A Framework for Entity Resolution with Efficient Blocking

Liangcai Shu Can Lin Weiyi Meng Yue Han
Department of Computer Science
State University of New York at Binghamton
Binghamton, NY 13902
lshu@cs.binghamton.edu clin27@binghamton.edu meng@cs.binghamton.edu yhan1@binghamton.edu

Clement T. Yu
Department of Computer Science
University of Illinois at Chicago
Chicago, IL 60607
yu@cs.uic.edu

Neil R. Smalheiser
Department of Psychiatry, UIC Psychiatric Institute MC912
University of Illinois at Chicago
Chicago, IL 60612
nsmalheiser@psych.uic.edu

Abstract

In applications of Web data integration, we frequently need to identify whether data objects in different data sources represent the same entity in the real world. This problem is known as entity resolution. In this paper, we propose a generic framework for entity resolution for relational data sets, called BARM, consisting of the Blocker, Attribute matchers and the Record Matcher. BARM is convenient for different blocking and matching algorithms to fit into it. For the blocker, we apply the SPectrAl Neighborhood (SPAN), a state-of-the-art blocking algorithm, to our data sets and show that SPAN is effective and efficient. For attribute matchers, we propose the Context Sensitive Value Matching Library (CSVML) for matching attribute values and also an approach to evaluate the goodness of matching functions. CSVML takes the meaning and context of attribute values into consideration and therefore has good performance, as shown in experimental results. We adopt Bayesian network as the record matcher in the framework and propose a method of inference from Bayesian network based on Markov blanket of the network. As a comparison, we also apply three other classifiers, including Decision Tree, Support Vector Machines, and the Naive Bayes classifier to our data sets. Experiments show that Bayesian network is advantageous in the book domain.

1. Introduction

In many applications of Web data integration, there is a need to identify whether data objects in different data sources represent the same entity in the real-world. This problem is known as *Entity Resolution* (ER), which is also known as record matching [22][11], record linkage [20][12][3] , deduplication [5][2] or data cleaning [6]. The problem arises quite often in information integration where data objects representing the same real-world entity are presented in different ways. In some cases, an object is represented by a single attribute value. For example, “Alcatel-Lucent”, “Alcatel Lucent” and “Lucent” may appear on different web pages but they all represent the same company. In more complicated cases, an object can be a record consisting of values of multiple attributes.

For example, here are two citations that probably refer to the same paper:

- `[Lashkari et al 94] Collaborative Interface Agents, Proceedings of the Twelfth National Conference on Artificial Intelligence, MIT Press, Cambridge, MA, 1994.`
- `Yezdi Lashkari, Max Metral, and Pattie Maes. Collaborative interface agents. In Conference of the American Association for Artificial Intelligence, Seattle, WA, August 1994.`

In this example, a record representing a paper or an entity, includes seven attributes — Author, Title, Conference/Venue, Publisher, Publisher Address, Conference Address and Publication Date. This poses a great challenge for resolving entities, i.e., identifying data objects that represent the same real-world entity.

There have been many techniques proposed for solving the ER problem (see [9] for a survey). In a typical ER process, the first step is *blocking*. The blocking algorithm divides records into blocks. Records from different blocks are considered not likely to match. Only pairs of records that are within the same block are considered in pairwise comparison. Blocking is used to improve efficiency. The second step is *attribute matching*, i.e., to compare the corresponding attribute values from two records within the same block. Multiple metrics can be used, including edit distance, Jaro distance, Cosine similarity, etc., to measure how well two attribute values match. The third step is *record matching*, i.e., to use an ER algorithm to combine the results of attribute matching. The ER algorithm can be either an unsupervised or a supervised learner.

In this paper, we propose BARM, a novel generic framework for entity resolution for relational data sets. This framework consists of a Blocker, a set of Attribute matchers and a Record Matcher (BARM). A feature of this framework is that the three components are loosely coupled, communicating with each other only through a few matrices — blocking matrix, matching matrix, etc. Therefore the most advanced approaches for blocking and matching can be conveniently integrated into this framework.

Our contributions include the following:

- We propose a generic framework for entity resolution for relational data sets (BARM), which consists of a blocker, a set of attribute matchers, and a record matcher. Since the components are loosely coupled, it is easy for different advanced blocking and attribute/record matching algorithms to fit into the framework.
- We present a precise definition of *blocker* or *blocking algorithm* and apply the state-of-the-art blocking algorithm SPAN [27] to a specific domain. We also propose a sparse blocking matrix to store the blocking results. This makes the blocking results reusable and easy to be combined into a *composite blocker*. This also enables iterative blocking possible, i.e., iteratively updating *blocking matrix* by feedback from entity resolution results in *resolution matrix*. Based on the updated *blocking matrices*, additional pairwise comparisons are performed to further reduce false negatives of entity resolution.
- We propose the Context Sensitive Value Matching Library (CSVML), for matching attributes. CSVML has several value matchers that take into account the context and meaning of input values when performing value matching. This has led to improved performance. And we also propose an approach to evaluate the *goodness* of matching functions.
- We propose a Bayesian network inference approach by means of Markov blanket for the entity resolution problem. We also perform experiments to compare the Bayesian network approach with three other classifiers, including Decision Tree, Support Vector Machines, and the Naive Bayes classifier.

The rest of the paper is organized as follows. Section 2 is related work. Section 3 introduces our framework for entity resolution. Section 4 presents CSVML. Section 5 presents inference of Bayesian network. Section 6 presents our experimental results in detail. Section 7 concludes the paper.

2. Related Work

There have been many methods and tools developed for entity resolution (see [32][1][9] for surveys). A variety of *learning-based* methods have been proposed for entity resolution, which can be categorized as *supervised* and *unsupervised* learning. For example, naive Bayes [26], logistic regression [24], support vector machines [5] and decision trees [14]) are *supervised* learning approaches, while co-training on clustering [30], probabilistic model [12], topic model [28][4], and clustering algorithms [15] are *unsupervised* learning approaches. Our approach is a combination of *unsupervised* (blocking) and *supervised* (record matching) learning algorithms.

