a unique key that can be used to join tables in distributed databases, and we must infer when two records from different databases, possibly with different structures, refer to the same entity. In many of these examples, cooccurrence information in the input can be naturally represented as a graph.

Traditional approaches to entity resolution and deduplication use a variety of attribute similarity measures, often based on approximate string-matching criteria. These work well for correcting typographical errors and other types of

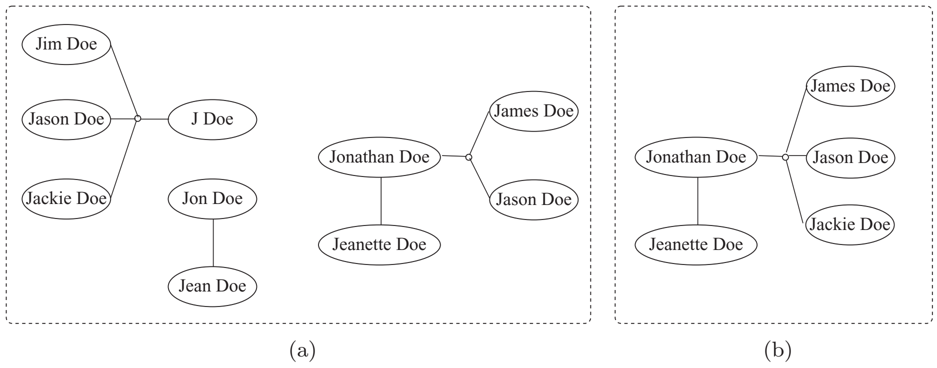


Fig. 1. Example of (a) a reference graph for the simple example given in the text and (b) the resolved entity graph.

noisy reference attributes. More sophisticated approaches make use of domain-specific attribute similarity measures and often learn such mapping functions from resolved data. However, it is still difficult to decide when identical references are in fact distinct. For example, two people with name J. Doe and living at the same address and of the same age may be brothers and not the same person.

More recent approaches take structural (i.e., relational) similarity into account [Ananthakrishna et al. 2002; Bhattacharya and Getoor 2004; Kalashnikov et al. 2005; Dong et al. 2005]. One approach simply looks at the attributes of related references and incorporates them into the attribute similarity score. For example, if we are comparing two census records for Jon Doe and Jonathan Doe, we would be more likely to match them if they are married to Jean Doe and Jeannette Doe. The problem becomes even more interesting when we assume that the entity for a reference depends not on the attribute similarities of related references but instead on the entities to which they correspond. Then the references cannot be assigned to entities independently any more—the entities for related references depend on one another. In our example, we would not consider Jon Doe and Jonathan Doe to be the same person simply because their wives' names are similar since different people may have wives with similar names. But if we can determine that they are married to the same person, this would provide significant evidence that these references refer to the same entity. Because the resolutions are no longer independent, the problem becomes one of *collective* entity resolution.

The collective entity resolution problem can be solved in an *iterative* fashion. Determining that two references refer to the same entity may in turn allow us to make additional inferences about their related references. In our census example, if we are able to determine that the two Jason Does refer to the same person, that would provide further evidence that their fathers, Jonathan Doe and J. Doe, are also the same, and possibly that their siblings Jim Doe and James Doe are the same person as well. Using the relational evidence for collective entity resolution has the potential benefit that we may be able to produce more accurate results than using only attribute similarity measures.

This potential benefit comes at the cost of additional algorithmic complexity resulting from propagating the dependence between different resolution decisions. In this work, we study the trade-off between the increased accuracy offered by collective resolution and the computational cost required to achieve this improvement.

Here we propose a collective entity resolution approach based on a novel unsupervised relational clustering algorithm. This article builds on our initial work on entity resolution in relational data described in a workshop paper [Bhattacharya and Getoor 2004] and included in a survey book chapter [Bhattacharya and Getoor 2006a]. The contributions of this article are: 1) a comprehensive description of the collective relational entity resolution algorithm, 2) a thorough exploration of different types of neighborhood similarity measures, 3) a comparison with a naive relational clustering approach, 4) evaluations on significantly larger real datasets that have multivalued attributes, and 5) a suite of synthetic data experiments which investigate the performance of collective entity resolution against varying structural characteristics of the data.

The rest of this article is organized as follows. In Section 2, we present a more realistic motivating example for entity resolution using the relations between references. In Section 3, we formalize the relational entity resolution problem. We explore and compare different approaches for entity resolution and formulate collective relational entity resolution as a clustering problem in Section 4. We propose novel relational similarity measures for collective relational clustering in Section 5. We discuss the clustering algorithm in further detail in Section 6. In Section 7, we describe experimental results using the different similarity measures on multiple real-world datasets. We also present detailed experiments on synthetically generated data to identify data characteristics that indicate when collective resolution should be favored over the more naive approaches. We review related work on entity resolution in Section 8, and finally conclude in Section 9.

2. MOTIVATING EXAMPLE FOR ENTITY RESOLUTION USING RELATIONSHIPS

We consider as our motivating example the problem of resolving the authors in a database of academic publications similar to DBLP, CiteSeer, or PubMed.

Consider the following set of four papers, which we will use as a running example:

(1) W. Wang, C. Chen, A. Ansari, A mouse immunity model
(2) W. Wang, A. Ansari, A better mouse immunity model
(3) L. Li, C. Chen, W. Wang, Measuring protein-bound fluxetine
(4) W. W. Wang, A. Ansari, Autoimmunity in biliary cirrhosis

Now imagine that we would like to find out, given these four papers, which of these author names refer to the same author entities. This involves determining whether paper 1 and paper 2 are written by the same author named Wang or whether they are different authors. We need to make similar decisions about the

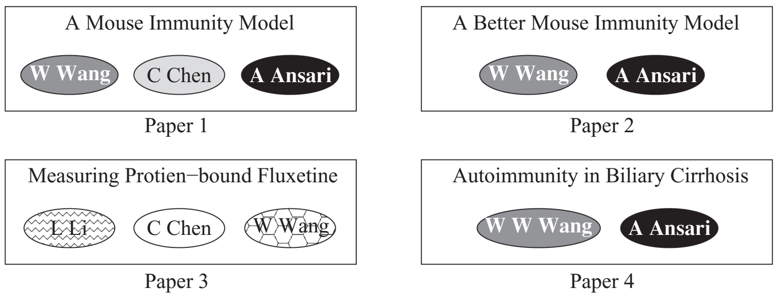


Fig. 2. The references in different papers in the bibliographic example. References to the same entity are identically shaded.

Wang from paper 3 and the Wang from paper 4 and all pairwise combinations. We need to answer similar questions about the other author names Ansari and Chen as well.

In this example, it turns out there are six underlying author entities, which we will call *Wang*1 and *Wang*2, *Chen*1 and *Chen*2, *Ansari* and *Li*. The three references with the name A. Ansari correspond to author *Ansari* and the reference with name L. Li to author *Li*. However, the two references with name C. Chen map to two different authors *Chen*1 and *Chen*2. Similarly, the four references with name W. Wang or W. W. Wang map to two different authors. The Wang references from the first, second, and fourth papers correspond to author *Wang*1, while that from the third paper maps to a different author, *Wang*2. This is shown pictorially in Figure 2, where references which correspond to the same authors are shaded identically.

There are two different subproblems that are of interest in solving the entity resolution problem. One is figuring out for any author entity the set of different name references which may be used to refer to the author. We refer to this as the *identification problem*. For example, for a real-world entity with the name Wei Wei Wang, her name may come up as Wei Wang, Wei W. Wang, W. W. Wang, Wang, W. W. and so on. There may also be errors in the data entry process so that the name may be incorrectly recorded as W. Wong or We Wang, etc.

In addition to the reconciliation of different-looking names which refer to the same underlying entity, a second aspect of the entity resolution problem is distinguishing references that have very similar, and sometimes exactly the same, name and yet refer to different underlying entities. We refer to this as the *disambiguation problem*. An example of this is determining that the W. Wang of paper 1 is distinct from the W. Wang of paper 3. The extent of the disambiguation problem depends on the domain. The problem can be exacerbated by the use of abbreviations—many databases (e.g., PubMed) store only abbreviated first names.

Our aim is to make use of the relationships that hold among the observed references to resolve them better and to solve both the identification and disambiguation problem at the same time. As in the case of the census example, we can represent the relationships as a graph where the vertices represent the author references and the hyperedges represent the coauthorship relations that

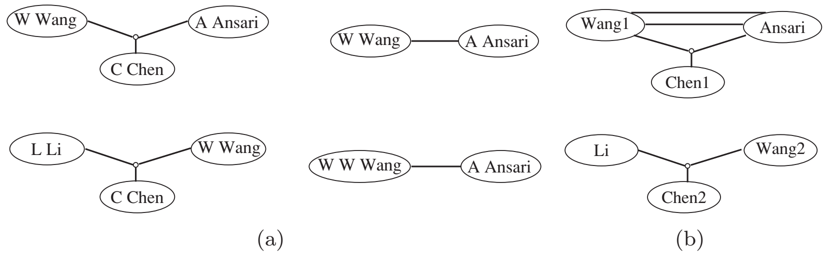


Fig. 3. (a) The reference graph and (b) the entity graph for the author resolution example.

hold between them in the dataset. Figure 3(a) shows the reference graph for our bibliographic example. Given this graph representation for our data, our goal is to take the hyperedges into account to better partition the references into entities. Now, in addition to the similarity of the attributes of the references, we consider their relationships as well. In terms of the graph representation, two references that have similar attributes are more likely to be the same entity if their hyperedges connect to the same entities as well. To see how this can help, observe in Figure 2(a) that the Wang references in papers 1, 2, and 4 collaborate with Ansari's, who correspond to the same author. This makes it more likely that they are the same entity. In contrast, the Wang from paper 3 collaborates with different authors which suggests that it does not refer to the same person as the other cases.

But it seems that we are stuck with a chicken-and-egg problem. The identity of a reference depends on those of its collaborators, and the identity of the collaborators depends on the identity of the reference itself. So where do we begin? Intuitively, we start with the resolutions that we are most confident about. For instance, two references with the name A. Ansari are more likely to be the same because Ansari is an uncommon name in contrast to references with common names such as Chen, Li or Wang. This then provides additional evidence for merging other references. In our example, after consolidating the Ansari's, the Wang references from paper 1, 2, and 4 have a common coauthor, which provides evidence for consolidating them. The entity resolution algorithm incrementally constructs the entity graph by considering as evidence the entity relationships that it has already discovered in earlier iterations. Figure 3(b) shows the resulting entity graph for our example after all the references have been correctly resolved.

3. ENTITY RESOLUTION USING RELATIONSHIPS: PROBLEM FORMULATION

In this section, we describe the notation we use for describing the relational entity resolution problem. In the entity resolution problem, we are given a set of references $\mathcal{R} = \{r_i\}$, where each reference r has attributes $r.A_1, r.A_2, \ldots, r.A_k$. The references correspond to some set of unknown entities $\mathcal{E} = \{e_i\}$. We introduce the notation $r.E$ to refer to the entity to which reference r corresponds. The problem is to recover the hidden set of entities $\mathcal{E} = \{e_i\}$ and the entity labels $r.E$ for individual references, given the observed attributes of the references.

