Efficient Spectral Neighborhood Blocking for Entity Resolution

Liangcai Shu [#1], Aiyou Chen [†2], Ming Xiong [‡3], Weiyi Meng [#4]

[#] *Dept. of Computer Science, SUNY at Binghamton*
PO BOX 6000, Binghamton, NY 13902, USA
[1] lshu@cs.binghamton.edu
[4] meng@cs.binghamton.edu

[†] *Bell Labs, Alcatel-Lucent*
700 Mountain Ave, Murry Hill, NJ 07974, USA
[2] aiyouchen@google.com

[‡] *Google Inc.*
76 9th Avenue, New York, NY 10011, USA
[3] mxiong@google.com

***Abstract*—In many telecom and web applications, there is a need to identify whether data objects in the same source or different sources represent the same entity in the real-world. This problem arises for subscribers in multiple services, customers in supply chain management, and users in social networks when there lacks a unique identifier across multiple data sources to represent a real-world entity. Entity resolution is to identify and discover objects in the data sets that refer to the same entity in the real world.**

We investigate the entity resolution problem for large data sets where efficient and scalable solutions are needed. We propose a novel unsupervised blocking algorithm, namely *SPectrAl Neighborhood* (SPAN), which constructs a fast bipartition tree for the records based on spectral clustering such that real entities can be identified accurately by neighborhood records in the tree. There are two major novel aspects in our approach: 1) We develop a fast algorithm that performs spectral clustering without computing pairwise similarities explicitly, which dramatically improves the scalability of the standard spectral clustering algorithm; 2) We utilize a stopping criterion specified by *Newman-Girvan* modularity in the bipartition process. Our experimental results with both synthetic and real-world data demonstrate that SPAN is robust and outperforms other blocking algorithms in terms of accuracy while it is efficient and scalable to deal with large data sets.

I. Introduction

In many telecom and web applications, there is a need to identify whether data objects in the same source or different sources represent the same entity in the real-world. This problem is known as *Entity Resolution* (ER), which is also known as record matching [1][2], record linkage [3][4][5], or deduplication [6][7]. The problem arises quite often in information integration where data objects representing the same "real-world" entity are presented in different ways, and there usually lacks a unique identifier in the system to represent a real-world entity. As an example, Telecom equipment suppliers can be referred as "Alcatel-Lucent", "Alcatel Lucent" and "Lucent" on different web pages while they all represent the same company. This, however, 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 entity resolution problem [8]. There are usually two criteria to judge those algorithms, namely statistical efficiency (i.e. *accuracy*) and computational efficiency. Statistical efficiency refers to correctness of the result, i.e., whether the data objects representing the same entity have been identified correctly, and computational efficiency refers to the complexity of the algorithms to resolve the entity resolution problem. Since entity resolution techniques need to be applied to large amounts of diverse information in telecom and web applications, it is quite important for them to be both statistically efficient and computationally efficient.

Blocking is an important technique for improving the computational efficiency of entity resolution algorithms. In blocking, all objects are assigned to a set of blocks, usually of small sizes. Objects from different blocks are not considered as matches, i.e., it is impossible for them to refer to the same entity. Therefore, *pairwise* comparisons are only necessary for pairs of objects within the same block to identify whether they represent the same entity or not, which eliminates the need of $O(n^2)$ pairs of comparison, for a data set of n objects. The key issue is then how to identify such blocks efficiently.

In prior work, there are three representative algorithms for blocking, i.e., *sorted neighborhood* (SN) [9], *bigram indexing* (BI) [10], and *canopy clustering* (CC) [11]. SN is one of the most computationally efficient blocking algorithms in the literature, and it has a time complexity of $O(n \log n)$. Unfortunately, it fails to capture the *pairwise* similarities between data objects if two similar strings start with different characters, e.g., "Alcatel-Lucent" and "Lucent-Alcatel". On the other hand, BI and CC capture pairwise similarities better than SN, but they both have computational complexities of $O(n^2)$ [12]. Thus they do not scale well with large data sets.

In this paper, we propose an efficient *SPectrAl Neighbor-*

hood (SPAN) algorithm for blocking in entity resolution. Our approach is based on *spectral clustering* [13], one of the most advanced clustering techniques in statistics and machine learning. SPAN significantly improves captured pairwise similarities on SN while empirically achieves the same time complexity, i.e., in $O(n \log n)$ time. SPAN is *unsupervised*, i.e., there are no training data sets needed for our algorithm. Similar to the aforementioned blocking algorithms, SPAN is *unconstrained*, i.e., it does not require as input the number of blocks or other domain specific parameters. This makes it applicable in many real-world applications where the number of blocks is *unknown* beforehand.

Although the performance of *spectral clustering* has been well studied [13][14][15], existing spectral clustering algorithms, which are based on matrices of pairwise similarities, are very limited and cannot be applied in blocking as it is highly expensive (e.g., it can be $O(n^2)$ in terms of both time and space complexity) to deal with large-scale data sets. A few fast algorithms [16][17] have been developed recently for performing spectral clustering, but they are all based on approximate solutions, either based on sampling (e.g. the Nyström method) or low rank matrix approximation. Low rank approximation does not apply here, because in the entity resolution problem, there can be as many as $O(n)$ entities (clusters), and thus the rank of the similarity matrix can be as high as $O(n)$. Sampling on the other hand will introduce unavoidable errors depending on the trade-off between performance degradation and computational gain. Fortunately, motivated by the sparse representation of records in the vector space model, we are able to design an efficient algorithm that performs *fast* bipartition spectral clustering (two clusters). Using this as a building block, we take the first step towards designing an efficient and scalable blocking algorithm to solve the entity resolution problem. Specifically, the SPAN algorithm takes a sparse qgram representation of records (i.e. the vector space model) based on *term frequency* and *inverse document frequency* (tf-idf) [18] as its input, and utilizes our *fast* bipartition spectral clustering algorithm to hierarchically bipartition the data points into clusters until it meets the stopping criterion specified by *Newman-Girvan* modularity [19]. The bipartition process constructs a binary tree, which can be used to examine the similarity of records in neighboring clusters for constructing blocks.

Our contributions include the following:

- We propose a novel blocking algorithm based on spectral clustering, namely SPAN. To the best of our knowledge, our algorithm is the first one that employs spectral clustering for blocking in large scale entity resolution problems. It has two novel features that differentiate itself from other existing techniques.
  1) We design a fast algorithm for bipartition spectral clustering, which is a building block for SPAN, and does not require the similarity matrix to be computed explicitly as the matrix may be too expensive to compute and store when the number of data points is large. Our analysis demonstrate that SPAN is as efficient as $O(Jn \log n)$ in terms of computational time complexity, where J is no more than the average string length of the records.
  2) In the construction of the binary tree, SPAN hierarchically bipartitions the data points into clusters until it meets the stopping criterion specified by the *Newman-Girvan* modularity, which has been successfully used in social network analysis. To the best of our knowledge, our algorithm is the first to utilize Newman-Girvan modularity as the stopping criterion in the spectral clustering partition process.
- We study our algorithm and other blocking algorithms, namely SN, BI and CC, extensively in our experiments. We conduct experiments with both synthetic and real-world data sets from a telecom application. Our experimental results demonstrate that SPAN outperforms other blocking algorithms in accuracy for data sets where the amount of errors is from low to medium level, which is usually the case of real-world data sets. Our experimental results also demonstrate that the SPAN algorithm can be scaled to deal with large data sets.

The rest of the paper is organized as follows. Section II presents a preliminary of *spectral clustering* and *Newman-Girvan modularity*. Section III presents the details of our SPAN algorithm. Section IV presents our experimental results in detail. Section V discusses the related work, and Section VI concludes the paper.

II. Preliminary

In this section, we first introduce spectral clustering which will be used to derive our blocking algorithm. Then we describe the Newman-Girvan modularity as a stopping criterion for blocking.