[27] proposes spectral neighborhood Blocking (SPAN), a novel and efficient *blocking* algorithm based on spectral clustering. Blocking has been studied extensively in the literature , and various blocking techniques have been proposed, including sorted neighborhood (SN) [16], bigram indexing (BI) [7][8], and canopy clustering (CC) [19]. However, SPAN is the first one that is derived from spectral clustering for large-scale entity resolution problems, and it improves prior approaches on capturing both *intra*- and *inter*-block similarities efficiently, i.e., in the time of $O(n \log n)$.

Iterative blocking has been proposed in [10], which combines multiple single-attribute blocking results to reduce false negatives (i.e., improve recall).

There have been various similarity measures available e.g., edit distance [17], token-based n-gram [13], and character-based $q$gram [29].

3. BARM: A Framework for Entity Resolution

3.1. Overview

Entity resolution aims to solve the entity ambiguity in a domain. In relational data sets, a record is referred to as an entity. We first give definitions in order to better describe this problem. Then we present our BARM framework for ER (see Fig. 1) for relational data sets.

The blocker takes input from two tables to be compared, and builds a sparse matrix — the *blocking matrix*. The *attribute matchers* compare the two tables, subject to the blocking matrix, and generates the *matching matrix* (each attribute matcher corresponds to one column of the matrix)

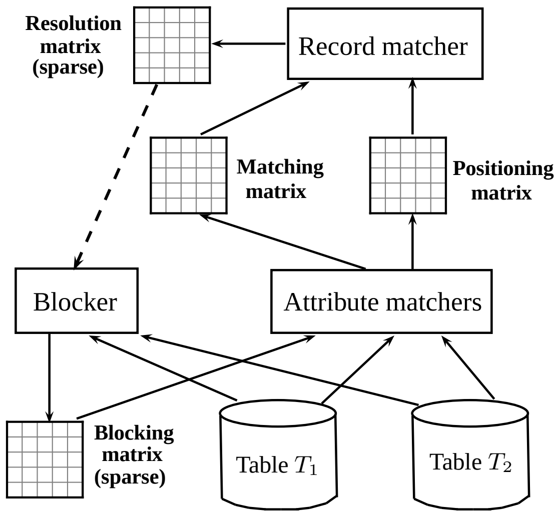

Figure 1. BARM: a framework for entity resolution (ER) for relational data sets.

and its *positioning matrix*. The *record matcher* makes entity resolution based on the matching matrix and positioning matrix and generates a sparse matrix — the *resolution matrix*.

Definition 1 *For a domain, the* universal attribute set *consists of all attributes from all data sources in this domain, denoted by* $\mathcal{A} = \{a_i\}_{1 \le i \le N}$*, where* i *is an integer and* $N = |\mathcal{A}|$ *is the number of attributes in this domain.*

Definition 2 *In a domain, a* table *is defined as a relational database table from a data source, denoted by* $T = (A, V)$*, where* A*, a subset of* $\mathcal{A}$*, is the* attribute set *of* T*, and* V *is a* value set *on* A*, also called the* record set *of* T*.*

Table 1. Table T_1 in domain *books*

Title	*Authors*	*Publisher*
Databases	Clement Yu	Prentice
Metasearch Engines	Weiyi Meng	Elsevier

Table 2. Table T_2 in domain *books*

Title	*Authors*	*Year*
Databases	Clement Yu	2001
Databases	Weiyi Meng	2001
Harry Potter	J. K. Rowling	2001
Metasearch Engines	W. Meng	2003

We use an example to illustrate the problem.

Example 3.1: Assume we have two *tables* $T_1 = (A_1, V_1)$ and $T_2 = (A_2, V_2)$ as in Tables 1 and 2 in domain *books*. Then we have the *universal attribute set* $\mathcal{A} = \{Title, Authors, Publisher, Year\}$ and attribute sets $A_1 = \{Title, Authors, Publisher\}$ and $A_2 = \{Title, Authors, Year\}$. The value set V_1 is in Table 1 and the value set V_2 is in Table 2.

3.2. Blockers

Entity resolution involves pairwise comparisons of records from two tables which may be inefficient as sizes of data sets increase. In this section, we will define blockers and propose a way to combine simple blockers into composite blockers.

Blockers and blocking matrices A blocking algorithm can be considered as a black box, its input data sets are two *tables*, and its output can be a set of pairs of records that are considered potentially matching and need pairwise comparisons. We use a sparse matrix to represent the set of pairs of objects. The blocker is defined as follows.

Definition 3 *Given any two tables in the same domain* $T_1 = (A_1, V_1)$ *and* $T_2 = (A_2, V_2)$*, where* V_i *(*$i = 1, 2$*) is the value set of* T_i*, a* blocker *or a* blocking algorithm*, denoted by* B*, is defined as a mapping* $b : (\boldsymbol{v}_1, \boldsymbol{v}_2) \mapsto s$*, where* $(\boldsymbol{v}_1, \boldsymbol{v}_2) \in V_1 \times V_2$*,* $s \in \{1, 0\}$*, and* $\times$ *is the Cartesian product. Here* 1 *means two records is likely to match and in the same block, and* 0 *means they are not likely to match and not in the same block. The blocking results are conveniently stored in a* $|V_1| \times |V_2|$ *sparse matrix, which includes entries of value 1 and* 0*. This matrix is called the* blocking matrix*, denoted by* $\mathbf{B}(V_1, V_2)$*.*

For our proposed framework, we use the spectral neighborhood (SPAN) blocking algorithm [27] we recently developed as the blocker.