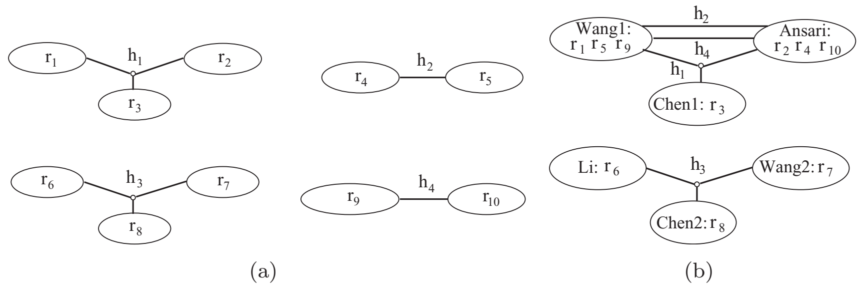


Fig. 4. (a) A more abstract representation of the reference graph for the author resolution example; the r's are references and the h's are hyperedges. (b) An abstract representation for the entity graph for the author resolution example; the nodes are the entities, the set of references they correspond to are listed, and the h's are hyperedges.

In addition to the attributes, we assume that the references are not observed independently but that they cooccur. We describe the cooccurrence with a set of hyperedges $\mathcal{H} = \{h_i\}$. Each hyperedge h may have attributes as well which we denote $h.A_1, h.A_2, \ldots, h.A_l$, and we use $h.R$ to denote the set of references that it connects. A reference r can belong to zero or more hyperedges, and we use $r.H$ to denote the set of hyperedges in which r participates. In this article, we only discuss entity resolution when each reference is associated with zero or one hyperedge, but in other domains, it is possible for multiple hyperedges to share references. For example, if we have paper, author, and venue references, then a paper reference may be connected to multiple author references and also to a venue reference.

Let us now illustrate how our running example is represented in this notation. Figure 4(a) shows the references and hyperedges. Each observed author name corresponds to a reference, so there are ten references r_1 through r_{10}. In this case, the names are the only attributes of the references, so, for example, $r_1.A$ is W. Wang, $r_2.A$ is C. Chen and $r_3.A$ is A. Ansari. The set of true entities $\mathcal{E}$ is $\{Ansari, Wang1, Wang2, Chen1, Chen2, Li\}$ as shown in Figure 4(b). References r_1, r_5, and r_9 correspond to $Wang1$, so that $r_1.E = r_5.E = r_9.E = Wang1$. Similarly, $r_2.E = r_4.E = r_{10}.E = Ansari$, and $r_3.E = Chen1$, and so on. There are also the hyperedges $\mathcal{H} = \{h_1, h_2, h_3, h_4\}$, one for each paper. The attributes of the hyperedges in this domain are the paper titles, for example, $h_1.A_1 =$ A Mouse Immunity Model. The references r_1 through r_3 are associated with hyperedge h_1 since they are the observed author references in the first paper. This is represented as $h_1.R = \{r_1, r_2, r_3\}$. Also, this is the only hyperedge in which each of these references participate. So $r_1.H = r_2.H = r_3.H = \{h_1\}$. We similarly represent the hyperedge associations of the other references.

4. ENTITY RESOLUTION APPROACHES

In this section, we compare and contrast existing entity resolution approaches. We distinguish between attribute-based, naive relational and collective relational entity resolution. While the attribute-based approach considers only the attributes of the references to be matched, the naive relational approach considers attribute similarities for related references as well. In contrast, the collective relational approach resolves related references jointly. We describe these approaches in detail in the following subsections.

4.1 Attribute-Based Entity Resolution

This is the traditional approach [Fellegi and Sunter 1969; Cohen et al. 2003] where similarity $sim_A(r_i, r_j)$ is computed for each pair of references r_i, r_j based on their attributes, and only those pairs that have similarity above some threshold are considered coreferent. We use the abbreviation **A** to refer to the attribute-based approach. Additionally, transitive closure may be taken over the pairwise decisions. We denote this approach as **A***.

Several sophisticated similarity measures have been developed for names, such as the Jaro, Levenstein, Jaro-Winkler, among others, and popular TF-IDF schemes may be used for other textual attributes like keywords. The measure that works best for each attribute can be used. Finally, a weighted combination of the similarities over the different attributes for each reference can be used to compute the attribute similarity between two references. In our example, the approach **A** may allow us to decide that the W. Wang references (r_1, r_5) are coreferent. We may also decide using **A** that W. Wang and W. W. Wang (r_1, r_9) are coreferent, but not as confidently. However, as we have already discussed, attributes are often insufficient for entity resolution, particularly for the disambiguation aspect of the problem. In our example, **A** is almost certain to mark the two W. Wang references (r_1, r_7) as coreferent, which is incorrect.

4.2 Naive Relational Entity Resolution

The simplest way to use relationships to resolve entities is to treat related references as additional attributes for matching. For instance, to determine if two author references in two different papers are coreferent, we can compare the names of their coauthors. In our running example, the naive relational decision about the references W. Wang and W. W. Wang, would consider that both have coauthors with the name A. Ansari. We refer to this approach as **NR**. As before, transitive closure can be taken over the pairwise decisions for **NR**. We refer to the transitive closure as **NR***.

A similar idea has been used in the context of matching in dimensional hierarchies [Ananthakrishna et al. 2002]. We generalize the idea for unordered relationships and define hyperedge similarity $sim_H(h_i, h_j)$ between two hyperedges h_i and h_j as the best pairwise attribute match between their references. Since the references in any hyperedge are not ordered, each reference $r \in h_i$ can be matched to any reference $r' \in h_j$. So for each reference $r \in h_i$ we find the best match to h_j:

$$sim_H(r, h_j) = max_{r' \in h_j} sim_H(r, r').$$

For symmetry, we also compute the best match to hyperedge h_i for each reference in h_j and then take the average over all of the references in the two hyperedges to get $sim_H(h_i, h_j)$. We then use this similarity measure between two hyperedges to find the hyperedge similarity $sim_H(r_i, r_j)$ between two references r_i and r_j by matching their hyperedges. When each reference belongs to just one hyperedge, $sim_H(r_i, r_j)$ can be computed simply as $sim_H(r_i.H, r_j.H)$. Otherwise, we need to make pairwise comparisons between their hyperedges. Finally, we take a simple linear combination of the attribute match $sim_A(r_i, r_j)$ and the hyperedge match $sim_H(r_i, r_j)$ to get naive relational similarity for two references r_i and r_j:

$$sim_{NR}(r_i, r_j) = (1 - \alpha) \times sim_A(r_i, r_j) + \alpha \times sim_H(r_i, r_j), \qquad 0 \leq \alpha \leq 1. \qquad (1)$$

While the naive relational approach improves significantly on the attribute-based approach, it can mislead in domains where most names are frequent and hyperedges are dense. In our example, the two W. Wang references, r_1 and r_7, are not coreferent, though they have coauthors with matching names C. Chen. Since we only match the strings, naive relational similarity returns a high match value. This may incorrectly lead to the decision that r_1 and r_7 are coreferent.

4.3 Collective Relational Entity Resolution

The problem with the naive relational approach is that it does not reason about the identities of the related references. For the two Wang references in the earlier example, the two C. Chen coauthors match regardless of whether they refer to *Chen*1 or *Chen*2. The correct evidence to use here is that the Chens are not coreferent. In such a setting, in order to resolve the W. Wang references, it is necessary to resolve the C Chen references as well and not just consider their name similarity. This is the goal of collective relational entity resolution (**CR**), where resolutions are not made independently, but instead one resolution decision affects other resolutions via hyperedges. We now motivate entity resolution as a clustering problem and propose a *relational clustering algorithm* for collective relational entity resolution.

Given any similarity measure between pairs of references, entity resolution can be posed as a clustering problem where the goal is to cluster the references so that only those that correspond to the same entity are assigned to the same cluster. We use a greedy agglomerative clustering algorithm where, at any stage, the current set $\mathcal{C} = \{c_i\}$ of *entity clusters* reflects the current belief about the mapping of the references to entities. We use $r.C$ to denote the current cluster label for a reference; references that have the same cluster label correspond to the same entity.

So far, we have discussed similarity measures for references. For the clustering approach to entity resolution, we need to define similarities between clusters of references. For collective entity resolution, we define the similarity of two clusters c_i and c_j as:

$$sim(c_i, c_j) = (1 - \alpha) \times sim_A(c_i, c_j) + \alpha \times sim_R(c_i, c_j), \qquad 0 \leq \alpha \leq 1 \qquad (2)$$

where $sim_A()$ is the similarity of the attributes and $sim_R()$ is the relational similarity between the references in the two entity clusters. On analyzing Equation (2), we can see that it reduces to attribute-based similarity for $\alpha = 0$. Also, the relational aspect of the similarity measures distinguishes it from the naive relational similarity measure from Equation (1). While naive relational similarity measures the attribute similarity of the related references that are connected through hyperedges, here we consider the labels of related clusters that represent entities. This similarity is dynamic in nature which is one of the most important and interesting aspects of the collective approach. For attribute-based and naive relational resolution, the similarity between two references is fixed. In contrast, for collective resolution, the similarity of two clusters depends on the current cluster labels of their neighbors and therefore changes as their labels are updated. In our example, the similarity between W. Wang and W. W. Wang increases once the Ansari references are given the same cluster label.

As we have mentioned earlier, similarity measures for attributes have been studied in great detail. Our focus is on measuring relational similarity between two clusters of references. The references in each cluster c are connected to other references via hyperedges. For collective entity resolution, relational similarity considers the cluster labels of all these connected references. Recall that each reference r is associated with one or more hyperedges in $\mathcal{H}$. Therefore, the set of hyperedges $c.H$ that we need to consider for an entity cluster c is defined as

$$c.H = \bigcup_{r \in \mathcal{R} \wedge r.C = c} \{h \mid h \in \mathcal{H} \ \wedge \ r \in h.R\}.$$

These hyperedges connect c to other clusters. The relational similarity for two clusters needs to compare their connectivity patterns to other clusters.

For any cluster c, the set of other clusters to which c is connected via its hyperedge set $c.H$ form the neighborhood $Nbr(c)$ of cluster c:

$$Nbr(c) = \bigcup_{h \in c.H, r \in h.R} \{c_j \mid c_j = r.C\}.$$

This defines the neighborhood as a set of related clusters, but the neighborhood can also be defined as a bag or multiset in which the multiplicity of the different neighboring clusters is preserved. We will use $Nbr_B(c_i)$ to denote the bag of neighboring clusters. In our example in Figure 4(b), the neighborhood of the cluster for *Wang*1 consists of the clusters for *Ansari* and *Chen*1; alternatively it is the bag of clusters $\{Ansari, Ansari, Ansari, Chen1\}$. Note that we do not constrain the definition of the neighborhood of a cluster to exclude the cluster itself. In Section 5.6, we discuss how such constraints can be handled when required by the domain.

For the relational similarity between two clusters, we look for commonness in their neighborhoods. This can be done in many different ways as we explore in the following section.

5. NEIGHBORHOOD SIMILARITY MEASURES FOR COLLECTIVE RESOLUTION

We have seen how the neighborhood of a cluster of references can be represented as a set (or alternatively as a bag or multiset) of cluster labels and that we can compute relational similarity between two clusters by considering the similarity of their neighborhoods. Many different metrics have been proposed and evaluated in the literature for measuring commonness between sets. For example Liben-Nowell and Kleinberg [2003] study their use for prediction tasks in social networks. Here we adapt and modify some of these measures and study their applicability for entity resolution.