A. Spectral clustering

Given n points, let $\mathbf{A}$ be an $n \times n$ symmetric matrix, where its (i, j)th entry measures the similarity between the ith and jth points. The goal of spectral clustering is to partition the n points into K disjoint clusters, and different spectral clustering methods formulate the partitions in different ways. We adopt the normalized cut (Ncut) formulation [13]. For $V_1, V_2 \subset \{1, \cdots, n\}$, let $w(V_1, V_2) = \sum_{i \in V_1, j \in V_2} A(i, j)$ be the total similarity between points in V_1 and V_2. The Ncut criterion defines the partition with $K = 2$ by the following optimization criterion (letting $V = \{1, \cdots, n\}$):

$$Ncut(V_1, V_2) \quad = \quad \frac{w(V_1, V_2)}{w(V_1, V)} + \frac{w(V_1, V_2)}{w(V_2, V)},$$

where V_1 and V_2 give a binary partition of the n points, i.e. $V_1 \cap V_2$ is empty and $V_1 \cup V_2 = \{1, \cdots, n\}$. The quantity $Ncut(V_1, V_2)$ measures a normalized similarity between points in V_1 and V_2 and thus minimization of the quantity defines a meaningful partition.

The above optimization problem is NP hard [13]. Spectral clustering based on $Ncut$ is derived as follows: 1) rewrite

Record#	Address
1	600 MOUNTAIN AVENUE
2	700 MOUNTAIN AVE
3	600-700 MOUNTAIN AVE
4	100 DIAMOND HILL RD
5	100 DIAMOND HILL ROAD
6	123 SPRINGFIELD AVENUE
7	123 SPRINFGIELD AVE

Fig. 1. Example records

$Ncut(V_1, V_2)$ as a normalized quadratic form of an indicator vector assigning points to V_1 and V_2, and 2) by replacing the indicators with real-values, solve an equivalent generalized eigenvector problem for the (normalized) graph Laplacian $\mathcal{L}$ of $\mathbf{A}$ defined as follows:

$$\mathcal{L}(\mathbf{A}) \quad = \quad \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2} \tag{1}$$

where $\mathbf{D} = diag(\mathbf{A1})$ with $\mathbf{I}$ being the identity matrix and $\mathbf{1}$ being a column vector of 1's [20]. Eventually, spectral clustering defines a binary partition of the points based on the sign of the entries in the eigenvector that corresponds to the second smallest eigenvalue of $\mathcal{L}(\mathbf{A})$. We will simply use the term *second smallest eigenvector* to stand for this eigenvector later. Note that for $x \in R^n$

$$x^T\mathcal{L}(\mathbf{A})x \quad = \quad \frac{1}{2}\sum_{i,j=1}^{n}\left(\frac{x_i}{\sqrt{\mathbf{D}(i,i)}} - \frac{x_j}{\sqrt{\mathbf{D}(j,j)}}\right)^2 \mathbf{A}(i,j),$$

thus the second smallest eigenvector of $\mathcal{L}(\mathbf{A})$ is the nontrivial minimizer of the above quadratic norm, since the smallest eigenvalue is always 0. This implies that records in the same cluster/entity that have large mutual similarities (i.e. $\mathbf{A}(i,j)$) and similar degrees (i.e. $\mathbf{D}(i,i) \approx \mathbf{D}(j,j)$) are projected onto local neighborhood in the second smallest eigenvector. Therefore, spectral clustering preserves locality nicely. When $K > 2$, the above binary partition is implemented recursively to obtain K partitions. There have been different forms of spectral clustering, based on different ways of normalization. We use the above version, due to its nice statistical consistency property [15] and great success in applications [13].

Example II.1: To illustrate the idea of spectral clustering, we provide a simple example of 7 records (see Figure 1) with its similarity matrix $\mathbf{A}$ shown in Table I. The second smallest eigenvector of the Laplacian matrix $\mathcal{L}(\mathbf{A})$ is $(0.26,\ 0.24,\ 0.25,\ -0.61,\ -0.61,\ 0.20,\ 0.18)$. Since the fourth and fifth entries of the eigenvector are negative while others are positive, spectral clustering will partition the 7 records into two clusters as $(1,\ 2,\ 3,\ 6,\ 7)$ and $(4,\ 5)$.

In this example, spectral clustering identifies the 4th and 5th records as one cluster, which belongs to one entity, and the remaining records (two entities) into the second cluster. It is interesting to note that spectral clustering does not cut the same entity into two clusters, which is consistent with the general locality preserving property of spectral clustering as mentioned in the above.

TABLE I
A SIMILARITY MATRIX FOR 7 RECORDS (SEE FIGURE 1)

	1	2	3	4	5	6	7
1	1	0.423	0.56	0.004	0.004	0.24	0.013
2	0.423	1	0.466	0.005	0.004	0.013	0.043
3	0.56	0.466	1	0.004	0.003	0.01	0.035
4	0.004	0.005	0.004	1	0.71	0.008	0.009
5	0.004	0.004	0.003	0.71	1	0.008	0.008
6	0.24	0.013	0.01	0.008	0.008	1	0.513
7	0.013	0.043	0.035	0.009	0.008	0.513	1

However, most spectral clustering algorithms are based on a precomputed similarity matrix $\mathbf{A}$, which is typically not sparse for the entity resolution problem, and thus are very expensive in terms of both space and computational cost for large n. Recently a few approximate algorithms have been proposed to speed up the computation of spectral clustering, which applies only when data can be presented in some low dimensional space [17]. Unfortunately it is nontrivial to represent all n records in the same low dimensional space without significant information loss [21]. A few alternatives use sampling (e.g. the Nyström method) to reduce computational complexity which will introduce unavoidable errors [16][17]. In this paper, we contribute a fast and scalable blocking algorithm that makes use of spectral clustering in a novel way without the concern of dimension reduction or sampling.

Note that spectral clustering itself does not give a precise way to measure the quality of the two clusters, which is however, important for the blocking problem, since we need to determine whether the points in V_1 and V_2 need to be further partitioned. This is essentially a model selection problem, which is technically still open in the statistics literature. We address this problem based on the Newman-Girvan modularity which has been recently introduced in the literature of social networks.

B. Newman-Girvan modularity

Clustering has a corresponding name "community detection" in the literature of social networks [19]. Given n nodes and links among the nodes, suppose that V_1 and V_2 give a binary partition of the nodes that defines two communities. Let $\mathcal{A}$ denote a symmetric matrix that represents the connectivity strength among the n nodes. The Newman-Girvan (NG) modularity is a quantity introduced by [19] for measuring the strength of the two communities, defined as follows:

$$Q(V_1, V_2) \quad = \quad \sum_{k=1}^{2}\left(\frac{O_{kk}}{L} - \left(\frac{L_k}{L}\right)^2\right), \tag{2}$$

where $O_{kk} = \sum_{i\in V_k, j\in V_k}\mathcal{A}(i,j)$ is the total number of connections among nodes in V_k, $L_k = \sum_{i\in V_k} d_i$ for $k = 1, 2$, and $L = \sum_{i=1}^{n} d_i$, where $d_i = \sum_{j=1}^{n}\mathcal{A}(i,j)$ denotes the degree of the ith node. Note that $L^{-1}O_{kk}$ is simply the observed connectivity density among nodes in V_k, and $(L^{-1}L_k)^2$ is the

expectation of the connectivity density when connections are randomly assigned to the $\binom{n}{2}$ pairs of nodes given the node degrees. Therefore, the NG modularity has a nice physical interpretation: $Q(V_1, V_2)$ measures how much the communities are stronger than the ones that are generated from randomly connected nodes. The larger $Q(V_1, V_2)$ is, the stronger the communities are. One would naturally claim no evidence for V_1 and V_2 to be two communities if $Q(V_1, V_2) = 0$, which corresponds to random connections. The NG modularity also has nice asymptotic properties such as statistical consistency [22]. We will adopt the concept of Newman-Girvan modularity as a stopping criterion for our blocking algorithm.

III. Spectral neighborhood blocking

Below we present SPAN for solving the entity resolution problem. Our method is based on the vector space model [23], 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 is "LUCENT" and $q = 3$, the corresponding qgrams are "##L", "#LU", "LUC", "UCE", "CEN", "ENT", "NT$" and "T$$", where '#' and '$' are the beginning and ending padding characters [8], respectively.

We first define the similarity matrix for the records based on the vector space model, and then derive SPAN based on spectral clustering. The Newman-Girvan modularity introduced in Section II-B is used as the stopping criterion for blocking.

A. The vector space model

Note that each record is a string and can be decomposed into multiple qgrams. Suppose that m is the total number of qgrams that appear in n records. Let $\mathbf{B}_1$ be an $n \times m$ matrix, where $\mathbf{B}_1(i, j)$ denotes how many times the jth qgram appears in the ith record. So $\mathbf{B}_1$ characterizes records by qgrams, which is the so-called vector space model. Below we define a record-record similarity based on this vector space model.