The *blocking matrix* is a perfect structure to store blocking results. Its rows correspond to records of $T_1 = (A_1, V_1)$, and its columns correspond to records of $T_2 = (A_2, V_2)$. Since it is sparse, the matrix is space efficient. And it also supports efficient retrieval and enumeration of blocking results, which is convenient for attribute matchers to use.

Simple and composite blockers

Definition 4 *If a blocker is based on one attribute* a*, it is called a* simple blocker*, denoted by* $SB(a)$ *and its blocking matrix is denoted by* $\mathbf{B}(a)$*. If a blocker is based on multiple attributes* $\{a_1, \ldots, a_n\}$*, it is called a* composite blocker*, denoted by* $CB(a_1, \ldots, a_n)$ *and its blocking matrix is denoted by* $\mathbf{B}(a_1, \ldots, a_n)$*.*

To build a composite blocker $CB(a_1, \ldots, a_n)$, we need to build n simple blocker $SB(a_i)(i = 1, \ldots, n)$. Then the blocking matrix of composite blocker $CB(a_1, \ldots, a_n)$ is

$$\mathbf{B}(a_1, \ldots, a_n) = \mathbf{B}(a_1) \vee \mathbf{B}(a_2) \vee \cdots \vee \mathbf{B}(a_n)), \quad (1)$$

where operator $\vee$ is logical OR. We define the logical OR operation of two matrices with the same dimension as the logical OR operation of corresponding elements. In this paper for our running example, the blocker is a composite blocker based on two attributes — *Title* and *Authors*.

3.3. Feedback blocking

In this paper, we propose a novel method, namely *feedback blocking*, which enables the blocking results to be updated from feedback from entity resolution (ER) results, and this update can be used to further improve quality of entity resolution results. That can happen iteratively until no more new matches can be found in entity resolution results.

In our framework BARM, blocking results are stored in the *blocking matrix* and entity resolution results are stored in the *resolution matrix* that we define below.

Definition 5 *Given two tables $T_1 = (A_1, V_1)$ and $T_2 = (A_2, V_2)$, the entity resolution results are stored in a $|V_1| \times |V_2|$ sparse matrix, containing only entries of value 1 and 0, representing* match *and* not match, *respectively. This matrix is called the* resolution matrix, *denoted by $\mathbf{R}$.*

Like the *blocking matrix*, the *resolution matrix* is sparse and thus is space efficient, and has good retrieval performance. The two matrices have the same dimension $|V_1| \times |V_2|$.

Assume that $\mathbf{B}_n$ is the blocking matrix and $\mathbf{R}_n$ is the resolution matrix at current stage. $\mathbf{R}_n$ represents the similarity between records in V_1 (as rows) and records in V_2 (as columns). Then $\mathbf{R}_n\mathbf{R}_n^T$ represents self-similarity within V_1 and $\mathbf{R}_n^T\mathbf{R}_n$ represents self-similarity within V_2. We can incorporate this transitivity property into our next-stage blocking matrix.

Therefore considering update from $\mathbf{R}_n$, the blocking matrix in next stage is

$$\mathbf{B}_{n+1} = \mathbf{B}_n\mathbf{R}_n^T\mathbf{R}_n \vee \mathbf{R}_n\mathbf{R}_n^T\mathbf{B}_n \tag{2}$$

Here is the physical meaning of this formula.

Assume that records $\mathbf{v}_1 \in V_1$ and $\mathbf{v}_2 \in V_2$ are determined by the ER algorithm (record matcher in this paper) as *match*. Then the elements sharing the same block with $\mathbf{v}_1$ in V_1 and the elements sharing the same block with $\mathbf{v}_2$ in V_2 will be connected in $\mathbf{B}_{n+1}$. The new connections will be submitted to the ER algorithm for pairwise comparison.

Since B_n have been submitted to ER before, we need to submit the difference matrix $\Delta\mathbf{B}_{n+1} = \mathbf{B}_{n+1} - \mathbf{B}_n$ for the new stage pairwise comparison. The positives of the ER results for the new stage pairwise comparisons will be added to $\mathbf{R}_n$ and generate $\mathbf{R}_{n+1}$.

This can be done iteratively until $\mathbf{R}$ converges (i.e., $\mathbf{R}_{n+1} = \mathbf{R}_n$).

3.4. Spectral neighborhood blocking

In this paper for blocking of entity resolution, we use spectral neighborhood blocking algorithm (SPAN) [27], which we recently developed. In books domain, we apply SPAN twice, one on attribute *Title* and the other on attribute *Authors*.

In this section, we give a brief introduction of SPAN. This method is based on the vector space model [25], where we represent each record by a vector of *qgrams*. A qgram (or N-gram [18]) is a length q substring of the blocking attribute value. For example, if an attribute value of *Title* in books domain is “DATABASES” and $q = 3$, the corresponding qgrams are “##D”, “#DA”, “DAT”, “ATA”, “TAB”, “ABA”, “BAS”, “ASE”, “SES”, “ES$” and “S$$”, where ‘#’ and ‘$’ are the beginning and ending padding characters [9], respectively.

The SPAN algorithm has two major steps: first, it constructs a binary tree using a *fast-bipartition* procedure recursively where the Newman-Girvan modularity [21] is used as the stopping criterion, such that its leaf nodes give a non-overlapping partition of the n records; second, it performs neighborhood search on the tree to identify candidate record pairs as input of an entity resolution algorithm.

For more details, please refer to the original SPAN paper [27].