5.1 Common Neighbors

This is the simplest approach for measuring commonness between sets and counts the number of elements that occur in both. For two clusters, c_i and c_j, their common neighbor score is defined as

$$CommonNbrScore(c_i, c_j) = \frac{1}{K} \times \left|Nbr(c_i) \bigcap Nbr(c_j)\right|, \tag{3}$$

where K is a large enough constant such that the measure is less than 1 for all pairs of clusters. For two references John Smith and J. Smith, where attribute similarity is not very informative, this score measures the overlap in their connected entities. The greater the number of common entities, the higher the possibility that the two references refer to the same entity as well.

This definition ignores the frequency of connectivity to a neighbor. Suppose John Smith has collaborated with the entity J. Brown several times, while J. Smith has done so only once. To investigate if this information is relevant for entity resolution, we also define a common neighbor score with frequencies that takes into account multiple occurrences of common clusters in the neighborhoods:

$$CommonNbrScore + Fr(c_i, c_j) = \frac{1}{K'} \times \left|Nbr_B(c_i) \bigcap Nbr_B(c_j)\right|. \tag{4}$$

5.2 Jaccard Coefficient

The main shortcoming of the common neighbor score is the normalizing constant K which is the same over all pairs of clusters. Consider the situation where we have two John Smith clusters, c_1 and c_2, both of which have the same number of neighbors in common with the J. Smith cluster, c_3. Then they are equally similar to c_3 in terms of the common neighbor score. Suppose that all of c_1's neighbors are shared with c_3, while c_2 has a very large neighborhood and only a small fraction of it is shared with c_3. When entities have large neighborhoods, finding shared neighbors by chance becomes more likely. In this case, we may want the similarity between c_1 and c_3 to be greater than the similarity between c_2 and c_3. We can get around this issue by taking into account the size of neighborhood. This gives us the Jaccard coefficient for two clusters:

$$JaccardCoeff\,(c_i, c_j) = \frac{|Nbr(c_i) \bigcap Nbr(c_j)|}{|Nbr(c_i) \bigcup Nbr(c_j)|}. \tag{5}$$

As before, we may consider neighbor counts to define the Jaccard coefficient with frequencies, $JaccardCoeff + Fr(c_i, c_j)$, by using $Nbr_B(c_i)$ and $Nbr_B(c_j)$ in the definition.

5.3 Adamic/Adar Similarity

Both the common neighborhood measure and Jaccard coefficient consider all cluster labels in the neighborhood as equally important and significant for determining coreference. However this is not always desirable. If a cluster is frequently linked with many different clusters, then its presence in a shared neighborhood is not as significant as a cluster which is less frequent. This is similar to the idea behind inverse document frequency in the commonly used TF-IDF scheme in information retrieval. Adamic and Adar [2003] use this idea for predicting friendship from Web page features. They proposed a similarity measure between two Web pages X and Y that individually considers the significance of each element that they share and assigns weights to them accordingly. This has come to be called the Adar/Adamic score:

$$similarity(X, Y) = \sum_{\text{shared feature } z} \frac{1}{\log(frequency(z))}.$$

Liben-Nowell and Kleinberg [2003] adapted this idea for the task of link prediction in social networks considering node neighborhoods where they used the size of a node's neighborhood for measuring frequency or commonness. We generalize this idea to propose a class of Adar/Adamic measures for entity resolution. If the uniqueness of a cluster label c (or a shared feature, in general) is denoted as $u(c)$, then we define the Adar similarity score of two clusters, c_i and c_j, as

$$Adar(c_i, c_j) = \frac{\sum_{c \in Nbr(c_i) \cap Nbr(c_j)} u(c)}{\sum_{c \in Nbr(c_i) \cup Nbr(c_j)} u(c)}, \tag{6}$$

where the denominator normalizes the score. Now the Jaccard coefficient can be viewed as a special case of the Adar score when all nodes are equally unique. Also, observe that without the normalization Equation (6) reduces to the similarity score of Liben-Nowell and Kleinberg [2003] for

$$u(c) = \frac{1}{\log(|Nbr(c)|)}. \tag{7}$$

We refer to the Adar score that uses this definition of uniqueness as the AdarNbr score. As before, we evaluate two versions, AdarNbr that considers the set of neighbors and AdarNbr+Fr that takes into account the multiplicity of the neighbors.

5.4 Adar Similarity with Ambiguity Estimate

While using the neighborhood size of a cluster to measure its uniqueness has been shown to work well in link prediction applications, it may not be appropriate for entity resolution. For entity resolution applications, we do not directly know the neighbors for each entity from the data. The true neighborhood size for any entity cluster is known only after the entity graph has been correctly reconstructed. So using the neighborhood size as a measure of uniqueness at any intermediate stage of the resolution algorithm is incorrect and is an overestimate of the actual neighborhood size.

As an alternative, we can use a definition of uniqueness which incorporates a notion of the ambiguity of the names found in the shared neighborhood. To understand what this means, consider two references with name A. Aho. Since

Aho can be considered an uncommon name, they are very likely to be the same person. In contrast, two other references with a common name such as L. Li are less likely to be the same person. So we define the ambiguity $Amb(r.Name)$ of a reference name as the probability that multiple entities share that particular name.

Intuitively, clusters which share neighbors with uncommon names are more likely to refer to the same entity and should be considered more similar. We define the uniqueness of a cluster c as inversely proportional to the average ambiguity of its references:

$$u(c) = \frac{1}{Avg_{r \in c}(Amb(r.Name))}. \tag{8}$$

In general, this approach is not specific to names and can be used with any attribute of the references. We refer to an Adar similarity score which uses this definition of uniqueness as AdarName when applied to the set of neighbors and AdarName+Fr to refer to the measure applied to the bag of neighbors.

The Adar-Name measure is defined in terms of the ambiguity of a reference's name. There are a number of ways to estimate the ambiguity of a name. One scheme that works quite well in our domains is to estimate the probability that two randomly picked references with $Name = n$ correspond to different entities. For a reference attribute A_1, denoted $R.A_1$, a naive estimate for the ambiguity of a value of n for the attribute is:

$$Amb(r.A_1) = \frac{|\sigma_{R.A_1 = r.A_1}(R)|}{|R|},$$

where $|\sigma_{R.A_1 = r.A_1}(R)|$ denotes the number of references with value $r.A_1$ for A_1. This estimate is clearly not good since the number of references with a certain attribute value does not always match the number of different entity labels for that attribute. We can do much better if we have an additional attribute A_2. Given A_2, the ambiguity for value of A_1 can be estimated as

$$Amb(r.A_1 \mid r.A_2) = \frac{|\delta(\pi_{R.A_2}(\sigma_{R.A_1 = r.A_1}(R)))|}{|R|},$$

where $|\delta(\pi_{R.A_2}(\sigma_{R.A_1 = r.A_1}(R)))|$ is the number of distinct values observed for A_2 in references with $R.A_1 = r.A_1$. For example, we can estimate the ambiguity of a last name by counting the number of different first names observed for it. This provides a better estimate of the ambiguity of any value of an attribute A_1 when A_2 is not correlated with A_1. When multiple uncorrelated attributes A_i are available for references, this approach can be generalized to obtain better ambiguity estimates.

5.5 Higher-Order Neighborhoods

Analysis of the commonness of neighborhoods can be viewed as an investigation of paths of length two between two clusters. We also investigate whether higher-order neighborhoods play a role in detecting coreference. In addition to the neighborhood similarity measures described, we also evaluate measures which take into account collaboration paths of length three. As the clusters change, it becomes computationally infeasible to recompute all paths between all cluster pairs. Instead, we calculate the second-order neighborhood $Nbr^2(c)$ for a cluster c by recursively taking the set union (alternatively, multiset union) of the neighborhoods of all neighboring clusters: $Nbr^2(c) = \bigcup_{c' \in Nbr(c)} Nbr(c')$. For paths of length three to be present between two clusters, c_i and c_j, there must be intersections between the $Nbr(c_i)$ and $Nbr^2(c_j)$ or vice versa. Then, to find the similarity over paths of length 3 or less for c_i and c_j, we take the average of the similarities over length-2 paths and length-3 paths:

$$\begin{aligned} Path3Sim(c_i, c_j) = & \frac{1}{3}[Jaccard(Nbr(c_i), Nbr(c_j)) \\ & + Jaccard(Nbr^2(c_i), Nbr(c_j)) \\ & + Jaccard(Nbr(c_i), Nbr^2(c_j))]. \end{aligned} \tag{9}$$

5.6 Negative Constraints From Relationships

The common relational structure we have considered so far can be seen as positive evidence for inferring that two author references refer to the same underlying author entity. Additionally, there may be negative constraints as well for entity resolution arising from relationships. For example, in many relational domains, two references appearing in the same hyperedge cannot refer to the same entity. As a real bibliographic example, consider a paper with coauthors M. Faloutsos, P. Faloutsos and C. Faloutsos. Despite the similarity of the uncommon last name, in reality these references correspond to distinct author entities. So, for bibliographic domains, we can add a constraint that two references which cooccur cannot refer to the same entity. In domains other than citation data, there may be different relational constraints. In general, we can have a set of negative relational constraints that clusters need to satisfy. We take these into account by setting the similarity between two cluster pairs in Equation (2) to zero if merging them violates any of the relational constraints.

6. RELATIONAL CLUSTERING ALGORITHM

Given the similarity measure for a pair of reference clusters, we use a greedy agglomerative clustering algorithm that finds the closest cluster pair at each step and merges them. High-level pseudocode for the algorithm is provided in Figure 5. In this section, we discuss several important implementation and performance issues regarding relational clustering algorithms for entity resolution.

6.1 Blocking to Find Potential Resolution Candidates

Unless the datasets are small, it is impractical to consider all possible pairs as potential candidates for merging. Apart from the scaling issue, most pairs checked by an $O(n^2)$ approach will be rejected since usually only about 1% of all pairs are true matches. Blocking techniques [Hernández and Stolfo 1995; Monge and Elkan 1997; McCallum et al. 2000] are usually employed to rule out pairs which are certain to be nonmatches. The goal is to separate references into possibly overlapping buckets and only pairs of references within

```
1.    Find similar references using blocking
2.    Initialize clusters using bootstrapping

3.    For clusters $c_i, c_j$ such that similar($c_i, c_j$)
4.        Insert $\langle sim(c_i, c_j), c_j, c_j \rangle$ into priority queue

5.    While priority queue not empty
6.        Extract $\langle sim(c_i, c_j), c_i, c_j \rangle$ from queue
7.        If $sim(c_i, c_j)$ less than threshold, then stop
8.        Merge $c_i$ and $c_j$ to new cluster $c_{ij}$
9.        Remove entries for $c_i$ and $c_j$ from queue
10.       For each cluster $c_k$ such that similar($c_{ij}, c_k$)
11.           Insert $\langle sim(c_{ij}, c_k), c_{ij}, c_k \rangle$ into queue
12.       For each cluster $c_n$ neighbor of $c_{ij}$
13.           For $c_k$ such that similar($c_k, c_n$)
14.               Update $sim(c_k, c_n)$ in queue
```

Fig. 5. High-level description of the relational clustering algorithm

each bucket are considered as potential matches. The relational clustering algorithm uses the blocking method as a blackbox and any method that can quickly identify potential matches minimizing false negatives can be used. We use a variant of an algorithm proposed by McCallum et al. [2000] that we briefly describe.