Given the record-qgram relation matrix $\mathbf{B}_1$, let $\mathbf{B}_2$ be an $n \times m$ matrix defined as follows: for $1 \le i \le n, 1 \le j \le m$,

$$\mathbf{B}_2(i, j) \quad = \quad \log(n/d_j)\mathbf{B}_1(i, j). \tag{3}$$

Here $d_j = \sum_{k=1}^{n} \delta[\mathbf{B}_1(k, j)]$ and function $\delta(x)$ is defined as: $\delta(x)$ is 1 if $x \ge 1$ and 0 if $x = 0$, where x is a nonnegative integer. Note that d_j is called the document frequency and $\mathbf{B}_2$ is also called the tf-idf weight matrix [23], which has been successfully applied in information retrieval for document representation. Here each record is taken as a document and each qgram as a word. The coefficient $\log(n/d_j)$ downgrades the weight for the jth qgram if it appears very often in all records.

Now given the tf-idf weight matrix $\mathbf{B}_2$ for the records, we define the similarity matrix $\mathbf{A}$ for each pair of the records by the cosine correlation, that is, the similarity between the ith and jth records is:

$$\mathbf{A}(i, j) \quad = \quad \frac{\sum_{k=1}^{m} \mathbf{B}_2(i, k)\mathbf{B}_2(j, k)}{\sqrt{\sum_{k=1}^{m} \mathbf{B}_2^2(i, k)}\sqrt{\sum_{k=1}^{m} \mathbf{B}_2^2(j, k)}}. \tag{4}$$

Obviously, the larger $\mathbf{A}(i, j)$ is, the closer the two records are. There have been many other ways to define similarity between two records. However, the above similarity metric defined from the vector space model not only captures the blocking structures accurately (demonstrated in our experiments later) but enables the development of an efficient approach for blocking.

Note that by defining $\mathbf{B}$ from $\mathbf{B}_2$ as

$$\mathbf{B}(i, j) \quad = \quad e_i^{-1}\mathbf{B}_2(i, j), \tag{5}$$

where $e_i = \sqrt{\sum_{k=1}^{m} \mathbf{B}_2^2(i, k)}$ is the Euclidean norm of the ith row of $\mathbf{B}_2$, we can rewrite $\mathbf{A}$ as:

$$\mathbf{A} \quad = \quad \mathbf{B}\mathbf{B}^T, \tag{6}$$

where the superscript ^{T} denotes matrix transpose. We call $\mathbf{B}$ the *normalized record-qgram matrix*, which can be computed quickly as above given the records.

The basic idea of our blocking algorithm is to partition records into small clusters based on the record-record similarity matrix $\mathbf{A}$ that allows fast neighborhood search for candidate record pairs, with the goal that each such pair of records belong to the same entity. Given the similarity matrix $\mathbf{A}$, there have been many methods available for clustering, and spectral clustering has been one of the successful ones. However, as mentioned earlier, spectral clustering cannot be implemented efficiently, since first, $\mathbf{A}$ is too expensive to compute and store for large n, and second, there is no simple way to represent all records in the same low dimensional space. This motivates us to develop a novel algorithm for fast and scalable blocking based on spectral clustering, where $\mathbf{A}$ is not computed explicitly, as described in the next subsections.

Remark. One can also use words instead of qgrams for defining the record-record similarity. We prefer qgrams instead of words for two reasons: (1) qgrams capture more local information than words which is important when words are noisy; (2) the total number of qgrams can be much smaller than that of words for large scale data sets when q is small and thus are more convenient to manipulate. How to choose q is domain- or language-specific. Work in [24] showed that $q = 4$ is a good choice for European languages. In our data sets, records or documents are relatively short, with the average length between 20 and 30 characters. In our experiments, we find that 3-grams give better performance. But 4-grams do not improve accuracy since noisy qgrams are introduced. Even worse, they increase dimensionality of data sets and degrade time performance. Thus in all simulation and experimental studies later, the results are reported with $q = 3$.

B. Sparsity analysis of matrices

As showed in Figure 2, as the number of records increases, the number of non-zero entries of the record-qgram matrix is approximately proportional to the number of records, much smaller than that of the record-record matrix, which is quadratic to the number of records. That means the record-record similarity matrix $\mathbf{A}$ is expensive to obtain, with time complexity $O(n^2)$. In contrast, the record-qgram matrix $\mathbf{B}_1$ is

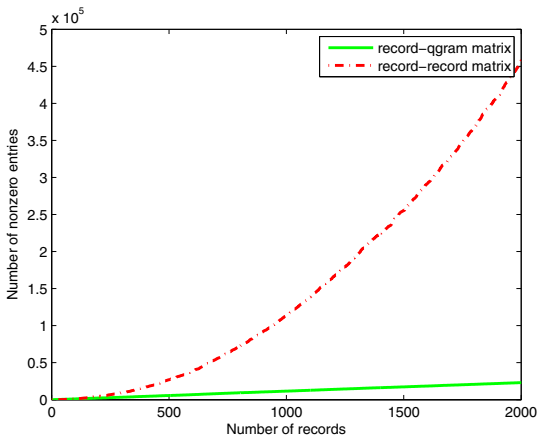

Fig. 2. Comparison of the numbers of non-zero entries in the record-record matrix and the record-qgram matrix.

sparse and cheap to obtain with time $O(Jn)$, where J is the average number of qgrams in a record. In our application, J is between 20 and 30, much smaller than n.

According to Eq. (3) and (5), matrix $\mathbf{B}$ can be efficiently computed since both computations in Eq. (3) and (5) can be done in time $O(Jn)$.

Sparse matrices exist in many applications of information retrieval and data mining. In our application, the sparsity property holds nicely because the average length of the records approximately remains stable as the number of records increases.

C. Fast bipartition spectral clustering

Recall that bipartition spectral clustering reduces to computing the second smallest eigenvector of the Laplacian matrix $\mathcal{L}(\mathbf{A})$ as defined in Eq. (1). The key of our fast algorithm is based on a mathematical equivalence as described in the theorem below.

Theorem 1: Given an $n \times m$ normalized record-$q$gram matrix $\mathbf{B}$, let $\mathbf{A} = \mathbf{B}\mathbf{B}^T$ be its corresponding record-record similarity matrix and let

$$\mathbf{D} = diag(\mathbf{B}(\mathbf{B}^T\mathbf{1})),$$

where $diag(\cdot)$ transforms a vector into a diagonal matrix. Define a matrix $\mathbf{C}$ by

$$\mathbf{C} = \mathbf{D}^{-1/2}\mathbf{B}. \tag{7}$$

Then the second smallest eigenvector of $\mathbf{A}$'s Laplacian matrix $\mathcal{L}(\mathbf{A})$ is the second left singular vector corresponding to the second largest singular value of $\mathbf{C}$.

Proof: The proof is in Appendix A. ■

Theorem 1 tells that for the bipartition spectral clustering introduced in Section II-A, we only need to compute the second left singular vector of $\mathbf{C}$.

It is easy to see that $\mathbf{C}$ can be computed quickly due to the sparsity of $\mathbf{B}$. Recall that the singular vectors of a matrix can be computed by the power method (orthogonal iteration)

```
struct TreeNode {
   TreeNode List:  childrenlist;
   Record Set:     block;
};
```

Fig. 3. Structure of TreeNode in Algorithm 1

(see Section 7.3 of [25] for details). Furthermore, according to Theorem 7.3.1 of [25], the complexity for computing the second maximum left singular vector of the sparse matrix $\mathbf{C}$ is $O(Jn\log(\frac{1}{\epsilon})/\log(\frac{\lambda_2}{\lambda_3}))$, where λ_2 and λ_3 are the second and third largest singular values of $\mathbf{C}$, ϵ is a prespecified convergence criterion, and Jn is the number of non-zero entries of $\mathbf{C}$. In practice, $\frac{\lambda_2}{\lambda_3}$, which indicates the strength of the bipartition, is typically bounded away from 1 [26][18]. In such cases, the computational complexity is simply $O(Jn)$. Thus the bipartition spectral clustering based on our similarity matrix $\mathbf{A}$ can be performed quickly as follows:

1) compute the second maximum singular vector of $\mathbf{C}$ defined by (7), and
2) assign the n records to one of two subsets according to the signs of the corresponding entries in the singular vector.

We call this procedure *fast-bipartition* for simplicity.

D. Efficient bipartitioning process of SPAN

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

The bipartitioning process of SPAN algorithm is described in detail below.