3.5. Attribute matchers

An attribute matcher with a blocker can be defined as follows:

Definition 6 *Given two tables $T_1 = (A_1, V_1)$ and $T_2 = (A_2, V_2)$ and a blocker B with blocking matrix $\mathbf{B}$, an* attribute matcher *for V_1 and V_2 with blocking matrix $\mathbf{B}$, denoted by $AM(V_1, V_2, \mathbf{B})$, is defined as a mapping $h : (\boldsymbol{v}_1, \boldsymbol{v}_2) \mapsto (s_1, \ldots, s_{|\mathcal{A}|})$, where $(\boldsymbol{v}_1, \boldsymbol{v}_2) \in V_1 \times V_2$, $\mathbf{B}(\boldsymbol{v}_1, \boldsymbol{v}_2) = 1$, and $(s_1 \ldots s_{|\mathcal{A}|}) \in S_1 \times S_2 \times \ldots \times S_{|\mathcal{A}|}$, and S_i ($1 \le i \le |\mathcal{A}|$) is the set of attribute matching values, which measure how well attribute values match for the ith attribute in the universal attribute set $\mathcal{A}$.*

Here $\mathbf{v}_1$ and $\mathbf{v}_2$ are actually two records. Attribute sets A_1 and A_2 might be different subsets of $\mathcal{A}$.

Attribute matching for two tables will generate a set of N-dimensional vectors, whose components correspond to $N = |\mathcal{A}|$ attributes for this domain. Records $\mathbf{v}_1$ and $\mathbf{v}_2$ are also vectors, and we have $\dim(\mathbf{v}_1) \le N$ and $\dim(\mathbf{v}_2) \le N$, where $\dim(\cdot)$ is the dimensionality of a vector. $\dim(\mathbf{v}_1) < N$ when values of some attributes in $\mathcal{A}$ are missing in $\mathbf{v}_1$.

Definition 7 *The* matching matrix, *denoted by $\mathbf{M}$, is a matrix that stores the matching results of attribute matcher $AM(V_1, V_2, \mathbf{B})$. The number of columns of $\mathbf{M}$ is $|\mathcal{A}|$, and* *the number of rows is the number of nonzero entries in the blocking matrix* $\mathbf{B}$.

Each row in the matching matrix $\mathbf{M}$ corresponds two records being compared, one from T_1 and the other from T_2. To track positions of the two records in T_1 and T_2, we use the *positioning matrix*.

Definition 8 *The* positioning matrix, *denoted by* $\mathbf{P}$, *is a matrix that stores the positions (or indices) of two records for the matching matrix* $\mathbf{M}$. *This matrix has the same number of rows as in* $\mathbf{M}$, *each row corresponding to one row in matching matrix* $\mathbf{M}$. *And it has two columns, one for positions in* T_1 *and the other for those in* T_2.

The positioning matrix is useful when generating the final matching results.

Attribute matching process The *attribute matcher* iterates and looks up the non-zero entries of the *blocking matrix*, and accesses a pair of records from two tables according to the positions of the nonzero entry. The matching results are saved into the *matching matrix* and positions are saved into the *positioning matrix*.

3.6. Record matchers

In this paper, the record matcher is an approach to map any two records to a boolean value (1 or 0) representing *match* or *not match*, based on the results of the attribute matcher.

Definition 9 *Given two tables* $T_1 = (A_1, V_1)$ *and* $T_2 = (A_2, V_2)$ *in the same domain, a* record matcher *is a mapping defined as* $f : (\mathbf{v}_1, \mathbf{v}_2) \mapsto r$, *where* $(\mathbf{v}_1, \mathbf{v}_2) \in V_1 \times V_2$, $r \in \{0, 1\}$, *and* $\times$ *is the Cartesian product. The matching results are stored in a* $|V_1| \times |V_2|$ *sparse matrix, i.e., the* resolution matrix $\mathbf{R}$.

After attribute matching is completed, a method is needed to map the attribute matching results to record matching results.

Assume attribute value matching has been finished for two given tables T_1 and T_2. In other words, we have known the attribute matching function $h(\mathbf{v}_1, \mathbf{v}_2) = (s_1, \ldots, s_N)$. Then the problem of finding the record matching function $f(\mathbf{v}_1, \mathbf{v}_2) = r$ boils down to that of finding a mapping $g : (s_1, \ldots, s_N) \mapsto r$. Formally, $f(\mathbf{v}_1, \mathbf{v}_2) = g(h(\mathbf{v}_1, \mathbf{v}_2))$.

In this paper, we aim to use supervised learning, specifically Bayesian network, to solve this mapping problem. As an example, the input data would be similar as those in Table 3, where columns m_t, m_a, m_p and m_y are matching results for *Title*, *Authors*, *Publisher* and *Year*, respectively. And the column record matching result m_r is the class label.

Table 3. Matching of Two Tables T_1 and T_2.

m_t	m_a	m_p	m_y	m_r
1	1	1/2	1/2	1
1	0	1/2	1/2	0
0	0	1/2	1/2	0
0	0	1/2	1/2	0
0	1	1/2	1/2	0
1	1/2	1/2	1/2	1

4. CSVML for Attribute Matching

We have developed some metrics for matching two attribute values from two records. We use matching functions in CSVML (Context Sensitive Value Matching Library). CSVML is a software package in Java we develop to match attributes for information integration. This package implements several semantic string matching functions, including number, date, person name, and journal/book title.

Several third party packages are utilized in the development of CSVML. SimMetrics[1] is a library including a number of string matching functions, such as edit/Levenshtein distance functions. We do not directly use SimMetrics' distance functions in CSVML, although we implement the same API interfaces as SimMetrics. NameParse[2] is a Perl implementation of person name parser. It accepts person name in free text format and parses it into different components, such as first name, last name, etc. We use this name parser to parse a name into components in the preprocessing of matching two person names. We also use a nickname table [3] compiled by Deron Meranda, which is used to calculate the similarity of first names.

We investigate the attribute matching problem in the book domain. In this domain, we have attributes *title* (T), *author* (A), *publisher* (P), *publication time* (PT), *price* (PR), *binding* (B), *edition* (E), *detail url* (DU), and *description*. In this paper we mainly use four types of metrics in CSVML: Person Name Distance for attribute *author*, Book Name Distance for attribute *title*, Date Distance for *publication time*, Number Distance for *price*.