The algorithm makes a single pass over the list of references and assigns them to buckets using an attribute similarity measure. Each bucket has a representative reference that is the most similar to all references currently in the bucket. For assigning any reference, it is compared to the representative for each bucket. It is assigned to all buckets for which the similarity is above a threshold. If no similar bucket is found, a new bucket is created for this reference. A naive implementation yields a $O(n(b + f))$ algorithm for n references and b buckets and when a reference is assigned to at most f buckets. This can be improved by maintaining an inverted index over buckets. For example, when dealing with names, for each character, we maintain the list of buckets storing last names starting with that character. Then the buckets can be looked up in constant time for each reference leading to an $O(nf)$ algorithm.

6.2 Relational Bootstrapping

Each iteration of the relational clustering algorithm makes use of clustering decisions made in previous iterations. This is achieved by measuring the shared neighborhood for similar clusters as explained in Section 4.3. But if we begin with each reference in a distinct cluster, then initially there are no shared neighbors for references that belong to different hyperedges. So the initial iterations of the algorithm have no relational evidence to depend upon. As a result, the relational component of the similarity between clusters would be zero and merges would occur based on attribute similarity alone. Many of such initial merges can be inaccurate, particularly for the references with ambiguous attribute values. To avoid this, we need to bootstrap the clustering algorithm so that each reference is not assigned to a distinct cluster. Specifically, if we are confident that some reference pair is coreferent, then they should be assigned to the same initial cluster. However, precision is crucial for the bootstrap process since our algorithm cannot undo any of these initial merge operations. Observe that this bootstrapping is not necessary for approaches that are not collective. For such approaches, the decision for any reference pair is the same irrespective of the decisions for other pairs. So bootstrapping does not have any effect on subsequent decisions. In this subsection, we describe our bootstrapping scheme for relational clustering that makes use of the hyperedges for improved bootstrap performance. The basic idea is very similar to the naive relational approach described in Section 4.2 with the difference that we use exact matches instead of similarity for attributes. To determine if any two references should be assigned to the same initial cluster, we first check if their attributes match exactly. For references with ambiguous attributes, we also check if the attributes of their related references match. We now discuss this in greater detail.

The bootstrap scheme goes over each reference pair that is potentially coreferent (as determined by blocking) and determines if it is a bootstrap candidate. First, consider the simple bootstrap scheme that looks only at the attributes of two references. It determines which attribute values are ambiguous and which are not using a data-based ambiguity estimate as described in Section 5.4. References with ambiguous attribute values are assigned to distinct clusters. Any reference pair whose attribute values match and are not ambiguous is considered to be a bootstrap candidate.

The problem with this simple approach is that it assigns all references with ambiguous attributes to distinct clusters leading to poor recall in datasets with high ambiguity. When hyperedges are available, they can be used as evidence for bootstrapping of ambiguous references. A pair of ambiguous references form a bootstrap candidate if their hyperedges match. Two hyperedges, h_1 and h_2, are said to have a *k-exact-match* if there are at least k pairs of references (r_i, r_j), $r_i \in h_1.R$, $r_j \in h_2.R$ with exact matching attributes, that is, $r_i.A = r_j.A$. Two references, r_1 and r_2, are bootstrap candidates if any pair of their hyperedges have a k. As a bibliographic example, two references with the name W. Wang will not be merged during bootstrapping on the basis of the name alone. However, if the first Wang reference has coauthors A. *Ansari* and C. Chen, and the second Wang has coauthor A. *Ansari*, then they have a 1-exact-match and, depending on a threshold for k, they would be merged. The value of k for the hyperedge test depends on the ambiguity of the domain. A higher value of k should be used for domains with high ambiguity. Also, when matching hyperedges, references with ambiguous attributes are not considered for matches in high ambiguity domains. For example, C. Chen may not be considered for a coauthor match since it is a common name.

Other attributes of the references, and also of the hyperedges, when available, can be used to further constrain bootstrap candidates. Two references are considered only if these other attributes do not conflict. In the bibliographic domain, author references from two different papers can be merged only if their languages and correspondence addresses match.

After the bootstrap candidates are identified, the initial clusters are created using the union-find approach so that any two references that are bootstrap candidates are assigned to the same initial cluster. In addition to improving accuracy of the relational clustering algorithm, bootstrapping reduces execution time by significantly lowering the initial number of clusters without having to find the most similar cluster-pairs or perform expensive similarity computations.

6.3 Merging Clusters and Updating Similarities

Once the similar clusters have been identified and bootstrapping has been performed, the algorithm iteratively merges the most similar cluster pair and updates similarities until the similarity drops below some specified threshold. This is shown in lines 5–14 of Figure 5. The similarity update steps for related clusters in lines 12–14 are the key steps that distinguish collective relational clustering from a traditional agglomerative clustering algorithm. In order to perform the update steps efficiently, indexes need to be maintained for each cluster. In this section, we describe the data structure that we maintain for this purpose.

In addition to its list of references, we maintain three additional lists with each cluster. First, we maintain the list of similar clusters for each cluster. The second list keeps track of all neighboring clusters. Finally, we keep track of all the queue entries that involve this cluster. For a cluster that has a single reference r, the similar clusters are those that contain references in the same bucket as r after blocking. Also, the neighbors for this cluster are the clusters containing references that share a hyperedge with r. Then, as two clusters merge to form a new cluster, all of these lists can be constructed locally for the new cluster from those of its parents. All of the update operations from lines 9–14 can be performed efficiently using these lists. For example, updates for related clusters are done by first accessing the neighbor list and then traversing the similar list for each of them.

6.4 Complexity Analysis

Now that we have described each component of our relational clustering algorithm, let us analyze its time complexity. First, we look at how the number of similarity computations required in lines 3–4 of Figure 5 is reduced by the blocking method. We consider the worst-case scenario where the bootstrapping approach does not reduce the number of clusters at all. We need to compare every pair of references within each bucket. Suppose we have n references that are assigned to b buckets with each reference being assigned to at most f buckets. Then, using an optimistic estimate, we have nf/b references in each bucket, leading to $O((nf/b)^2)$ comparisons per bucket and a total of $O(n^2 f^2/b)$ comparisons. In all of our discussion, we assume that the number of buckets is proportional to the number of references, that is, b is $O(n)$. Additionally, assuming that f is a small constant independent of n, we have $O(n)$ computations. It should be noted that this is not a worst-case analysis for the bucketing. A bad bucketing algorithm that assigns $O(n)$ references to any bucket will lead to $O(n^2)$ comparisons.

Now, let us look at the time taken by each iteration of the algorithm. To analyze how many update/insert operations are required, we assume that for each bucket that is affected by a merge operation, all the $O((nf/b)^2)$ computations need to be redone. Then we need to find out how many buckets may be affected by a merge operation. We say that two buckets are connected if any hyperedge connects two references in the two buckets. Then if any bucket is connected to k other buckets, each merge operation leads to $O(k(nf/b)^2)$ update/insert operations. This is still only $O(k)$ operations when f is a constant independent of n and b is $O(n)$. Using a binary-heap implementation for the priority queue, the extract-max and each insert and update operation take $O(\log q)$ time, where q is the number of entries in the queue. So the total cost of each iteration of the algorithm is $O(k \log q)$.

Next, we count the total number of iterations that our algorithm may require. In the worst case, the algorithm may have to exhaust the priority queue before the similarity falls below the threshold. So we need to consider the number of merge operations that are required to exhaust a queue that has q entries. If the merge tree is perfectly balanced, then the size of each cluster is doubled by each merge operation and as few as $O(\log q)$ merges are required. However, in the worst case, the merge tree may be q-deep requiring as many as $O(q)$ merges. With each merge operation requiring $O(k \log q)$ time, the total cost of the iterative process is $O(qk \log q)$.

Finally, in order to put a bound on the initial size q of the priority queue, we again consider the worst-case scenario where bootstrapping does not reduce the number of initial clusters. This results in $O(n^2 f^2/b)$ entries in the queue as shown earlier. Since this is again $O(n)$, the total cost of the algorithm can be bounded by $O(nk \log n)$. The one cost that we have not considered so far is that of bootstrapping. We can analyze the bootstrapping by considering it as a sequence of cluster merge operations that do not require any updates or inserts to the priority queue. Then the worst-case analysis of the number of iterations accounts for the bootstrapping as well.

To see how this compares against the attribute and naive relational baselines, observe that they need to take a decision for each pair of references in a bucket. This leads to a worst-case analysis of $O(n)$ using the same assumptions as before. However, each similarity computation is more expensive for the naive relational approach (Equation (1)) than the attribute-based approach, since the former requires a pairwise match to be computed between two hyperedges.

7. EXPERIMENTAL EVALUATION

We evaluated our relational entity resolution algorithm on several real-world and synthetic datasets. We begin with a description of our experiments on real bibliographic datasets.

7.1 Evaluation on Bibliographic Data

Our real-world datasets describe publications in several different scientific research areas. As in our running example, the goal is to use coauthor relationships in the papers to help discover the underlying author entities in the domain and map the author references to the discovered author entities. We first describe the datasets in more detail, and then describe our evaluation and results.

7.1.1 *Datasets*

CiteSeer. The CiteSeer dataset contains 1,504 machine learning documents with 2,892 author references to 1,165 author entities. For this dataset, the only attribute information available is author name. The full last name is always given, and, in some cases, the author's full first name and middle name are given and other times only the initials are given. The dataset was originally created by Giles et al. [1998], and the version that we use includes the author entity ground truth provided by Aron Culotta and Andrew McCallum, University of Massachusetts, Amherst.

arXiv. The arXiv dataset describes high-energy physics publications. It was originally used in KDD Cup 2003[1]. It contains 29,555 papers with 58,515 references to 9,200 authors. The attribute information available for this dataset is also just the author name with the same variations in form as described previously. The author entity ground truth for this data set was provided by David Jensen, University of Massachusetts, Amherst.

BioBase. Our third dataset, describing biology publications, is the Elsevier BioBase dataset[2] which was used in a recent IBM KDD-Challenge competition. It was created by selecting all Elsevier publications on Immunology and Infectious Diseases between years 1998 and 2001. It contains 156,156 publications with 831,991 author references. Unlike arXiv and CiteSeer that have complete as well as initialed author names, in BioBase, all of the first names and middle names are abbreviated. However the BioBase dataset has other attributes which we use for resolution including keywords, topic classification, language, country of correspondence, and affiliation of the corresponding author. There is a wide variety in the data with 20 languages, 136 countries, 1,282 topic classifications, and 7,798 keywords. Entity labels are available only for the top 100 author names with the highest number of references. We evaluate entity resolution performance for BioBase over 10,595 references that have these 100 names, although our collective resolution algorithm requires resolving many of the other references as well.

Ground truth was determined for all of these datasets by the owners using a combination of automatic and manual strategies. The process is not completely free from errors and we had to perform additional cleaning for some CiteSeer and arXiv references in the course of our experiments. For BioBase, 97% of the labels are estimated to be correct.