Let $\mathbf{B}$ be the normalized record-qgram matrix for the n records, then the binary tree is constructed by starting with the root node that contains all n records and growing itself by recursively splitting each existing leaf node as follows:

1) Given a leaf node that contains a subset R_s of records in a data set, perform the above fast-bipartition procedure where the input is the submatrix that consists of the rows of $\mathbf{B}$ corresponding to R_s, and obtains a bipartition say R_{s1} and R_{s2};
2) Compute the Newman-Girvan modularity say $Q(R_{s1}, R_{s2})$ for the above bipartition according to Section II-B;
3) If $Q(R_{s1}, R_{s2}) > 0$, split the node R_s into two leaf nodes R_{s1} and R_{s2} and grow the tree; otherwise, this node is a final leaf and it is not split.

For detailed algorithm, please see Algorithm 1. The node of the tree is defined as in Figure 3. Each leaf node of the tree contains a block of records. For inner nodes, field `block` is **null**.

Algorithm 1 Bipartitioning process of SPAN

Global:
R: data set, as input.
a: attribute for blocking, as input.
$\mathbf{B}$: normalized record-$q$gram matrix of R w.r.t a.
S: set of blocks of records, as output.
T: root node of a bipartition tree, as output.
void BLOCKING-BY-HIERARCHICAL-CLUSTERING()
Method:

1: Generate matrix $\mathbf{B}$ of data set R w.r.t attribute a.
2: $S \leftarrow \phi$
3: $T \leftarrow$ **null**
4: RECURSIVE-BIPARTITION(R, **null**)

void RECURSIVE-BIPARTITION(R_s, par)
Input:
R_s: subset of data set R.
par: parent of current node.
Method:

1: $cur \leftarrow$ **new** `TreeNode()` // current node
2: **if** $par \neq$ **null then**
3: Add cur to par.`childrenlist`.
4: Get $\mathbf{B}_s$ corresponding to data set R_s, where $\mathbf{B}_s$ consists of some rows of matrix $\mathbf{B}$ and $\mathbf{B}_s = \mathbf{B}$ if $R_s = R$.
5: $\mathbf{D} \leftarrow \mathrm{diag}(\mathbf{B}_s(\mathbf{B}_s^T\mathbf{1}))$ // get degree matrix of $\mathbf{B}_s\mathbf{B}_s^T$
6: $\mathbf{C} \leftarrow \mathbf{D}^{-1/2}\mathbf{B}_s$
7: Split R_s into R_{s1} and R_{s2} according to fast-bipartition procedure on C.
8: Compute Newman-Girvan modularity Q for this split.
9: **if** $Q \leq 0$ **then**
10: // cancel this split and consider R_s as a block
11: cur.`block` $\leftarrow R_s$
12: $S \leftarrow S \bigcup \{R_s\}$ // add $\{R_s\}$ to set of blocks
13: **else**
14: // validate this split and continue splitting
15: RECURSIVE-BIPARTITION(R_{s1}, cur)
16: RECURSIVE-BIPARTITION(R_{s2}, cur)
17: **if** $par =$ **null then**
18: $T \leftarrow cur$ // cur is root node and $R_s = R$

The bipartition tree construction falls into the general framework of divisive hierarchical clustering.

Note that in item 1) one may also use a locally normalized matrix based on just the subset of the records, instead of a submatrix of the global normalized matrix $\mathbf{B}$.

Below we illustrate our SPAN algorithm with records in Example II.1 in details.

Example III.1: First each record is decomposed into qgrams with $q = 3$, which results in 72 qgrams in total. For example, the second record "700 MOUNTAIN AVE" is decomposed into qgrams as follows: "##7", "#70", "700", "00␣", "0␣M", "␣MO", "MOU", "OUN", "UNT", "NTA", "TAI", "AIN", "IN␣", "N␣A", "␣AV", "AVE", "VE$" and "E$$". And then the 7×72 record-qgram matrix with tf-idf entries is computed and normalized by rows to obtain the sparse matrix $\mathbf{B}$. Note that the similarity matrix $\mathbf{A} = \mathbf{B}\mathbf{B}^T$ shown in Table I was only for the purpose of demonstration and is not needed by SPAN. Now we describe how the binary tree is built as below.

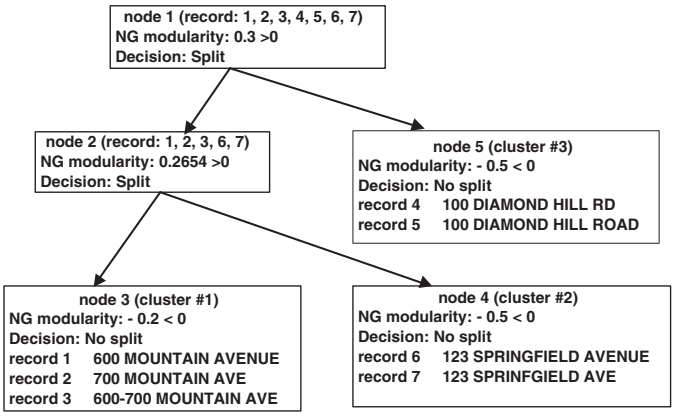

Fig. 4. Bipartition tree in Example III.1

To build the first level of the tree, i.e. to bipartition all seven records at the root node, compute matrix $\mathbf{C}$ from $\mathbf{B}$ and the second left singular vector of $\mathbf{C}$, from which the signs of the entries lead to the bipartition $(1,\ 2,\ 3,\ 6,\ 7)$ and $(4,\ 5)$, as in Example II.1. The NG modularity is computed as 0.3, which is positive and thus the tree grows at this node.

Now it is necessary to check each of the two new nodes. To bipartition $(1,\ 2,\ 3,\ 6,\ 7)$, compute $\mathbf{C}$ (5×72) based on the submatrix of $\mathbf{B}$ which consists of the rows of $(1,\ 2,\ 3,\ 6,\ 7)$ and its second left singular vector. This gives a bipartition $(1,\ 2,\ 3)$ and $(6,\ 7)$, and the corresponding NG modularity is computed to be 0.2654, which is positive and thus the tree grows to level 3 at this node. However, for bipartition of the node $(4,\ 5)$ to (4) and (5), since the NG modularity is computed to be -0.5, which is negative, this bipartition is invalid and the tree does not grow at the node $(4,\ 5)$.

Similarly, the third level nodes of the tree, i.e. $(1,\ 2,\ 3)$ and $(6,\ 7)$ are checked. Neither of the nodes can be further bipartitioned because the NG modularities for both of them are negative. So the tree does not grow further. The final bipartition tree and the complete bipartition process are depicted in Figure 4.

E. Neighborhood search of SPAN

After the bipartition tree is generated, we search for the neighborhood in order to generate candidate record pairs that are input of an entity resolution algorithm (e.g. [27][28][6][29][30][4][31][32][33]) for accurate matching which generally considers all the attribute values of records and is assumed to be costly. We do not address the entity resolution algorithm as it is beyond the scope of this paper.

We define $distance$ as $1 - similarity$. We know each leaf node of the tree represents a cluster of records. Let T_{inter} and T_{intra} (both in $[0, 1]$) denote two pairwise-distance thresholds where the subscripts $inter$ and $intra$ stand for inter- and intra-clusters, respectively. First the bipartition tree is generated according to Section III-D. Then the candidate record pairs

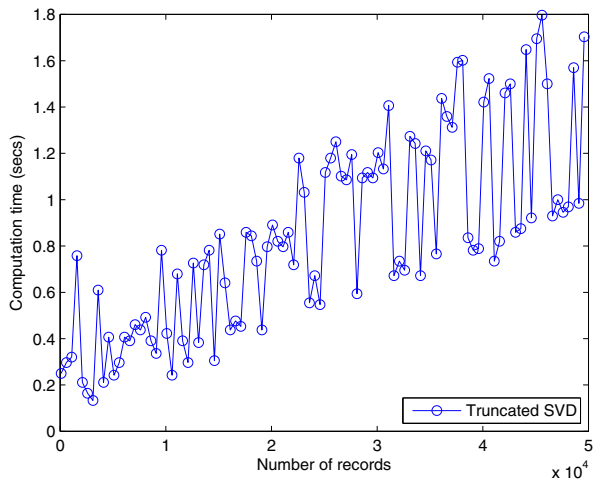

Fig. 5. The computation time of truncated SVD of matrix $\mathbf{C}$ for bipartitioning. It is bounded by $O(Jn)$ where n is the number of records, and $J = 24$ is the average number of qgrams in a record. The computation time is also approximately proportional to the number of non-zero entries of matrix $\mathbf{C}$.

are generated as follows: (1) for each record pair (R_i, R_j) in a cluster, if the distance between R_i and R_j is less than T_{intra}, then this pair is a candidate record pair; and (2) for each record pair (R_i, R_j) where R_i and R_j are from two neighboring leaf nodes, if the distance is less than T_{inter}, then this record pair is a candidate record pair.