4.1. Matching functions

Person name similarity We now discuss our method to compute person name distance for two person names. We first decompose names into a series of name components: first names (or first initials), middle names (or middle initials), last names, nicknames, name titles (e.g., Dr., Mr., Ms, etc). Then we compare corresponding name compo-

[1]http://sourceforge.net/projects/simmetrics/

[2]http://search.cpan.org/dist/Lingua-EN-NameParse/lib/Lingua/EN/NameParse.pm

[3]http://deron.meranda.us/data/nicknames.txt

nents and get component similarities. Finally we calculate the overall similarity from components similarities for the two person names. The overall similarity is normalized into $[0, 1]$.

We use a set of rules to calculate the similarity (denoted by s) of a pair of name components. For first names, if both first names are full first names and they are the same, $s = 1$; if one is the variant of the other according to the nickname table, then the likelihood (which is provided by nickname table from Deron Meranda) is used as the similarity; if one is an initial letter and the other is a full first name, the similarity is a predefined similarity (we use 0.85); if both first names are initial letters and are the same, $s = 0.85^2 = 0.7225$; for other cases, $s = 0$. We use strict rules for last names which are the most important components in names: if they are the same, $s = 1$ and otherwise $s = 0$. Here are some examples. *Weiyi Meng* and *Weiyi Meng* is a perfect match; *Weiyi Meng* and *W. Meng* is a partial match with lower similarity; *W. Meng* and *W. Meng* is a weaker partial match with even lower similarity even though the two strings are identical; *Robert Smith* and *Bob Smith* also match with high similarity because *Bob* is variant of *Robert*. We also apply a set of rules to calculate the similarity for middle names, nicknames and titles, so our person name matching algorithm can handle most cases.

The overall similarity is calculated from similarities for each name component. Let $\bar{s}$ be the overall similarity of two names. We consider the following cases: (a) if all component similarities are 1, $\bar{s} = 1$ and the two names are fully matched; (b) if one of the component similarities is 0, $\bar{s} = 0$ and the two names are not matched; (c) if all component scores are positive, $\bar{s} = \sqrt[k]{\alpha}$, where α is the average of the component scores and k is determined by the following rules. Let m be the number of component scores from non-null matches (i.e., both of the involved items in a match are non-null). If $m \leq 2$, make $k = 1$ (which means $\bar{s} = \alpha$). If $m > 2$, make $k = m - 1$. The idea is that if two names have more non-null components and they all match to some extent, then their matching score should be higher.

Book title similarity Suppose T_1 and T_2 are two book titles, first we remove stop words in titles, then each title is tokenized into a sequence of words. Next we find the longest common subsequence of T_1 and T_2, at word level (not at character level). In order to calculate common subsequence, we need to determine if two words are matched. In this paper, two words are considered to be matched if they are the same or one is the prefix of the other. The reason for considering this type of prefix match is because terms in titles (like journals) are frequently represented by prefixes. Suppose $LCS(T_1, T_2)$ is the length of longest common subsequence, $Len(T)$ is the length of T, the overall similarity is $LCS(T_1, T_2)/\min(Len(T_1), Len(T_2))$. For example, consider two titles, T_1 is *Journal of psychosomatic research* and T_2 is *J-Psychosom-Res*. After removing stop words, and tokenizing, $T_1 = [Journal, psychosomatic, research]$, $T_2 = [J, Psychosom, Res]$. Because $Journal$ matches J, $psychosomatic$ matches $Psychosom$, $research$ matches Res, so $LCS(T_1, T_2) = 3$, and the similarity between the two titles is 1.

Number similarity While computing the similarity between two numbers, we also need all possible numbers in the domain or the range of the domain. For a set of numbers, suppose the min is the minimum value, max is the maximum value, then for numbers n_1 and n_2, the distance is $|n_1 - n_2|/(max - min)$. The similarity between two numbers is 1 minus their distance. The similarity is therefore normalized into [0, 1]. For example, if we know the price range of a set of books is from \$10 to \$50, and book A's price is \$30, book B's price is \$20, the distance is (30 - 20) / (50 - 10) = 0.25, and the similarity is 1 - 0.25 = 0.75.

Date similarity We first parse a date to year, month and day based on a list of common date patterns. Date has three date types — year-type (e.g., 1997), month-type (e.g., June 2005), and day-type (e.g. 25 July 2006). If the two dates have different date types, we prefer year-type computation first, and then month-type and day-type. For year-type, the distance is the difference in years; for month-type, the distance is the difference in months; and for day-type, the distance is the difference in days.

Finally, by considering the maximum date range for all the dates under the corresponding date attributes, we normalize the distance into [0, 1]. For example, for a set of dates, the minimum date is Jan 1, 2001; the maximum date is Jan 1, 2011. Suppose we want to calculate the similarity of two dates, say Jan 1, 2005 and Feb 2005. Because Feb 2005 is month-type, so month-type is used for the calculation. The distance between Jan 1, 2005 and Feb 2005 is 1 month, considering that the maximum distance is 120 months, the normalized distance is 1/120, and the similarity is $1 - 1/120$.

4.2. Goodness of matching functions

To measure the goodness of a matching function, we propose the notion of *positive confidence* and *negative confidence*.

Definition 10 *Given two values of an attribute, assume a matching function gives a value* $\alpha \in [0, 1]$, *which represents the likelihood of* match. *We call* α *as* positive confidence *and* $1 - \alpha$ *as* negative confidence.