Despite the common underlying domain, these datasets vary in a number of important ways. The most important difference is in the inherent uncertainty in the name references. We introduce two measures, which we refer to as *ambiguity* (corresponding to the disambiguation aspect of resolution) and

[1]http://www.cs.cornell.edu/projects/kddcup/index.html.
[2]http://help.sciencedirect.com/robo/projects/sdhelp/about_biobase.htm.

dispersion (corresponding to the identification aspect), to measure the uncertainty in the data. We consider a name (last name and first initial) to be ambiguous if multiple entities share that name. In CiteSeer dataset, only 3 out of 1,185 names are ambiguous, and the average number of entities per ambiguous name is 2.33 (the maximum is 3). For arXiv, 374 of the 8,737 names are ambiguous, and the average number of entities for these ambiguous names is 2.41 (the maximum is 11). For BioBase, the ambiguity is much higher, 84 of the 100 names are ambiguous. The number of entities for each name ranges from 1 to 100 with an average of 32. We introduce dispersion as another measure of the inherent difficulty of the entity resolution problem for a domain. The dispersion for an entity is the number of distinct observed names for each entity. For CiteSeer, 202 out of the 1,164 entities have multiple recorded names, the average and maximum dispersion are 1.24 and 8, respectively. In contrast, 3,083 out of 8,967 entities for arXiv are dispersed over multiple names, and the average dispersion is 1.44 and the maximum is 10. Since we do not have complete ground truth for the BioBase dataset, dispersion cannot be directly measured. Apart from the level of uncertainty, BioBase differs significantly from the other two datasets in terms of its hyperedge structure. For BioBase, the number of author references per publication ranges from 1 to 100 with the average being 5.3. In comparison, the averages are 1.92 and 1.98, respectively, for CiteSeer and arXiv, the range being 1 to 10 for both.

7.1.2 *Evaluation.* We compare attribute-based entity resolution (**A**), naive relational entity resolution (**NR**) that uses attributes of related references, and our collective relational entity resolution (**CR**). For the first two algorithms, we also consider variants which perform transitive closures over the pairwise decisions (**A*** and **NR***).

In order to measure the performance of our algorithms, we consider the correctness of the pairwise coreference decisions over all references. We evaluate the pairwise decisions using the F1 measure which is the harmonic mean of precision and recall. For additional insight, we also plot precision-recall curves. For a fair comparison, we consider the best F1 for each of these algorithms over all possible thresholds for determining matches.

7.1.3 *Experimental Details.* For comparing attributes, which is required for all of the algorithms, we use the Soft TF-IDF similarity for names [Cohen et al. 2003; Bilenko et al. 2003] since it has been shown to perform well for name-based entity resolution. Essentially, Soft TF-IDF augments the TF-IDF similarity for matching token sets with approximate token matching using a secondary string similarity measure. Jaro-Winkler is reported to be the best secondary similarity measure for Soft TF-IDF, but for completeness, we also experiment with the Jaro and the Scaled Levenstein measures. Scaled Levenstein belongs to the edit-distance family of similarity measures that assigns unit cost to each edit operation and normalizes the result. Jaro and Jaro-Winkler do not belong to the edit-distance family. They measure the number and order of common characters between strings. Jaro-Winkler is a variant of Jaro that also considers the longest common prefix [Cohen et al. 2003]. They are both well

Table I. Performance of Different Algorithms on the CiteSeer, arXiv and BioBase Datasets (We report the mean and the standard deviations (within parenthesis) of the F1 scores obtained using Scaled Levenstein, Jaro, and Jaro-Winkler as secondary similarity measure within Soft TF-IDF)

	CiteSeer	arXiv	BioBase
A	0.980 (0.001)	0.974 (0.002)	0.568 (0)
A*	0.990 (0.001)	0.967 (0.003)	0.559 (0)
NR	0.981 (0.006)	0.975 (0.016)	0.710 (0)
NR*	0.991 (0.002)	0.972 (0.039)	0.753 (0)
Bootstrap H-Amb	0.217 (0)	0.119 (0)	0.452 (0)
Bootstrap L-Amb	0.942 (0)	0.977 (0)	0.317 (0)
CR	0.995 (0)	0.985 (0)	0.819 (0)

suited for short strings such as personal names. In the case of BioBase, where we had other multivalued attributes to make use of besides names, we used TF-IDF similarity.

Since for CiteSeer and arXiv, it is not feasible to consider all pairs as potential duplicates, blocking is employed to extract the potential matches. This approach retains $\sim$99% of the true duplicates for both CiteSeer and arXiv by allowing at most two character transpositions for last names. We use bootstrapping for our relational clustering algorithm (**CR**) for all three datasets. We use bootstrap for low-ambiguity domains with $k = 1$ for CiteSeer and arXiv, and bootstrap for high-ambiguity domains with $k = 2$ for BioBase. Recall that our relational clustering algorithm (**CR**) and the naive relational approach (**NR** and **NR***) both use a combination weight α. We measure performance of both algorithms at 20 different values of α between 0 and 1 and report the best performance for each of them over this range. For estimating ambiguity of references for AdarName, we use last names with first initial as the secondary attribute. This resulted in very good estimates of ambiguity—the ambiguity estimate for a name is strongly correlated (correlation coefficient 0.8) with the number of entities for that name.

7.1.4 *Results.* Table I gives an overview of the F1 results of the various algorithms on our three datasets. Recall that our collective relational clustering uses bootstrapping to initialize the clusters. In addition to our three entity resolution approaches that we have discussed, we also include for comparison the two bootstrapping approaches, one for low-ambiguity domains (**Bootstrap L-Amb**) that is used by **CR** for CiteSeer and arXiv, and the other for high-ambiguity data (**Bootstrap H-Amb**) that is employed for BioBase. For **CR**, Table I records the performance for the best neighborhood similarity measure, which is Jaccard for CiteSeer and arXiv, and AdarName for BioBase. As mentioned earlier, there are several possible choices for the secondary string metric used with the Soft TD-IDF similarity for comparing the names. The results above are the averages using three choices—Scaled Levenstein, Jaro, and Jaro-Winkler—with the standard deviation shown in parenthesis.

First, note that the standard deviation in Table I measures sensitivity of entity resolution performance in terms of the similarity measure used for names. We can see that the results are not very sensitive to the secondary string metric choice. In fact, for collective relational entity resolution (**CR**), the choice is irrelevant and for the BioBase dataset, in which we have additional features besides the names, the choice is also irrelevant. For the cases in which there were some small differences, Scaled Levenstein was most often, but not always, the best.

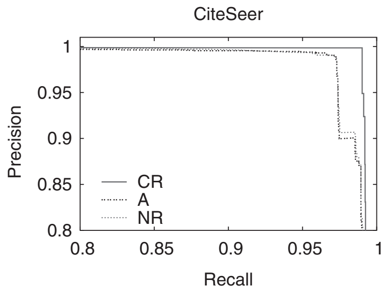
Second, looking at the first line in Table I, note the differences in performance for attribute-based entity resolution (**A**) for the three datasets. The attribute-based algorithm performs remarkably well for the CiteSeer database, and its performance on the arXiv dataset is also respectable. This is in keeping with our earlier observation about the hardness of the datasets in terms of ambiguity and dispersion. The CiteSeer dataset has very little ambiguity, and arXiv is only moderately more ambiguous. When datasets are not ambiguous, all dispersed entities can be successfully identified simply by raising the discrimination threshold for determining duplicates. This increases recall without generating false positives. However, this is not possible when there is significant ambiguity in the data as we see in the case of BioBase. The performance of the bootstrapping algorithms highlight the same trend. For CiteSeer and arXiv, the low-ambiguity version (**Bootstrap L-Amb**) performs almost as well as the attribute baseline. In a higher ambiguity dataset such as BioBase, it performs many incorrect matches. The high-ambiguity bootstrap strategy (**Bootstrap H-Amb**), which is more cautious for ambiguous references, performs poorly for CiteSeer and arXiv due to low recall but improves performance over **Bootstrap L-Amb** for BioBase by increasing precision.

Next, observe that the naive relational entity resolution algorithm (**NR**) which uses attributes of related references in its similarity calculations improves performance over **A** only marginally for CiteSeer and arXiv, while the improvement is quite significant in the case of BioBase. This suggests that, while the attributes of related entries can help in disambiguation in domains with high ambiguity, there may not be much improvement for less ambiguous domains.

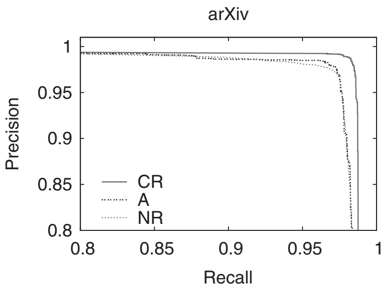
The table also shows that the effect of transitive closure on entity resolution performance also varies over the datasets. While it improves performance for both **A** and **NR** for CiteSeer and arXiv, in the case of BioBase, it helps **NR** but not **A**. A possible explanation is that transitive closure improves performance in domains with low ambiguity, but it may result in false identifications in datasets with higher ambiguity.

Finally, note that across all three datasets, the collective relational entity resolution algorithm (**CR**) performs the best. The gains for the less ambiguous domains are more modest, while in the most ambiguous domain, the gain is quite significant. In addition, the performance improvements of **CR** over **NR** highlights the importance over considering the identities of related references rather than just their attributes. Also, since the performance is insensitive to the choice of attribute similarity used, overall **CR** is more robust than **A** and **NR**.

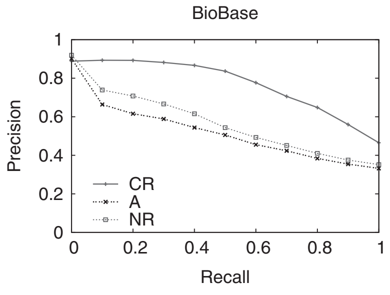
In Figure 6, we show the precision-recall curves for the three algorithms in our three datasets using the Jaro similarity measure for names. For CiteSeer and arXiv, we have zoomed in to show the region where the curves differ



(a)



(b)



(c)

Fig. 6. Precision vs recall for (a) CiteSeer, (b) arXiv, and (c) BioBase.

significantly from each other. The plots confirm that the benefits of **CR** are significantly larger in domains with high ambiguity such as BioBase.