We regard two leaf nodes as neighbors if they are close to each other in the tree, i.e. the path length in terms of the number of edges between them on the tree is small. In our experiments, we take the path length 4 as the threshold to define the neighbors.

It is important to point out that the total number of such pairwise examinations is $O(n)$ as the size of a neighborhood is bounded. Therefore, to examine those pairs, one can in fact afford an even finer similarity metric than the tf-idf metric used in the fast construction of the bipartition tree. In our experiments, we simply use the same similarity metric for the pairwise examinations, and the results are reported in Section IV.

We expect that the performance is robust to the choices of T_{inter} and that the best performance is achieved when T_{inter} is close to 0 and T_{intra} is close to 1, where the extreme case reduces to simply claiming only records in the same leaf as neighbors. The intuition is that two records in the same leaf are either very similar or brought together through a third record, which is similar to both records. The latter case is called *transitivity*, a nice property preserved by the nature of spectral clustering. These are also verified by our experiments.

F. Time complexity analysis

1) Pre-bipartitioning time

Given n records, let J be the average number of qgrams in a record. We can obtain matrix $\mathbf{B}$ in time $O(Jn)$ as mentioned in Section III-B. It is easy to confirm that matrix $\mathbf{C}$ can be obtained in time $O(Jn)$ according to Eq. (7).

2) One bipartitioning time

a) Truncated SVD computation: We use truncated Singular Value Decomposition (SVD) for bipartitioning, in which only the first two singular values and singular vectors need to be computed. This is much more efficient than the full SVD. Its computational time is proportional to the number of non-zero entries in the matrix. The truncated SVD is done on matrix $\mathbf{C}$. Since $\mathbf{C}$ is a sparse matrix with Jn non-zero entries, the time for truncated SVD is approximately $O(Jn)$ as mentioned in Section III-C, see Figure 5 for some empirical evidence.

b) NG modularity computation: Note that the NG modularity formula (2) is based on matrix $\mathcal{A}$, which represents connectivity among the n nodes and diagonal elements are 0. However, in our similarity matirx $\mathbf{A}$, all diagonal elements are 1. Therefore it is necessary to deduct diagonal 1's while applying the NG modularity to $\mathbf{A}$. And by Eq. (6) we have

$$\begin{aligned} L &= \sum_{i=1}^{n}\sum_{j=1}^{n}\mathbf{A}(i,j) - n = \mathbf{1}^T\mathbf{A}\mathbf{1} - n \\ &= (\mathbf{B}^T\mathbf{1})^T(\mathbf{B}^T\mathbf{1}) - n, \end{aligned}$$

which can be computed quickly due to the sparsity of $\mathbf{B}$. Similarly O_{kk} and L_k defined in (2) are computed by deducting diagonal 1's and can also be computed quickly. Thus the NG modularity for a bipartition of n records can be computed directly based on $\mathbf{B}$ with complexity $O(Jn)$.

3) Total bipartitioning time

After bipartition, there would be on average $O(\log n)$ levels in the bipartition tree.

Furthermore, as discussed earlier, one *fast-bipartition* procedure takes time $O(Jn)$. Then we can derive bipartitioning for nodes in each level with complexity $O(Jn)$. Therefore, the average time complexity of our SPAN algorithm is $O(Jn \log n)$.

4) Time for neighborhood search

In SPAN, besides generating a binary tree in $O(Jn \log n)$, the additional time of pairwise comparison within cluster and between clusters is $O(ms^2)$ where s is the maximum size of clusters, and m is the number of clusters. In practice, the maximum number of each cluster's neighboring clusters is fixed, typically small. Thus the time for neighborhood search is no more than $O(n)$. Therefore the total average time for SPAN is dominated by the binary tree construction.

Overall, the time complexity of SPAN can be written as $O(Jn \log n)$, which is much faster than the state of the art blocking algorithms such as canopy clustering [11] and bigram indexing [10][34].

IV. Experiments

To study the performance of SPAN, we use three popular measures: precision, recall and F1-measure, where *precision* is defined as the proportion of correctly identified record pairs, i.e., two records matching the same entity, to candidate pairs returned by a blocking algorithm, *recall* is the proportion of correctly identified record pairs to the correct record pairs

TABLE II
ACCURACY COMPARISON OF BLOCKING ALGORITHMS

Datasets	SPAN			CC			BI			SN		
	precis.	recall	F1	precis.	recall	F1	precis.	recall	F1	precis.	recall	F1
H1-500	0.763	0.517	0.617	0.611	0.679	**0.643**	0.577	0.340	0.428	0.226	0.264	0.243
H2-500	0.888	0.621	0.731	0.900	0.698	**0.786**	0.889	0.485	0.628	0.423	0.426	0.425
M1-500	0.980	0.769	**0.862**	0.756	0.808	0.781	0.722	0.695	0.708	0.508	0.573	0.539
M2-500	0.987	0.948	**0.967**	1.000	0.860	0.925	1.000	0.614	0.761	0.637	0.650	0.643
L1-500	0.999	0.952	**0.975**	0.988	0.961	0.974	0.862	0.864	0.863	0.570	0.680	0.620
L2-500	1.000	0.956	**0.977**	0.973	0.955	0.964	0.930	0.619	0.743	0.660	0.679	0.669
H1-1500	0.598	0.355	0.446	0.615	0.506	**0.555**	0.299	0.347	0.322	0.189	0.192	0.190
H2-1500	0.908	0.531	**0.670**	0.628	0.716	0.669	0.577	0.478	0.523	0.382	0.385	0.384
M1-1500	0.936	0.671	**0.781**	1.000	0.560	0.718	0.611	0.518	0.560	0.497	0.538	0.516
M2-1500	0.971	0.911	**0.940**	0.958	0.842	0.896	0.814	0.626	0.708	0.634	0.617	0.625
L1-1500	0.987	0.882	**0.931**	0.882	0.912	0.896	0.787	0.659	0.717	0.574	0.607	0.590
L2-1500	0.998	0.939	**0.968**	0.959	0.913	0.936	0.819	0.693	0.750	0.647	0.666	0.657
H1-5000	0.352	0.168	0.227	0.556	0.297	**0.387**	0.302	0.197	0.238	0.155	0.154	0.154
H2-5000	0.872	0.409	**0.557**	0.902	0.319	0.472	0.496	0.354	0.413	0.356	0.352	0.354
M1-5000	0.907	0.643	**0.753**	0.970	0.546	0.699	0.666	0.383	0.486	0.480	0.496	0.488
M2-5000	0.956	0.887	**0.920**	0.938	0.843	0.888	0.572	0.574	0.573	0.620	0.610	0.615
L1-5000	0.986	0.764	0.861	0.962	0.797	**0.872**	0.645	0.515	0.573	0.568	0.589	0.578
L2-5000	0.989	0.908	**0.947**	0.923	0.921	0.922	0.754	0.560	0.643	0.648	0.651	0.650