A matching function is good if it gives a large *positive confidence* to a *match* and gives a large *negative confidence*

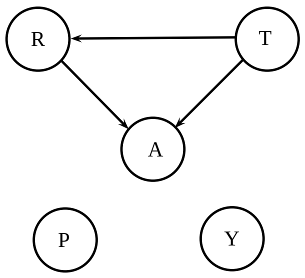

Figure 2. A Bayesian network built for Example 3.1, where nodes T, A, P and Y represent matching results for *Title*, *Authors*, *Publisher* **and** *Year*, **respectively, and R represents record matching results.**

Table 4. Abbreviations in Tables 5, 6, 7, and 8

Abbrev.	Text
NGT	Newmam-Girvan modularity threshold
BA	Blocking Attributes
NMB	Number of Matches in Blocker
RMB	Recall of Matches in Blocker
NPO	Number of Pairs Output
PPO	Proportion of Pairs Output
T	simple blocker on Title
A	simple blocker on Author
TA	composite blocker on Title and Author
APC	Average Positive Confidence for matches
ANC	Average Negative Confidence for non-matches
HM	Harmonic Mean of APC and ANC

to a *non-match* in the real world. For a matching function on a labeled data set, we generally use three measures to evaluate its goodness: *average positive confidence* for all real-world matches in the data set, *average negative confidence* for all real-world non-matches in the data set, and harmonic mean of these two measures.

5. Bayesian network as record matcher

In this paper, we apply the approach of structure learning of Bayesian network described in [31] to data sets in the books domain. We will present our method of inference from a Bayesian network after structure learning.

After applying the structure learning approach in [31] to the labeled data in Table 3, we get a Bayesian network as shown in Fig. 2. And the conditional probability specifier for each node is also determined, the network can be used for inference. Assume we have built the Bayesian network with nodes $s_1, \dots, s_N$ and r, where s_i is an attribute matching result and r is the unknown record matching result. We want to know the probability distribution $p^*(r|s_1, \dots, s_N)$

Table 5. abebook-amazon. The number of matches in total is 171. The number of pairs in total is 7,800,000. The match rate is $1/45614$. See Table 4 for abbreviations.

NGT	0			0.025		
BA	T	A	TA	T	A	TA
NMB	162	105	165	162	108	165
RMB(%)	94.7	61.4	96.5	94.7	63.2	96.5
NPO	2487	3765	5949	6653	9552	15688
PPO(%)	.032	.048	.076	.085	.122	.201

from the Bayesian network.

Inference by Markov blanket The Markov blanket of a node is a set of its parents, its children, and its children's parents [23].

The Markov blanket of a node shields off impact from other nodes to this node. The state of a node only depends on the state of its Markov blanket. Then we have

$$p(r|s_1, \dots, s_N) = p(r|\partial r),$$

where ∂r is the Markov blanket of node r in the Bayesian network. For example, in Fig. 2, the node R is the *record matching result* with value *match* or *non-match*. R's Markov blanket is $\{A, T\}$.

However, $p(r|\partial r)$ cannot be conveniently and precisely derived directly from labeled data when ∂r includes too many nodes and labeled data are limited compared to the size of the feature space. Here we present a method by which we derive $p(r|\partial r)$ from the conditional probability specifiers of the Bayesian network, which have been computed in structure learning.

According to conditional independence rules in the Bayesian network, we have

$$p(r|\partial r) = \frac{p(r\,\partial r)}{\sum_{r \in R} p(r\,\partial r)} \propto p(r\,\partial r), \tag{3}$$

where $\propto$ means *proportional to*. Here $\sum_{r \in R} p(r\,\partial r)$ is considered as a normalizing constant and is ignored.

And we also have

$$p(r\,\partial r) = p(r|Par(r)) \prod_{s \in \partial r} p(s|Par_r(s)), \tag{4}$$

where $Par(r)$ denote r's parents in the network and $Par_r(s)$ is defined as $Par(s) \cap (\{r\} \cup \partial r)$. The reason to use $\cap$ here is that we are concerned with only nodes in $\{r\} \cup \partial r$, and $Par(s)$ may include nodes not in $\{r\} \cup \partial r$. In other words, $Par_r(s)$ excludes nodes that are not r and not in r's Markov blanket.

We can verify that $p(r|Par(r))$ and $p(s|Par_r(s))$ can be obtained directly from the conditional probability specifiers of the Bayesian network, or obtained through simple

Table 6. abebook-bookpool. The number of matches in total is 789. The number of pairs in total is 23,286,000. The match rate is $1/29513$. See Table 4 for abbreviations.

NGT	0			0.025			0.05			0.1			0.15		
BA	T	A	TA	T	A	TA	T	A	TA	T	A	TA	T	A	TA
NMB	510	384	642	576	450	675	624	501	732	630	501	732	630	528	735
RMB(%)	64.6	48.7	81.4	73.0	57.0	85.6	79.1	63.5	92.8	79.8	63.5	92.8	79.8	66.9	93.2
NPO	4042	6418	9927	14387	18507	31928	31147	34937	64687	55410	47814	101622	77917	67099	143070
PPO(%)	.017	.028	.043	.062	.079	.137	.134	.150	.278	.238	.205	.436	.335	.288	.614

Table 7. bookpool-amazon. The number of matches in total is 849. The number of pairs in total is 5,045,300. The match rate is $1/5943$. See Table 4 for abbreviations.

NGT	0			0.025			0.05			0.1			0.15		
BA	T	A	TA	T	A	TA	T	A	TA	T	A	TA	T	A	TA
NMB	595	615	783	617	628	796	627	635	802	640	636	808	644	636	808
RMB(%)	70.1	72.4	92.2	72.7	74.0	93.8	73.9	74.8	94.5	75.4	74.9	95.2	75.9	74.9	95.2
NPO	2451	3594	5539	8513	12525	20395	14626	18599	32510	30018	28269	57403	57993	35085	72057
PPO(%)	.049	.071	.110	.169	.248	.404	.290	.369	.644	.595	.560	1.14	.753	.695	1.43

summation of the specifiers. Therefore, from Eq. (3) and Eq. (4), we can compute conditional probability distribution $p(r|\partial r)$, on which the *match* or *non-match* decision is based.