Recall that **CR**, **NR**, and **NR*** involve a weighting parameter α for combining attribute and relational similarity. As mentioned earlier, the numbers in Table I record the best performance over different values of α for each of these algorithms. The best performance is not always achieved for the same value of α for different datasets or for the 100 different reference names in BioBase. In Figure 7, we see how the performance of the different algorithms changes over different values of α for the three datasets. For BioBase, we plot the average performance over all 100 reference names for a particular value of α. As a reference, we also show the performances of **A** and **A*** which do not depend on α. We can make two interesting observations from the plots. First, the relational clustering algorithm **CR** consistently outperforms the naive relational baselines (**NR**) and (**NR***) for all values of α for all three datasets. Secondly, for CiteSeer and arXiv, the naive relational approach outperforms the attribute-only baseline only marginally for small values of α and then its performance drops significantly at higher values. It is more stable for BioBase but performs still drops below the attribute-only baseline for high values of α. The performance of **CR** is significantly more stable over varying α for all three datasets in that it never falls below that of the baselines. This is another validation of the usefulness of resolving related references instead of considering their similarities. Note that for CiteSeer and arXiv, performance does not improve as much beyond $\alpha = 0$ as for BioBase. This is due to a ceiling effect and the sparsity

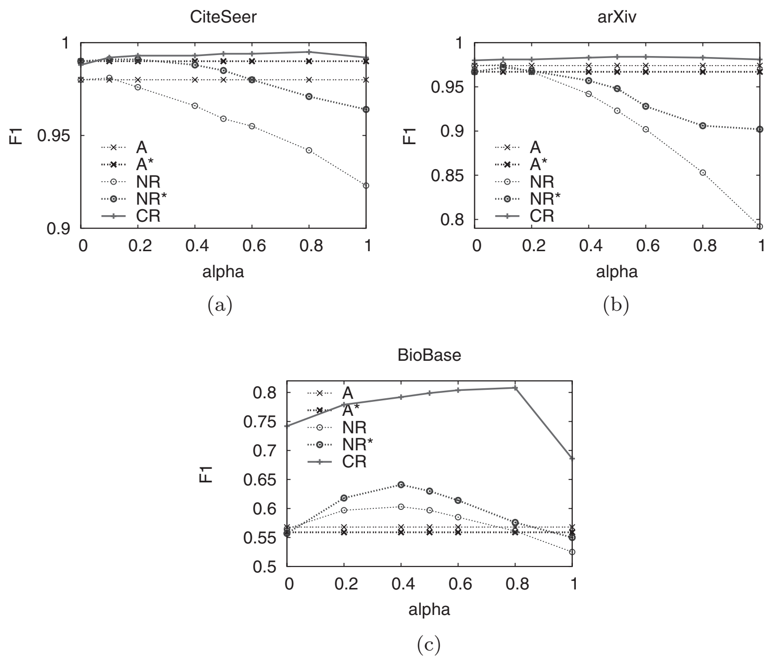


Fig. 7. Entity resolution performance at different values of α for (a) CiteSeer, (b) arXiv, and (c) BioBase.

of available relationships in these two datasets. It should also be pointed out that the performance gap between the **CR** and the attribute baseline at $\alpha = 0$ comes from the bootstrapping phase which makes use of both attributes and relationships as explained in Section 6.2.

Now we explore **CR** in more depth by comparing the performance of the algorithm using different graph-based similarity measures. Table II shows the performance of the collective relational entity resolution with the different proposed measures on our three datasets. There is little difference in performance on the CiteSeer and arXiv datasets. The simplest measure, **Common**, correctly retrieves almost all duplicates in CiteSeer. Recall that due to the blocking approach, 100% recall—and therefore an F1 score of 1.0—is not attainable for these two datasets.

There is a bit more of an impact on the BioBase results. The numbers do not provide enough evidence to validate the use of frequencies (+**Fr**) for comparing neighborhoods. It improves performance in some cases and affects it adversely in others. So, in the following discussion, we concentrate on the basic similarity measures where the cluster neighborhood is treated as a set rather than as a bag. We make four observations:

—**Jaccard** similarity improves performance over **Common** neighbors. Recall that the difference between the two is in the normalization. This shows the importance of considering the size of the common neighborhood as a fraction of the entire neighborhood.

Table II. F1 Performance for Collective Relational Entity Resolution Using Different Neighborhood Similarity Measures in the Three Bibliographic Datasets

	CiteSeer	arXiv	BioBase
Common	0.994	0.984	0.814
Common+Fr	0.994	0.984	0.816
Jaccard	0.994	0.985	0.818
Jaccard+Fr	0.995	0.985	0.818
AdarNbr	0.994	0.984	0.815
AdarNbr+Fr	0.994	0.984	0.816
AdarName	0.994	0.985	0.819
AdarName+Fr	0.995	0.984	0.817
Path3Sim	0.994	0.984	0.812

—**AdarNbr** performs worse than **Jaccard**. Recall that Adar similarity considers the importance or uniqueness of each cluster in the shared neighborhood. We pointed out that the connectedness of a shared neighbor is not a reliable indicator in our case since the graph is consolidated over iterations and new hyperedges are added to each cluster. This is validated by the drop in performance as we move to **AdarNbr** from **Jaccard**.

—**AdarName** performs the best over all the graph-based similarity measures. Recall that **AdarName** attempts to capture the uniqueness of a cluster of references, and this, combined with Adar similarity, works the best of all the neighborhood similarity measures on BioBase.

—**Path3Sim** has the lowest performance of all the graph-based measures. Recall that Path3Sim explores second-order neighborhoods for detecting coreference. This suggests that in dense collaboration graphs with many ambiguous entities where distinct entities with similar attributes have common higher-order neighbors, going beyond immediate neighborhood can hurt entity resolution performance. Along similar lines, we show later in our experiments on sythetic data that dense first-order relationships can also be confusing and have an adverse effect on resolution accuracy.

The numbers in Table II show the average performance of the different measures over all 100 instances in BioBase. However, it is not the case that performance is affected for every instance by changing the similarity measure. For example, performance changes in only 22 of the 100 instances when using **Jaccard** similarity instead of **AdarName** similarity as compared to 80 for **Jaccard** compared to the baseline attribute-based similarity measure. In Figure 8, we compare the other measures with Jaccard similarity by measuring the average change in F1-measure over only the affected instances. We see the same trends as discussed above, but the difference between the measures become more pronounced.

7.1.5 *Execution Time.* As we have seen, the use of collective relational entity resolution improves entity resolution performance over attribute-based baselines. However, it is more expensive computationally. Table III records the execution times in CPU seconds of the baseline algorithms and **CR** on the three datasets. All execution times are reported on a Dell Precision 870 server

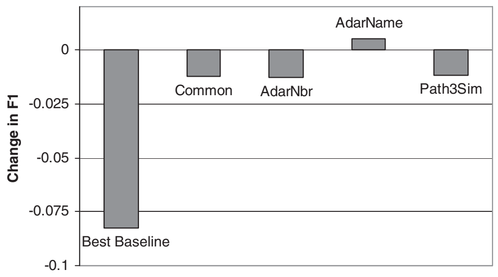


Fig. 8. Comparison of different relational similarity measures against **Jaccard** over only the affected instances in BioBase in each case. The best baseline for BioBase is **NR***.

Table III. Execution Time of Different Algorithms in CPU Seconds

	CiteSeer	arXiv	BioBase
A	0.1	11.3	3.9
NR	0.1	11.5	19.1
CR	2.7	299.0	45.6

with 3.2GHz Intel Xeon processor and 3GB of memory. Let us first consider the execution times for CiteSeer and arXiv. As expected, **CR** takes more time than the baseline, but it is still quite fast. It takes less than 3 seconds for the 2,982 references in CiteSeer and less than 5 minutes for the 58,515 references in arXiv. This is around 9 times as long as the baseline for CiteSeer and 17 times for arXiv. Recall that the complexity of neighborhood similarity is linear in the average connectivity between similar names. The average number of neighbors per entity for CiteSeer is 2.15 and, for arXiv, it is 4.5. So this difference in the degree of relational connectivity explains the difference in execution times for the two datasets. Also, the available attribute for these two datasets is the author name and the average number of authors per publication is very small (1.9) for both. So very little extra computation is needed for the naive relational approach over the attribute baseline.

Now let us consider BioBase. The time recorded for BioBase in Table III is not for cleaning the entire dataset. Rather, it is the average time for collectively resolving references with each of the 100 labeled names. We picked each of the 100 names in the BioBase dataset and extracted all the references relevant for resolving references with that name collectively. The time recorded for BioBase in Table III is the average time taken by different algorithms to resolve these relevant references for each name. The relevant references for each name are found iteratively by including all references that are reachable from the base references that have this name in k steps. The average number of relevant references for each of the 100 instances in 5,510. Table III shows that the difference in execution time between **CR** and the baselines is much smaller for BioBase. One reason for this is that BioBase has many attributes in addition to author name that the attribute-only baseline also need to take into account. Also, the average number of authors per publication is 5.3 for BioBase compared to 1.9 for the other two datasets. This makes the naive relational approach significantly more expensive than the attribute-only baseline.

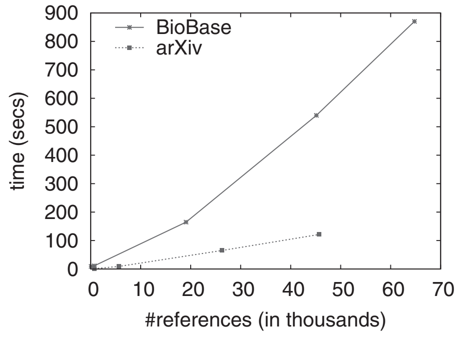


Fig. 9. Execution time of CR with increasing number of references.

Table IV. Comparison of CR with LDA-ER

	CiteSeer		arXiv	
	F1	secs	F1	secs
CR	0.995	2.7	0.985	299
LDA-ER	0.993	240	0.981	36,000

We also used this iterative setup to explore how the collective relational entity resolution algorithm scales with increasing number of references. We created 4 datasets of varying sizes from arXiv and BioBase. Figure 9 shows how **CR** scales with increasing number of references in the dataset. Recall that the complexity of **CR** is $O(nk \log n)$ for n input references where k represents the degree of connectivity among the references.

7.1.5.1 *Comparison With Other Approaches.* Before moving on to experiments on synthetically generated data, we briefly look at how **CR** compares with other collective relational approaches for entity resolution. We have also developed a probabilistic generative model for collective entity resolution (**LDA-ER**) [Bhattacharya and Getoor 2006b] that uses a nonparametric approach to resolve entities by discovering underlying groups of entities from observed relationships. In Table IV, we compare the performance and execution times for **LDA-ER** and **CR** on CiteSeer and arXiv. We are currently not able to compare them for BioBase since extending the **LDA-ER** generative process for multiple attributes is ongoing work. We can see that **CR** is superior in terms of performance. However, it requires a similarity threshold to be specified. In comparison, **LDA-ER** does not require any such threshold and automatically figures out the most likely number of entities. Additionally, **LDA-ER** automatically discovers hidden group structures among the entities from the observed hyperedges. This is often a useful byproduct in the knowledge discovery process. The price for this improvement in the case of **LDA-ER** is significantly longer execution times as shown in Table IV. In terms of complexity, **LDA-ER** runs in $O(t(ng+e))$ time for n references, where t is the number of iterations to converge, g is the number of groups, and e is the number of entities discovered. In general, the algorithm needs to go over many iterations before converging

```
            Creation Stage
1.          Repeat N times
2.              Create random attribute $x$ with ambiguity $p_a$
3.              Create entity $e$ with attribute $x$
4.          Repeat M times
5.              Choose entities $e_i$ and $e_j$ randomly
6.              Set $e_i = Nbr(e_j)$ and $e_j = Nbr(e_i)$

            Generation Stage
7.          Repeat R times
8.              Randomly choose entity $e$
9.              Generate reference $r$ using $\mathcal{N}(e.x, 1)$
10.             Initialize hyper-edge $h = \langle r \rangle$
11.             Repeat with probability $p_c$
12.                 Randomly choose $e_j$ from $Nbr(e)$ without replacement
13.                 Generate reference $r_j$ using $\mathcal{N}(e_j.x, 1)$
14.                 Add $r_j$ hyper-edge $h$
15.             Output hyper-edge $h$
```

Fig. 10. High-level description of synthetic data generation algorithm.

and, the hidden constants in the complexity are also high in comparison to **CR** as the execution times for CiteSeer and arXiv demonstrate. So we can see that collective relational entity resolution (**CR**) compares favorably with other collective resolution approaches both in terms of performance and execution time.