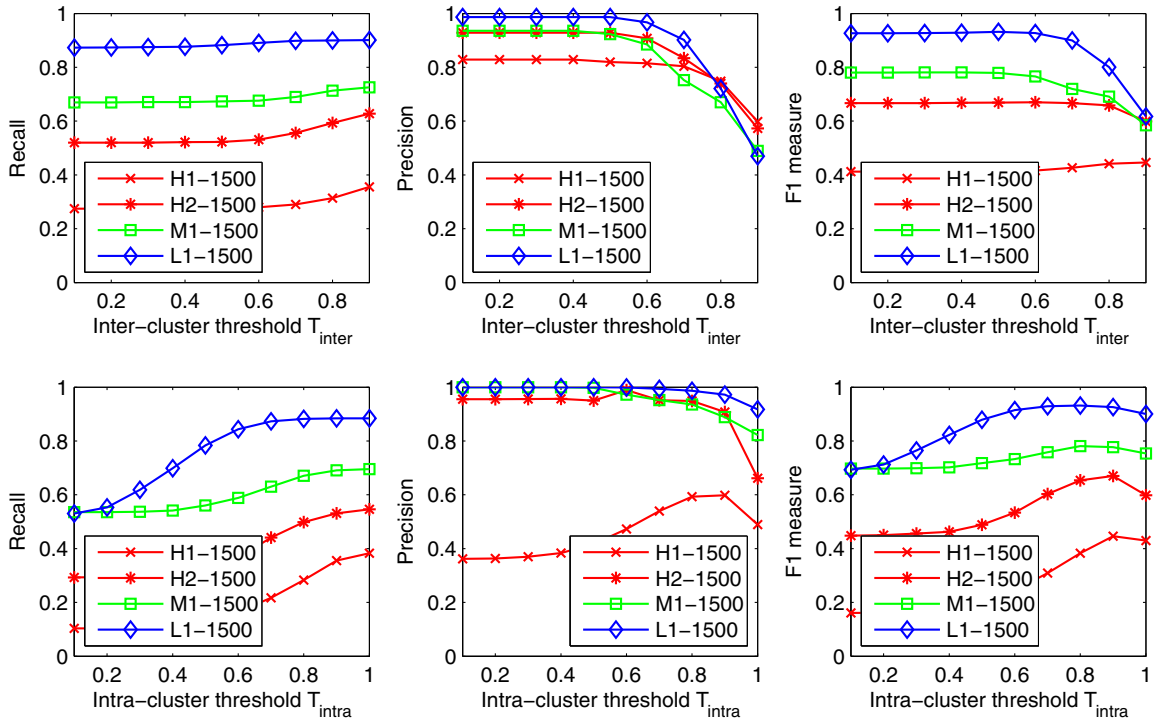

Fig. 6. Insensitivity of SPAN to parameter selection.

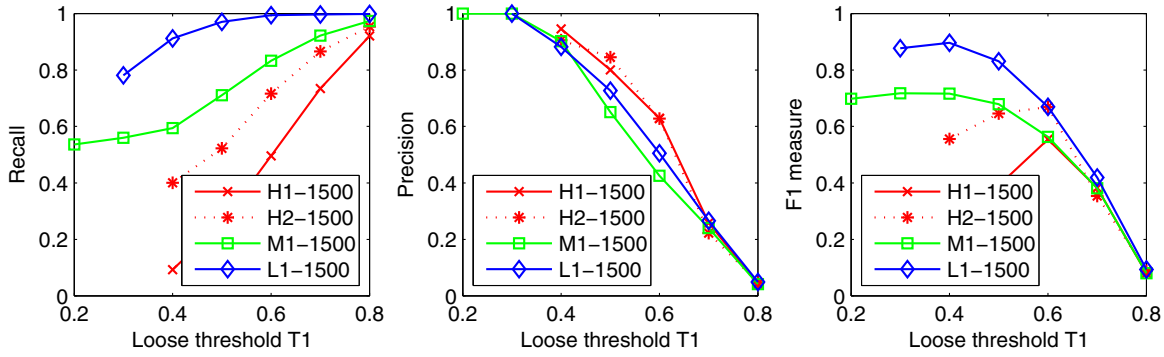

Fig. 7. Sensitivity of CC to parameter (T_1) selection.

based on the ground-truth entities, and *F1-measure* is the harmonic average of precision and recall: $2/(1/\text{precision} + 1/\text{recall})$.

We compare SPAN with three representative blocking algorithms in the literature: *Sorted Neighborhood* (SN), *Canopy Clustering* (CC) and *Bigram Indexing* (BI). Our experiments use both published synthetic data and real data sets. Our results demonstrate that 1) SPAN is fast and scalable to large scale data sets with average time complexity $O(Jn \log n)$, while CC and BI are not; 2) SPAN outperforms the other three algorithms in cases when data have low or medium noise, which is often the case of real-world applications; and 3) SPAN is much more robust than both CC and bigram indexing in the sense that its performance is very robust to the tuning parameters, while the performance of CC and BI highly depend on a fine choice of the tuning parameters, which requires lots of labeled data and thus is often not possible with data sets in real-world applications.

We first describe the implementation details of SPAN, as well as the data sets and then report the accuracy and efficiency of SPAN, in comparison with BI, SN and CC. In our experiments, company names are used as the blocking attribute.

A. Data sets

Below is a description of two types of datasets used in our experiments. The labeled data is necessary so that we can compare the *accuracy* (i.e., quality of resulting blocks) of different algorithms.

1) Labeled data: The labeled data, in which the real entities are known for records, are synthesized data obtained from the University of Toronto [33]. In these data sets, J, the average number of qgrams in a record, is between 20 and 30. We select six medium-sized data sets: two high-noise data sets (H1 and H2), two medium-noise data sets (M1 and M2), and two low-noise data sets (L1 and L2). Each of them includes about 5000 records, and each record has the same attribute - company name. For each of the six data sets, we also take the first 500 and 1500 records to form new data sets. In total, we get 18 data sets, named as H1-500, H1-1500, H1-5000, and so on.

2) Unlabeled data: We have a large real corporate dataset which is unlabeled, called A500k, with about 500,000 records, which will be used only for time complexity investigation. We again use company names as the blocking attribute, which has an average string length 24 and then J is about 26.

In addition, we use a publicly accessible large data set from Netflix [35], which is a large sparse matrix with 500,000 users as rows and 17,770 movies as columns, on average 200 movies per user, i.e., the average number of features in a record is $J = 200$. Here the sparse matrix $\mathbf{B}_1$ indicates which movies each user has rated.

B. Performance comparison

In BI [12][10], if two records share c bigrams, they are assigned to the same block and each record can be assigned to multiple blocks, where $c = t * b$, b is the average number of bigrams in the records and $t \in (0, 1]$ is a tuning parameter. CC has two parameters [11] - loose threshold T_1 and tight threshold T_2 with $0 < T_2 < T_1 < 1$. Briefly in CC, by starting from the list of n records, the algorithm randomly picks a record as a center, includes records with distance within T_1 as a canopy, and removes records with distance within T_2 from the list, which is repeated until the list is empty. In CC, canopies can overlap. We run SPAN, BI, SN and CC on all 18 labeled data sets in our experiments.

1) Accuracy: The accuracy comparisons in terms of precision, recall and F1 measures are given in Table II, which lists the best performance of each algorithm on the data sets. For SPAN in Table II, threshold T_{intra} is 0.9 for all data sets, and threshold T_{inter} is set as follows: 0.8 for high-noise data sets H1 and H2, 0.5 for medium-noise data sets M1 and M2, and 0.3 for low-noise data sets L1 and L2. Table II demonstrates that CC and SPAN significantly outperform SN and BI. For example, consider the data set M1-5000, the *F1-measure*s of SPAN, CC, BI and SN are 0.753, 0.699, 0.486 and 0.488, respectively. SPAN is the best among all algorithms, and it is more than 50% better than BI and SN. Note that BI gives equal weight to each bigram, thus it suffers from the large number of common bigrams derived from stopping words (e.g. 'co' and 'om' from 'com') that appear frequently in the data. On the other hand, both SPAN and CC use tf-idf weights, which give

TABLE III
TIME COST (SECONDS) OF SPAN AND CC ON DATA SET A500K.

No. of records	SPAN	CC
500	10.58	11.37
5000	34.02	278.31
10000	50.11	3095.76
50000	259.57	*
100000	811.65	*
500000	3485.21	*

different weights to frequently occurred substrings and less frequent ones. Finally, SN has the major drawback of relying on the key selected for sorting data records. Thus only keys of good quality cause similar records to be close to each other, which is not necessarily the case in our experiments.

Table II also shows that, overall, SPAN outperforms CC almost uniformly on medium- and low-noise data sets.

In the presence of high-noise data sets, there is no clear cut whether CC or SPAN outperforms the other. For example, Table II presents F1 results for H1-5000 and H2-5000 data sets, which are opposite. It is unclear to us why sometimes CC may outperform SPAN, while SPAN may outperform CC at other times. We plan to investigate this in our future work.

We expect that SPAN in general outperforms CC in real-world applications, as spectral clustering is capable to capture not only the pairwise similarity of the records, but also the connection between records through transitivity, whereas canopy clustering captures only pairwise similarity.