For the example in Fig. 2, according to Eq. (3) and Eq. (4), we have $p(R|\partial R) = p(R|T,A) \propto p(R,T,A) = p(T)p(R|T)p(A|R,T) \propto p(R|T)p(A|R,T)$. Here $p(T)$ is ignored because it makes no difference for $R = 1$ and $R = 0$ and thus has no impact on the decision. Obviously, $p(R|T)$ and $p(A|R,T)$ are directly from the conditional probability specifiers. Then $p(R|\partial R)$ can be computed for record matching decision. If $p(R = 1|\partial R) > p(R = 0|\partial R)$, corresponding two records are considered as a *match*. Otherwise they are considered as a *non-match*.

6. Experiments

6.1. Data sets

Basic data sets In this paper, we focus on data sets in *book* domain. Each data set is a relational table, and having a different but overlapping attribute set. The three tables are abebooks (6000 records), bookpool (3881 records) and amazon (1300 records). We have nine attributes as the *universal attribute set* of the book domain, including *title* (T), *author* (A), *publisher* (P), *publication time* (PT), *price* (PR), *binding* (B), *edition* (E), *detail url* (DU), and *desc* (D).

Derived data sets Based on blocking results, we match attribute values and derive the data sets of attribute matching results. A derived data set is based on two tables, e.g. abebook and bookpool. Each attribute in the universal attribute set will be matched. An example derived data set is shown in Table 3. In this paper we will use three derived data sets: *abebook-amazon*, *abebook-bookpool* and *bookpool-amazon*.

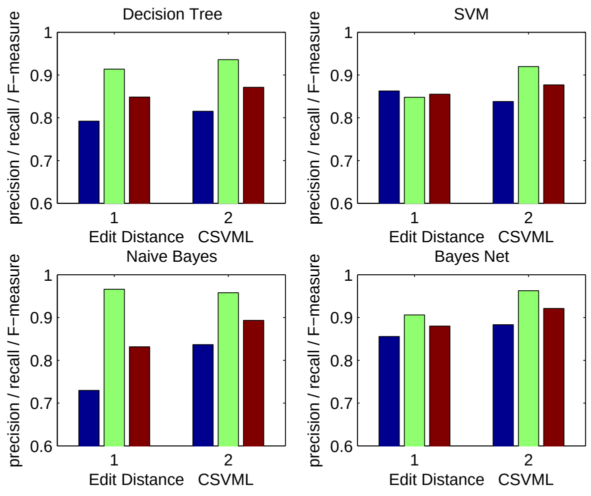

Figure 3. Performance comparison of CSVML and edit distance (baseline) on data set bookpool-amazon by four classifiers.

6.2. Blocker

We apply SPAN blocking algorithm to these data sets, selecting *author* and *title* as blocking attributes.

Tables 5, 6 and 7 show some interesting statistics of blocking in order to generate the derived data sets. Abbreviations in these tables are in Table 4. We evaluate a blocker on (1) recall of matches in blocker (RMB) and (2) proportion of pairs output over all pairs (PPO). PPO measures the extent to which pairwise comparisons are reduced and the smaller is better. Generally PPO less than 1% is acceptable. In all three tables, we can see the composite blocker (TA)

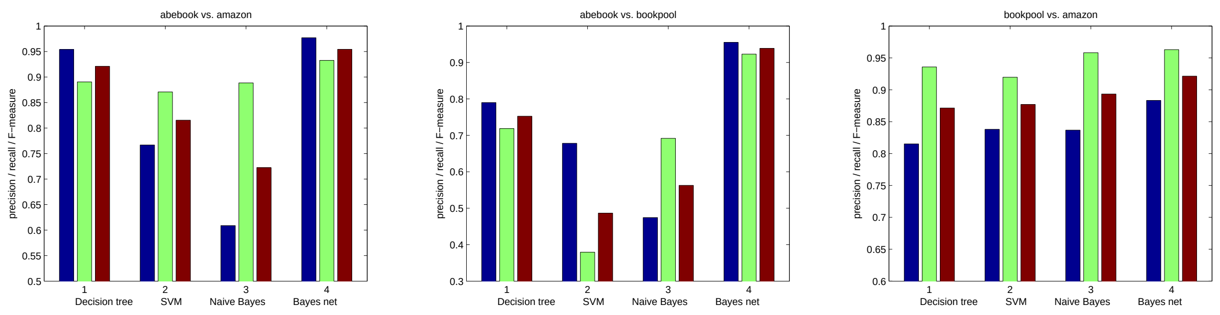

Figure 4. Performance comparison of four record matchers on three data sets: abebook-amazon (left), abebook-bookpool (center) and bookpool-amazon (right).

has larger RMB than simple blockers (T) and (A).

The three tables show that our framework BARM works well with SPAN and in it SPAN is effective and efficient.

Table 8. Goodness comparison of normalized edit distance (baseline) and CSVML (person name)

Functions	Normalized edit distance			CSVML(person name)		
Measures	APC	ANC	HM	APC	ANC	HM
abebook-amazon	0.974	0.640	0.772	0.979	0.989	0.984
abebook-bookpool	0.983	0.664	0.793	0.984	1.0	0.992
bookpool-amazon	0.984	0.657	0.788	0.994	1.0	0.997

6.3. Attribute matchers

Goodness of CSVML We proposed a method in Section 4.2 to measure the goodness of a matching function. By this method, we compare two matching functions: normalized edit distance (baseline) and person name matcher in our package CSVML on three data sets. Results in Table 8 show CSVML's person name matching function has higher APC, ANC and HM, especially ANC and HM, due to name specific semantics being considered by CSVML.

Performance comparison In our comparison experiments, we use the Bayesian network as described in Section 5. As a comparison, we use three other classifiers — Decision Tree (C4.5), Support Vector machines (SVM) and Naive Bayes Classifier, which are from WEKA software package of machine learning [33].

We compare CSVML and edit distance. We get derived data by using person name distance on attribute *author* and book title distance on attribute *title*. As a comparison, we used edit distance on the two attributes above to get derived data for baseline.