7.2 Experiments on Synthetic Data

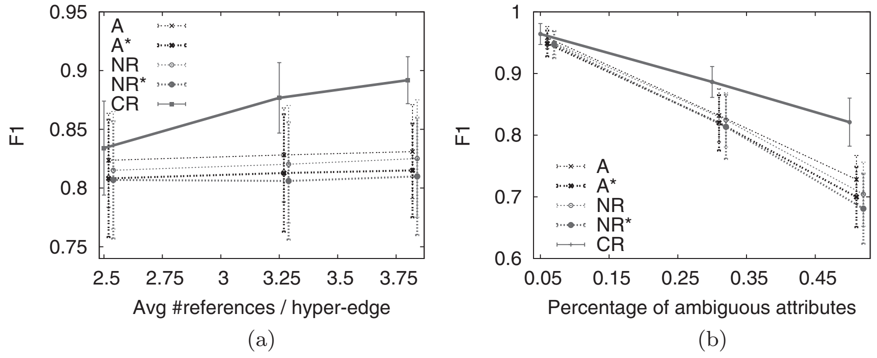
As we saw in the previous section, the benefit of using collective relational entity resolution varied across the different datasets. We attribute this performance difference to the differences in structural properties of the datasets, such as the fraction of references that are ambiguous, the number of neighbors per entity, etc. To better understand how these different structural characteristics affect the performance of our collective relational entity resolution, we also experiment with synthetically generated data where we can control the different structural characteristics. As we explain next, our synthetic data generator is not tailored solely to bibliographic data but can model general relationships between entities as in social network data or email data.

7.2.1 *Synthetic Data Generator.* We designed a two-stage data generator as described in Figure 10. Intuitively, we first construct a collaboration graph describing which entities are related, and then, using this graph, we generate a collection of observed relationships between entity references. In the first stage, the domain entities and their relationships are created. During the creation stage, we first create N entities and their attributes, and then add M binary relationships between them. For simplicity, rather than generating strings, we have one floating point attribute x for each entity and its references are later generated from a Gaussian distribution with mean x and variance

1.0. p_a controls the ambiguity of the generated attributes; with probability p_a, the entity attribute is chosen from values that are already in use by other entities. Values of x are chosen from a range that is large enough to accommodate attributes of all N entities at the specified ambiguity level. Once the entities are created, M binary relations (e_i, e_j) are added by choosing entities e_i and e_j randomly. In the next stage, we generate the publications and their coauthors. R hyperedges are generated, each with its own references. Each hyperedge $\langle r_i, r_{i1}, \ldots, r_{ik} \rangle$ is generated by first sampling an entity e and generating a reference r_i from it according to its Gaussian distribution. Each instance r_{ij} comes from randomly sampling (without replacement) a neighbor of e. Instances r_{i1} through r_{ik} are generated one-by-one with probability p_c of continuing further after each step.

7.2.2 *Evaluation.* We performed three sets of experiments on synthetically generated data. In all of our experiments, we consider average performance over 200 different runs. In our first experiment, we studied the effect of the number of references in each hyperedge. The objective of this experiment is twofold. Consider a collaborative graph where an entity e has many neighbors. If hyperedges are small in size, then two hyperedges involving references r_1 and r_2 corresponding to e may not have any other entities in common. Then it is not possible to identify whether r_1 and r_2 refer to the same entity e even using relational similarity. Secondly, in ambiguous domains, a single shared neighbor may not be enough to distinguish between two entities. In both cases, collective resolution is expected to benefit from larger hyperedge sizes. In each run for this experiment, we first constructed an entity graph by creating 100 entities and 200 binary relationships. Then we created different reference datasets, each with 500 hyperedges. We varied p_c, which led to different number of references in the edges. Figure 11(a) shows the performance of the different entity resolution algorithms on these datasets. Note that we have used standard deviation for error bars; standard error is too small for all three plots. We see that, while the performances of the attribute baselines (**A** and **A***) does not change, the performance of **CR** improves with increasing number of references per hyperedge. Interestingly, performance of the naive relational approach (**NR***) degrades with increasing number of references. This demonstrates the importance of resolving related names instead of considering only their attribute similarities.

In our second experiment, we varied the number of ambiguous references in the data. Collective resolution is particularly useful for ambiguous references. It may be possible to address the problem of identifying dispersed references for any entity by using a smaller similarity threshold with the attribute-only baseline. In contrast, disambiguation cannot be done using attributes but is often possible using relationships. So we expected our collective resolution approach to show larger improvements over the baseline for more ambiguous data. We created five sets of datasets, each with 100 entities, but with different ambiguous attribute probability p_a. Then we added 200 binary relations between these entities and generated 500 hyperedges with an average of 2 references per hyperedge. Figure 11(b) compares entity resolution performance for the different algorithms on the datasets. As expected, the performance of all algorithms



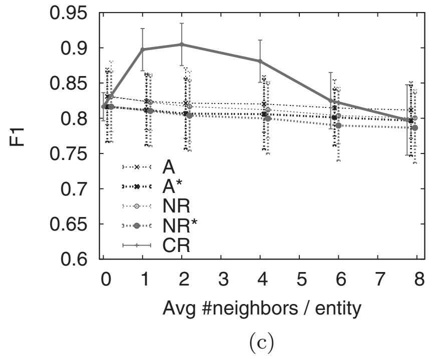


Fig. 11. Performance of different entity resolution algorithms on data synthetically generated by varying different structural parameters such as (a) the average size of hyperedges, (b) the percentage of ambiguous references, and (c) the average number of neighbors per entity. Standard deviations are shown using error bars.

drops with increasing percentage of ambiguous references. However, the performance drop for **CR** is significantly slower than those for the attribute and naive relational baselines since the entity relationships help to make the algorithm more robust. As a result, the gap between **CR** and the baselines increases as the percentage of ambiguous references in the data increases.

In our final experiment, we explored the impact of varying the number of relationships between the underlying entities. In the extreme situation, where there are no relationships between entities, clearly no improvement can be obtained using collective resolution. At the other extreme, when all entities are connected to each other, there is no pattern in the relationships that collective resolution can exploit. The objective of this experiment was to explore how increased connectivity among entities affects collective resolution. We first created a set of 100 entities. Then we created different entity graph structures by adding different number of relations between the entities. As before, we generated 500 hyperedges (with an average of 2 references per hyperedge) from each of these different entity graphs and compared performances of the different algorithms for the different datasets. The results are shown in Figure 11(c). First we note that, as expected, the performances of the attribute baselines (**A** and **A***) do not change significantly since they do not depend on the relationships.

The naive relational approaches (**NR** and **NR***) degrade in performance with higher neighborhood sizes, again highlighting the importance of resolving related references. The performance of **CR** increases initially as the number of relationships increases. However, it peaks when the average number of neighbors per entity is around 2, and then it starts falling off. In fact, it falls below the attribute baseline when the neighborhood size increases to 8. This is an interesting result that shows that increasing number of relationships does not always help collective entity resolution. As more relationships get added between entities, relationship patterns between entities are less informative and may actually hurt performance. In this experiment, the probability of ambiguous attributes p_a was 0.3. We observe the same trend for other values of p_a, the only change is the position of the peak. The peak occurs earlier as p_a is increased.

Finally, we discuss two of the current limitations of our approach. For the first limitation, recall that the similarity measure in Equation (2) involves a weighting parameter α for combining attribute and relational similarity. It is not clear how the optimal value for α should be chosen for each case and, for most of our comparisons, we consider the best F1 score over all values of α. Figure 7 shows the performance for a fixed value of α in contrast to a different optimal value for each case. It demonstrates that there are significant performance improvements using **CR** for any value of α over its entire range. Recall that the similarity measure uses only attributes when $\alpha = 0$ and only relations when $\alpha = 1$. For CiteSeer and arXiv, performance does not vary significantly with α. Since BioBase has much higher ambiguity in terms of attributes (many references have exactly the same name and therefore mostly the same countries, and all papers are from the same area), resolution performance improves with increasing α. Secondly, as with any clustering algorithm, determination of the termination threshold is an issue. Note that this comes up for all of the baselines as well, and here we report best accuracy over all thresholds. This is an area of ongoing research.

8. RELATED WORK

The entity resolution problem has been studied in many different areas under different names such as coreference resolution, deduplication, object uncertainty, record linkage, reference reconciliation, etc. Here we review some of the main work, but the review is not exhaustive. Winkler [1999] also provides a nice summary report.

The traditional approach to entity resolution looks at textual similarity in the descriptions of the entities. For example, determining whether two citations refer to the same paper depends on the similarity measure such as edit distance between the two citation strings. There has been extensive work on defining approximate string similarity measures [Monge and Elkan 1996; Navarro 2001; Cohen et al. 2003; Chaudhuri et al. 2003] that may be used for unsupervised entity resolution. Another approach is to use adaptive supervised algorithms that learn string similarity measures from labeled data [Ristad and Yianilos 1998; Bilenko and Mooney 2003; Cohen and Richman 2002; Tejada et al. 2001]. One of the difficulties in using a supervised method for resolution is constructing a good training set that includes a representative collection of positive and negative examples. Other approaches use active learning [Sarawagi and Bhamidipaty 2002; Tejada et al. 2001] where the user is asked to label ambiguous examples by the learner.

Even the attribute-only approach to entity resolution is known to be a hard problem computationally. Therefore, efficiency issues have long been a focus for data cleaning, the goal of which is the development of inexpensive algorithms for finding approximate solutions. The key mechanisms for doing this involve computing the matches efficiently and employing techniques commonly called blocking to quickly find potential duplicates [Hernández and Stolfo 1995; Monge and Elkan 1997; McCallum et al. 2000]. Gravano et al. [2003] propose a sampling approach to quickly compute cosine similarity between tuples for fast text-joins within an SQL framework. Chaudhuri et al. [2003] use an error tolerant index for data warehousing applications to efficiently look up a small but probabilistically safe set of reference tuples as candidates for matching for an incoming tuple. Swoosh [Benjelloun et al. 2005] has recently been proposed as a generic entity resolution framework that considers resolving and merging duplicates as a database operator and the goal is to minimize the number of record-level and feature-level operations.

The groundwork for posing entity resolution as a probabilistic classification problem was done by Fellegi and Sunter [1969], who extend the ideas of Newcombe et al. [1959] for labeling pairs of records from two different files to be merged as match or nonmatch on the basis of agreement among their different fields. Winkler [2002] and, more recently, Ravikumar and Cohen [2004] have built upon this work. Probabilistic models that take into account interaction between different entity resolution decisions have been proposed for named-entity recognition in natural language processing and for citation matching. McCallum and Wellner [2004] use conditional random fields for noun coreference and use clique templates with tied parameters to capture repeated relational structure. Singla and Domingos [2004] use the idea of merging evidence to allow the flow of reasoning between different pairwise decisions over multiple entity types. These two relational models are supervised and require labeled data to train the parameters. Availability of sufficient labeled data is often an issue for this problem and unsupervised relational models have also been developed. Li et al. [2005] address the problem of disambiguating entity mentions, potentially of multiple types, in the context of unstructured textual documents. They propose a probabilistic generative model that captures a joint distribution over pairs of entities in terms of comentions in documents. Pasula et al. [2003] propose a generic probabilistic relational model framework for the citation matching problem. In other work of our own [Bhattacharya and Getoor 2006b], we have extended the Latent Dirichlet Allocation model for documents and topics and extended it to propose a generative group model for collective entity resolution. Instead of performing a pairwise comparison task, we use a latent group variable for each reference, which is inferred from observed collaborative patterns among references, in addition to attribute similarity to predict the entity label for each reference. All of these probabilistic models have been shown to perform well in practice and have the advantage that the match/nonmatch decisions do not depend on any user-specified threshold but are learned directly from data. However, this benefit comes at a price. Inference in relational probabilistic models is an expensive process. Exact inference is mostly intractable and approximate strategies such as loopy belief propagation and Monte Carlo sampling strategies are employed. Even these approximate strategies take several iterations to converge and extending such approaches to large datasets is still an open problem.