2) Robustness: We next investigate the robustness of SPAN and CC, in terms of the choice of the tuning parameters. In SPAN, there are two parameters - inter-cluster threshold T_{inter} and intra-cluster threshold T_{intra}. Figure 6 shows the changes of recall, precision and F1 measures for SPAN on four data sets as one parameter changes while the other is fixed. The best value for T_{intra} is around 0.9, while the best value for T_{inter} depends on data - it is higher for high-noise data than for low-noise data, i.e., we need higher dissimilarity for high-noise data when matching two records, which is reasonable. Figure 7 shows the similar tests for CC. However, the precision, recall and F1 measure vary dramatically for CC, almost from 0 to 1 as the loose threshold T_1 varies in $[0, 1]$, although they vary not that much as the tight threshold T_2 varies. Therefore, CC is quite sensitive to the tuning parameter T_1, whose value is difficult to decide in practice. Meanwhile, the performance of SPAN does not change so much as the tuning parameters vary. This means that SPAN is more robust and more practical to use than CC, although both are unsupervised learning. The robustness of SPAN is due to the bipartition tree, which gives a sound and preliminary partition of the records and enables fast neighborhood search.

C. Comparison of time complexity

In order to investigate how the time cost of the algorithms grows with the number of records, this subsection presents our results with the two real unlabeled data sets. As analyzed in Section III-F, SPAN is at the level of $O(n \log n)$, as fast as SN in terms of time complexity, although SPAN has a bigger constant than SN. But SN does not work as accurately as SPAN. As far as we know there is no other blocking algorithm at the level of $O(n \log n)$. As SPAN and CC outperform BI and SN significantly in Table II, we only focus on studying the former two algorithms in this subsection.

TABLE IV
TIME COST (SECONDS) OF SPAN AND CC ON DATA SET NETFLIX.

No. of records	SPAN	CC
50000	814.18	13594.82
100000	1560.17	70405.10
500000	7496.78	*

Table III shows the comparison of running time for SPAN and CC on the larger data set A500k, in which * stands for the case where the algorithm does not finish within 24 hours. Table IV shows the comparison on data set Netflix [35]. Obviously SPAN significantly outperforms CC in terms of running time. And the numbers for SPAN in the tables are consistent with $O(n \log n)$.

V. RELATED WORK

There have been many methods and tools developed for entity resolution (see [36][37][38][8] 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 [27], logistic regression [28], support vector machines [6] and decision trees [29]) are *supervised* learning approaches, while co-training on clustering [30], probabilistic model [4], topic model [31][32], and clustering algorithms [33] are *unsupervised* learning approaches. Our approach is in the category of *unsupervised* learning algorithms.

This paper proposes SPAN, a novel and efficient *blocking* algorithm based on spectral clustering. Blocking has been studied extensively in the literature (See [12] for a blocking review), and various blocking techniques have been proposed, including sorted neighborhood (SN) [9], bigram indexing (BI) [10][34], and canopy clustering (CC) [11]. However, our blocking technique 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 $O(Jn \log n)$. In particular, our analysis and experimental results demonstrate that our algorithm outperforms benchmark blocking algorithms.

[39] presented a simple and scalable graph clustering method called power iteration clustering (PIC). Different from our work, their focus is not on blocking. [40] studied a related problem of name disambiguation using K-way spectral clustering under the usual clustering framework, where the number of clusters is small, and their approach does not apply to large-scale problems.

There has been lots of literature on distance based neighborhood search. The most popular algorithms are kd-tree and its

variations [41], which only apply to low dimensional feature spaces. Studies in high dimensional neighborhood search [42] have mostly focused on generic problems and shown that it is impossible to achieve the complexity of $O(n \log n)$. SPAN provides a fast and approximate solution for neighborhood search in high dimensional data space. The key for SPAN to achieve fast computation is to take advantage of the sparse structure in the feature space that the average number of qgrams is only J though the total number of qgram can be as large as n.

Iterative blocking has been proposed in [43], which combines multiple single-attribute blocking results to reduce false negatives (i.e., improve recall). Our work is related to iterative blocking because each single-attribute blocking result can be generated by one of the aforementioned blocking methods, i.e., SN, BI, CC, or SPAN.

There have been various similarity measures available e.g., edit distance [44], token-based n-gram [45], and character-based $q$gram [46]. These can also been used for neighborhood search in the second step of SPAN.

Our work also differs from *rule-based* [2][47][9] approaches as those ones usually utilize semantics of the data, which can only be derived from expert knowledges. While our techniques can be applied in the absence of such knowledges, it is worth pointing out that ours can also be enhanced by such rules. This, however, is left as our future work.

As a fast clustering algorithm, our work should be compared with state-of-the-art methods that have the same assumptions as our solution. But some clustering algorithms in the literature listed above cannot be compared with our solution because they have different assumptions. Currently, we first apply this algorithm to the blocking of entity resolution problem because there was no accurate algorithm efficient enough for this problem, which has extensive applications in data mining and information retrieval.

VI. Conclusions and future work

In this paper, we propose a novel algorithm, namely *spectral neighborhood blocking*, which is an efficient and scalable blocking approach for entity resolution. Our algorithm is developed based on *spectral clustering* to hierarchically partition the data points into blocks until it meets the stopping criterion specified by the *Newman-Girvan* modularity. One of the key features in our algorithm is that it does not require the similarity matrix to be computed explicitly, which enables the scalability of the algorithm as the similarity matrix may be too expensive to compute when the number of data points becomes large. Our analysis and empirical results from real world data sets show that our algorithm is in the time complexity of $O(Jn \log n)$, which is faster than two of the existing blocking algorithms, i.e., bigram indexing and canopy clustering. On the other hand, our experimental results from synthetic data sets demonstrate that our algorithm outperforms three existing state-of-the-art blocking algorithms, i.e., sorted neighborhood, bigram indexing and canopy clustering, in terms of accuracy of blocks when the amount of errors in the data sets varies from low to medium level. This strongly indicates that our algorithm is a very promising solution for entity resolution applications in practice.

We believe our work makes significantly new contributions in efficiency of clustering, namely, to greatly improve the efficiency while keeping the accuracy of spectral clustering. This makes our approach very useful and practical in many applications (including blocking of entity resolution as showed in this paper) where spectral clustering wasn't used.

There are still many open problems to be studied. First, we intend to conduct experiments of our algorithm on large-scale real data sets with labeled data, which, unfortunately, are unavailable to us now. It should be able to shed light on how well our algorithm performs on real data sets. Second, we will investigate how to scale our algorithm in large clusters so that parallel computation can be enabled for our blocking algorithm. Finally, there are still interesting issues in our algorithm that need further investigation, e.g., finding a better metric for examining similarity of records in the spectral neighborhood than the one used in our current algorithm, and how to rank the candidate pairs or blocks similar to [4] but without training data. The investigation of such issues can improve our algorithm even further.

Acknowledgment

We would like to thank Tin Kam Ho at Bell Labs for her valuable comments and suggestions on our work, and Ben Lowe at Bell Labs and Laurence Orazi at Alcatel-Lucent for their help with the data. We also thank Wenbo Zhao at UCSD for good discussion. This work is supported in part by NSF grants IIS-0414981 and CNS-0958501, and a summer research internship at Bell Labs.

This work was done while Ming Xiong was with Bell Labs, Alcatel-Lucent.

Aiyou Chen's current address is Google, Inc., 1600 Amphitheatre Pkwy, Mountain View, CA 94043.

References

[1] H. Pasula, B. Marthi, B. Milch, S. Russell, and I. Shpitser, "Identity uncertainty and citation matching," in *Advances in Neural Information Processing (NIPS)*, 2002. [Online]. Available: http://people.csail.mit.edu/milch/papers/nipsnewer.pdf

[2] W. Fan, X. Jia, J. Li, and S. Ma, "Reasoning about record matching rules," in *The 35th International Conference on Very Large Data Bases (VLDB)*, 2009.

[3] H. B. Newcombe, J. M. Kennedy, S. J. Axford, and A. P. James, "Automatic linkage of vital records," *Science*, vol. 130, no. 3381, pp. 954–959, 1959.

[4] I. Fellegi and A. Sunter, "A theory for record linkage," *Journal of the American Statistical Society*, vol. 64, no. 328, pp. 1183–1210, 1969.

[5] 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*, 2004.

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

[7] I. Bhattacharya and L. Getoor, "Deduplication and group detection using links," in *ACM SIGKDD Workshop on Link Analysis and Group Detection*, 2004.

[8] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios, "Duplicate record detection: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 1, pp. 1–16, 2007.