We apply four classifiers to the data set *bookpool-amazon* with 10-fold cross validation. And the results are shown in Fig. 3. There are two groups in each sub-figure for edit distance and CSVML, respectively. Each group consists of three values: precision, recall and F-measure, from left to right.

For the results, we see that CSVML is better than the baseline in all four classifiers. This is because CSVML takes semantics of string into account. For person name matching, CSVML is aware of different name components and therefore provide more precise matching. For book title matching, CSVML removes unnecessary stop words.

6.4. Record matchers

In this paper, we use classifiers as record matchers. We compare four classifiers: Decision Tree (C4.5), Support Vector machines (SVM), Naive Bayes Classifier, and Bayesian network on three data sets: *abebook-amazon*, *abebook-bookpool*, and *bookpool-amazon*. 10-fold cross validation is used as well. The experimental results are in Fig. 4. In each figure, there is one group for each record matcher. Each group consists of three values: precision, recall and F-measure, from left to right.

The Bayesian network performs the best on all three data sets. Bayesian network models the causal relationship between attributes. As a probabilistic method, Bayesian network has an advantage that it is not subject to overfitting as training data size increases. The experiments show Bayesian network's advantage on the book domain.

7. Conclusions

In this paper, we present a generic framework (BARM) for entity resolution for relational data sets. BARM is convenient for blocking and matching algorithms to fit into it because of its loose coupling.

We also presented a precise definition of *blocker* or *blocking algorithm* and applied SPAN blocking algorithm to the book domain. We also proposed a sparse blocking matrix to store the blocking results. This makes the blocking results reusable and easy to be combined into a *composite blocker*.

For attribute matching, we proposed CSVML. CSVML considers semantics and context of in the attribute values and therefore led to better performance.

In this paper, we for the first time applied Bayesian network to the entity resolution problem. We also compared Bayesian network with other record matchers. We found that in the book domain, Bayesian network outperforms other classifiers, because of its probabilistic feature and free of overfitting. In the future, we plan to apply more attribute matchers and classifiers to other domains.

Acknowledgments

This work is supported in part by the following grants: CNS-0958501 from NSF and R01-LM010817 from NIH. We also would like to thank Yi-Jing Tan for her involvement in developing the name matching function and Xiaofeng Meng for sharing a data set used in our experimental evaluation.

References

[1] C. Batini and M. Scannapieco. Data quality: Concepts, methodologies annd techniques. Springer, 2006.

[2] I. Bhattacharya and L. Getoor. Deduplication and group detection using links. In *ACM SIGKDD Workshop on Link Analysis and Group Detection (LinkKDD-04)*, 2004.

[3] I. Bhattacharya and L. Getoor. Iterative record linkage for cleaning and integration. In *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD)*, 2004.

[4] I. Bhattacharya and L. Getoor. A latent Dirichlet model for unsupervised entity resolution. In *SDM*, 2006.

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

[6] A. Chapman. *Principles and Methods of Data Cleaning*. Report for the Global Biodiversity Information Facility, 2005.

[7] P. Christen and T. Churches. Febrl: Freely extensible biomedical record linkage release 0.3, 2005. http://datamining.anu.edu.au/linkage.html.

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

[9] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(1):1–16, 2007.

[10] S. Euijong Whang, D. Menestrina, G. Koutrika, M. Theobald, and H. Garcia-Molina. Entity resolution with iterative blocking. In *SIGMOD*, 2009.

[11] W. Fan, X. Jia, J. Li, and S. Ma. Reasoning about record matching rules. In *VLDB*, 2009.

[12] I. Fellegi and A. Sunter. A theory for record linkage. *J. of the American Statistical Society*, 64(328):1183–1210, 1969.

[13] L. Gravano, P. G. Ipeirotis, N. Koudas, and D. Srivastava. Text joins in an RDBMS for web data integration. In *WWW*, pages 90–101, 2003.

[14] S. Guha, N. Koudas, A. Marathe, and D. Srivastava. Merging the results of approximate match operations. In *VLDB*, 2004.

[15] O. Hassanzadeh, F. Chiang, H. C. Lee, and R. J. Miller. Framework for evaluating clustering algorithms in duplicate detection. In *VLDB*, 2009.

[16] M. Hernandez, M. A. H. Andez, S. Stolfo, and U. Fayyad. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data Mining and Knowledge Discovery*, 2(1):9–37, 1998.

[17] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

[18] C. D. Manning, P. Raghavan, and H. Schutze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[19] A. McCallum, K. Nigam, and L. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *KDD*, pages 169–178, 2000.

[20] H. B. Newcombe, J. M. Kennedy, S. J. Axford, and A. P. James. Automatic linkage of vital records. *Science*, 130(3381):954–959, 1959.

[21] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69(2):026113+, Feb 2004.

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

[23] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988.

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

[25] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

[26] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *KDD*, pages 269–278, 2002.

[27] L. Shu, A. Chen, M. Xiong, and W. Meng. Efficient spectral neighborhood blocking for entity resolution. In *ICDE*, pages 1067–1078, Hannover, Germany, April 2011.

[28] L. Shu, B. Long, and W. Meng. A latent topic model for complete entity resolution. In *ICDE*, 2009.

[29] E. Ukkonen. Approximate string matching with q-grams and maximal matches. *Theoretical Computer Science*, 92(1):191–211, 1992.

[30] V. S. Verykios and A. K. Elmagarmid. Automating the approximate record matching process. *Information Sciences*, 126:83–98, 1999.

[31] J. Williamson. *Bayesian Nets and Causality: philosophical and computational foundations*, chapter 3, pages 14 – 48. Oxford University Press, 2005.

[32] W. Winkler. Overview of record linkage and current research directions. Technical report, US Bureau of the Census, 2006.

[33] I. Witten and E. Frank. *Data mining: practical machine learning tools and techniques, second edition*. Morgan Kaufmann, San Francisco, 2005.