Alternative approaches [Ananthakrishna et al. 2002; Bhattacharya and Getoor 2004; Kalashnikov et al. 2005; Dong et al. 2005] consider the relational structure of the entities for data integration but avoid the complexity of probabilistic inference. By avoiding a formal probabilistic model, these approaches can handle complex relationships between different entities more easily, and the resolution process is significantly faster as well. Kalashnikov et al. [2005] enhance feature-based similarity between an ambiguous reference and the many entity choices for it with relationship analysis between the entities such as affiliation and coauthorship. They propose a content-attraction principle hypothesizing that an ambiguous reference will be more strongly connected via such relationships to its true entity compared to other entity choices for it. They translate this principle to a set of nonlinear equations involving connection strengths in the entity graph which are solved to determine the entity choice for each reference. They show improvements by the use of relationship chains connecting multiple types of entities in contrast to just first-order relationships. This approach is useful for incremental data cleaning when the set of entities currently in the database is known and an incoming reference needs to be matched with one of these entities. In the more general setting that we consider, the entities are not known and need to be discovered. Ananthakrishna et al. [2002] introduce relational deduplication in data warehouse applications where there is a dimensional hierarchy over the relations. Using an approach similar in spirit to our naive relational baseline, they augment the string similarity measure between two tuples with the similarity between their foreign key relations across the hierarchy which they call children sets. In the specific case, where the relationships represent an ordered set as in a domain hierarchy, they show how the similarity computation can be made more efficient. To avoid comparison between all pairs of tuples in a relation, they propose a grouping strategy that makes use of the relational hierarchy. In earlier work of our own [Bhattacharya and Getoor 2004], we have done a preliminary exploration of clustering approaches for collective entity resolution in the presence of relations. The approach that is the most similar in spirit to ours is that of Dong et al. [2005]. They collectively resolve entites of multiple types by propagating relational evidence in a dependency graph and demonstrate the benefits of collective resolution in real datasets. Their approach creates a binary decision node for each potential pair of duplicates which can be expensive in large datasets. One key strategy that they employ to propagate evidence is merging attribute decision nodes. Specifically, for a pair of names such as J. Smith and John Smith, they create a single decision node in the dependency graph and multiple entity pairs can share this decision. However, while John Smith and

J. Smith may refer to the same individual for a particular mention of J Smith, it is quite possible for another mention of J. Smith to refer to James Smith. This approach is useful for identifying many dispersed references for the same entity but not for domains where disambiguation is important.

9. CONCLUSION

Entity resolution is an area that has been attracting growing attention in order to address the influx of structured and semistructured data from a multitude of heterogeneous sources. Accurate resolution is important for a variety of reasons, ranging from cost effectiveness and reduction in data volume to accurate analysis for critical applications. In the case of structured data, it is especially important to look at entity resolution from a relational perspective. Collective relational entity resolution is a powerful and promising approach that combines attribute similarity with relational evidence and shows improved performance over traditional approaches.

In this article, we have shown how entity resolution may be posed as a relational clustering problem where the entity labels of related references depend on each other. We described an algorithm for this problem that augments a general class of attribute similarity measures with relational similarity among the entities to be resolved. We investigated the effectiveness of these relational similarity measures on three real bibliographic datasets with different characteristics and levels of ambiguity. In all datasets, our similarity measures significantly outperformed the baseline algorithms, but the degree of improvement depended on the intrinsic ambiguity of the dataset—the greater the ambiguity, the greater the benefit obtained using collective resolution. To study the dependence of the algorithm's performance on different structural characteristics of the domain, we performed detailed experiments on synthetically generated data where we can vary the data characteristics in a controlled fashion. We showed that the benefits of relational clustering over attribute-based approaches is greatest when there are many references per hyperedge and when there are many ambiguous references in the data. Interestingly, we also showed that, as the number of relations between underlying entities increases, the performance of collective resolution peaks for a particular neighborhood size. But then the gains diminish as relational patterns become less informative.

There are many interesting avenues for future work. While our algorithms are general and can handle multiple entity types, our experimental results have focused on a single entity and relation type. Also all our real-world datasets are bibliographic, it would be interesting to study our algorithms on different types of relational data including consumer data, social network data, and biological data. Our work starts from data in which the references have already been extracted; it would be interesting to integrate the collective resolution process with the extraction process. Finally, in this article, we have viewed entity resolution as an offline data cleaning process; an alternative view, which we have recently begun to investigate [Bhattacharya and Getoor 2006c], is the notion of query-time entity resolution in which only the data relevant to a query is resolved.

ACKNOWLEDGMENTS

We wish to thank our anonymous reviewers for their many constructive suggestions for improving the article.

REFERENCES

ADAMIC, L. AND ADAR, E. 2003. Friends and neighbors on the Web. *Social Networ. 25*, 3 (July), 211–230.

ANANTHAKRISHNA, R., CHAUDHURI, S., AND GANTI, V. 2002. Eliminating fuzzy duplicates in data warehouses. In *The International Conference on Very Large Databases (VLDB)*. Hong Kong, China.

BENJELLOUN, O., GARCIA-MOLINA, H., SU, Q., AND WIDOM, J. 2005. Swoosh: A generic approach to entity resolution. *Tech. rep.*, Stanford University. (March)

BHATTACHARYA, I. AND GETOOR, L. 2004. Iterative record linkage for cleaning and integration. In *The ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD)*. Paris, France.

BHATTACHARYA, I. AND GETOOR, L. 2006a. Mining graph data. In *Entity Resolution in Graphs*. L. Holder and D. Cook, Eds. John Wiley.

BHATTACHARYA, I. AND GETOOR, L. 2006b. A latent dirichlet model for unsupervised entity resolution. In *The SIAM Conference on Data Mining (SIAM-SDM)*. Bethesda, MD.

BHATTACHARYA, I. AND GETOOR, L. 2006c. Query-time entity resolution. In *The ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*. Philadelphia, PA.

BILENKO, M. AND MOONEY, R. 2003. Adaptive duplicate detection using learnable string similarity measures. In *The ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*. Washington, DC.

BILENKO, M., MOONEY, R., COHEN, W., RAVIKUMAR, P., AND FIENBERG, S. 2003. Adaptive name matching in information integration. *IEEE Intellig. Syst. 18*, 5, 16–23.

CHAUDHURI, S., GANJAM, K., GANTI, V., AND MOTWANI, R. 2003. Robust and efficient fuzzy match for online data cleaning. In *The ACM International Conference on Management of Data (SIGMOD)*. San Diego, CA.

COHEN, W. 2000. Data integration using similarity joins and a word-based information representation language. *ACM Trans. Inform. Syst. 18*, 288–321.

COHEN, W., RAVIKUMAR, P., AND FIENBERG, S. 2003. A comparison of string distance metrics for name-matching tasks. In *The IJCAI Workshop on Information Integration on the Web (IIWeb)*. Acapulco, Mexico.

COHEN, W. AND RICHMAN, J. 2002. Learning to match and cluster large high-dimensional data sets for data integration. In *The ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*. Edmonton, Canada.

DONG, X., HALEVY, A., AND MADHAVAN, J. 2005. Reference reconciliation in complex information spaces. In *The ACM International Conference on Management of Data (SIGMOD)*. Baltimore, MD.

FELLEGI, I. AND SUNTER, A. 1969. A theory for record linkage. *J. Amer. Statis. Assoc. 64*, 1183–1210.

GILES, C. L., BOLLACKER, K., AND LAWRENCE, S. 1998. CiteSeer: An automatic citation indexing system. In *The ACM Conference on Digital Libraries*. Pittsburgh, PA.

GRAVANO, L., IPEIROTIS, P., KOUDAS, N., AND SRIVASTAVA, D. 2003. Text joins for data cleansing and integration in an RDBMS. In *The IEEE International Conference on Data Engineering (ICDE)*. Bangalore, India.

HERNÁNDEZ, M. AND STOLFO, S. 1995. The merge/purge problem for large databases. In *The ACM International Conference on Management of Data (SIGMOD)*. San Jose, CA.

KALASHNIKOV, D., MEHROTRA, S., AND CHEN, Z. 2005. Exploiting relationships for domain-independent data cleaning. In *The SIAM International Conference on Data Mining (SIAM SDM)*. Newport Beach, CA.

LI, X., MORIE, P., AND ROTH, D. 2005. Semantic integration in text: From ambiguous names to identifiable entities. *AI Magazine*. Special Issue on Semantic Integration *26*, 1, 45–58.

LIBEN-NOWELL, D. AND KLEINBERG, J. 2003. The link prediction problem for social networks. In *The International Conference on Information and Knowledge Management (CIKM)*. New Orleans, LA.

McCallum, A., Nigam, K., and Ungar, L. 2000. Efficient clustering of high-dimensional data sets with application to reference matching. In *The International Conference On Knowledge Discovery and Data Mining (SIGKDD)*. Boston, MA.

McCallum, A. and Wellner, B. 2004. Conditional models of identity uncertainty with application to noun coreference. In *The Annual Conference on Neural Information Processing Systems (NIPS)*. Vancouver, Canada.

Monge, A. and Elkan, C. 1996. The field matching problem: Algorithms and applications. In *The International Conference on Knowledge Discovery and Data Mining (SIGKDD)*. Portland, ME.

Monge, A. and Elkan, C. 1997. An efficient domain-independent algorithm for detecting approximately duplicate database records. In *The SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD)*. Tuscon, AZ.

Navarro, G. 2001. A guided tour to approximate string matching. *ACM Comp. Sur. 33*, 1, 31–88.

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

Pasula, H., Marthi, B., Milch, B., Russell, S., and Shpitser, I. 2003. Identity uncertainty and citation matching. In *The Annual Conference on Neural Information Processing Systems (NIPS)*. Vancouver, Canada.

Ravikumar, P. and Cohen, W. 2004. A hierarchical graphical model for record linkage. In *The Conference on Uncertainty in Artificial Intelligence (UAI)*. Banff, Canada.

Ristad, E. and Yianilos, P. 1998. Learning string edit distance. *IEEE Trans. Patt. Anal. Mach. Intell. 20*, 5, 522–532.

Sarawagi, S. and Bhamidipaty, A. 2002. Interactive deduplication using active learning. In *The ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*. Edmonton, Canada.

Singla, P. and Domingos, P. 2004. Multi-relational record linkage. In *The ACM SIGKDD Workshop on Multi-Relational Data Mining (MRDM)*. Seattle, WA.

Tejada, S., Knoblock, C., and Minton, S. 2001. Learning object identification rules for information integration. *Inform. Syst. J. 26*, 8, 635–656.

Winkler, W. 1999. The state of record linkage and current research problems. Tech. rep., Statistical Research Division, U.S. Census Bureau, Washington, DC.

Winkler, W. 2002. Methods for record linkage and Bayesian networks. Tech. rep., Statistical Research Division, U.S. Census Bureau, Washington, DC.

Received July 2006; revised November 2006; accepted January 2007