[9] 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 Knowlege Discovery*, vol. 2, no. 1, pp. 9–37, 1998.
[10] P. Christen and T. Churches, "Febrl: Freely extensible biomedical record linkage release 0.3," 2005, http://datamining.anu.edu.au/linkage.html.
[11] A. McCallum, K. Nigam, and L. Ungar, "Efficient clustering of high-dimensional data sets with application to reference matching," in *Knowledge Discovery and Data Mining*, 2000, pp. 169–178.
[12] R. Baxter, P. Christen, and T. Churches, "A comparison of fast blocking methods for record linkage," in *Proceedings of 9th ACM SIGKDD Workshop on Data Cleaning, Record Linkage and Object Consolidation*, 2003.
[13] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, vol. 22, no. 8, pp. 888–905, 2000.
[14] A. Y. Ng, M. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," in *NIPS 14*, 2002.
[15] U. von Luxburg, M. Belkin, and O. Bousquet, "Consistency of spectral clustering," *Ann. Statist.*, vol. 36, no. 2, pp. 555–586, 2008.
[16] C. Fowlkes, S. Belongie, F. Chung, and J. Malik, "Spectral grouping using the nyström method," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 2, pp. 214–225, 2004.
[17] D. Yan, L. Huang, and M. I. Jordan, "Fast approximate spectral clustering," in *SIGKDD*, 2009, pp. 907–916.
[18] C. D. Manning, P. Raghavan, and H. Schutze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.
[19] M. E. J. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Physical Review E*, vol. 69, no. 2, pp. 026 113+, Feb 2004.
[20] F. Chung, *Spectral Graph Theory*, ser. Number 92 in CBMS Regional Conference Series in Mathematics. American Mathematical Society, 1997.
[21] C. C. Aggarwal, "On the effects of dimensionality reduction on high dimensional similarity search," in *PODS*, 2001, pp. 256–266.
[22] P. J. Bickel and A. Chen, "A nonparametric view of network models and Newman-Girvan and other modularities," *PNAS*, vol. 106, no. 50, pp. 21 068–21 073, 2009.
[23] G. Salton, A. Wong, and C. S. Yang, "A vector space model for automatic indexing," *Communications of the ACM*, vol. 18, no. 11, pp. 613–620, 1975.
[24] P. McNamee and J. Mayfield, "Character n-gram tokenization for european language text retrieval," *Information Retrieval*, vol. 7, no. 1, pp. 73–97, 2004.
[25] G. Golub and C. van Loan, *Matrix Computations (3rd Edition)*. The Johns Hopkins University Press, 1996.
[26] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *Journal of the American Society for Information Science*, vol. 41, pp. 391–407, 1990.
[27] S. Sarawagi and A. Bhamidipaty, "Interactive deduplication using active learning," in *SIGKDD*, 2002, pp. 269–278.
[28] J. C. Pinheiro and D. X. Sun, "Methods for linking and mining massive heterogeneous databases," in *SIGKDD*, 1998.
[29] S. Guha, N. Koudas, A. Marathe, and D. Srivastava, "Merging the results of approximate match operations," in *VLDB*, 2004.
[30] V. S. Verykios and A. K. Elmagarmid, "Automating the approximate record matching process," *Information Sciences*, vol. 126, pp. 83–98, 1999.
[31] L. Shu, B. Long, and W. Meng, "A latent topic model for complete entity resolution," in *ICDE*, 2009.
[32] I. Bhattacharya and L. Getoor, "A latent Dirichlet model for unsupervised entity resolution," in *The SIAM International Conference on Data Mining*, 2006.
[33] O. Hassanzadeh, F. Chiang, H. C. Lee, and R. J. Miller, "Framework for evaluating clustering algorithms in duplicate detection," in *VLDB*, 2009.
[34] W. Cohen and J. Richman, "Learning to match and cluster large high-dimensional data sets for data integration," in *SIGKDD*, 2002.
[35] Netflix, "Netflix prize," http://www.netflixprize.com/index.
[36] W. Winkler, "The state of record linkage and current research problems," *Technical Report, Statistical Research Division, U.S. Bureau of the Census*, 1999.
[37] W. E. Winkler, W. E. Winkler, and N. P, "Overview of record linkage and current research directions," US Bureau of the Census, Tech. Rep., 2006. [Online]. Available: http://www.census.gov/srd/papers/pdf/rrs2006-02.pdf
[38] C. Batini and M. Scannapieco., "Data quality: Concepts, methodologies annd techniques." Springer, 2006.
[39] F. Lin and W. W. Cohen, "Power iteration clustering," in *ICML*, 2010.
[40] H. Han, H. Zha, and C. L. Giles, "Name disambiguation in author citations using a k-way spectral clustering method," in *JCDL'05: Proceedings of the 5th ACM/IEEE-CS joint conference on Digital libraries*. New York, NY, USA: ACM, 2005, pp. 334–343.
[41] D. T. Lee and C. K. Wong, "Worst-case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees," *Acta Informatica*, vol. 9, no. 1, pp. 23–29, 1977.
[42] J. E. Goodman, J. O'Rourke, and P. Indyk, *Handbook of Discrete and Computational Geometry (2nd ed.)*. CRC Press, 2004, ch. Nearest neighbors in high-dimensional spaces.
[43] S. Euijong Whang, D. Menestrina, G. Koutrika, M. Theobald, and H. Garcia-Molina, "Entity resolution with iterative blocking," in *SIGMOD*, 2009.
[44] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet Physics Doklady*, vol. 10, no. 8, pp. 707–710, 1966.
[45] L. Gravano, P. G. Ipeirotis, N. Koudas, and D. Srivastava, "Text joins in an RDBMS for web data integration," in *WWW*, 2003, pp. 90–101.
[46] E. Ukkonen, "Approximate string matching with q-grams and maximal matches," *Theoretical Computer Science*, vol. 92, no. 1, pp. 191–211, 1992.
[47] Y. R. Wang and S. E. Madnick, "The inter-database instance identification problem in integrating autonomous systems," in *ICDE*, 1989, pp. 46–55.

APPENDIX

A. Proof of Theorem 1

Proof: The normalized Laplacian matrix [20] of $\mathbf{A}$ is

$$\mathcal{L}(\mathbf{A}) = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2},$$

where $\mathbf{I}$ is the identity matrix and $\mathbf{D}$ is the degree matrix of $\mathbf{A}$, and with convention $\mathbf{D}^{-1}(i,i) = 0$ for $\mathbf{D}(i,i) = 0$. Then we have

$$\begin{aligned} \mathcal{L}(\mathbf{A}) &= \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{B}\mathbf{B}^T\mathbf{D}^{-1/2} \\ &= \mathbf{I} - \mathbf{C}\mathbf{C}^T. \end{aligned} \tag{8}$$

There exists a factorization of $\mathbf{C}$ for SVD as

$$\mathbf{C} = \mathbf{U}\Sigma\mathbf{V}^T. \tag{9}$$

where $\mathbf{U}$ is an n-by-n orthogonal matrix, $\mathbf{V}$ is an m-by-m orthogonal matrix, and Σ is an n-by-m (rectangular) diagonal matrix with nonnegative real singular values on the diagonal. $\mathbf{U}$ includes left singular vectors, and $\mathbf{V}$ includes right singular vectors.

Put Eq. (9) into the right side of Eq. (8), and consider $\mathbf{U}^T = \mathbf{U}^{-1}$ and $\mathbf{V}^T = \mathbf{V}^{-1}$, we have

$$\mathcal{L}(\mathbf{A}) = \mathbf{U}(\mathbf{I} - \Sigma\Sigma^T)\mathbf{U}^{-1}. \tag{10}$$

Note that Σ^T is m-by-n, different from Σ. Let

$$\Lambda = \mathbf{I} - \Sigma\Sigma^T. \tag{11}$$

It is easy to verify that Λ is a diagonal matrix. Then we conclude that Eq. (10) is the eigendecomposition of matrix $\mathcal{L}(\mathbf{A})$. And eigenvalues are diagonal elements of Λ. According to Eq. (11), the n eigenvalues of $\mathcal{L}(\mathbf{A})$ are $1 - \sigma_1^2 \leq \ldots \leq 1 - \sigma_m^2 \leq 1 = \ldots = 1$, if m singular values $\sigma_1 \geq \ldots \geq \sigma_m$ are given. Eq. (8) also shows that $\mathcal{L}(\mathbf{A})$'s eigenvectors are from matrix $\mathbf{U}$ which include the left singular vectors of the SVD of matrix $\mathbf{C}$. Then the conclusion follows. ■