Entity Disambiguation in Anonymized Graphs Using Graph Kernels

Linus Hermansson,
Tommi Kerola
Chalmers University of Technology
Gothenburg, Sweden
{hlinus,kerola}@student.chalmers.se

Fredrik Johansson,
Vinay Jethava,
Devdatt Dubhashi
Chalmers University of Technology
Gothenburg, Sweden
{frejohk,jethava,dubhashi}@chalmers.se

ABSTRACT

This paper presents a novel method for entity disambiguation in anonymized graphs based on local neighborhood structure. Most existing approaches leverage node information, which might not be available in several contexts due to privacy concerns, or information about the sources of the data. We consider this problem in the supervised setting where we are provided only with a base graph and a set of nodes labelled as ambiguous or unambiguous. We characterize the similarity between two nodes based on their local neighborhood structure using graph kernels; and solve the resulting classification task using SVMs. We give empirical evidence on two real-world datasets, comparing our approach to a state-of-the-art method, highlighting the advantages of our approach. We show that using less information, our method is significantly better in terms of either speed or accuracy or both. We also present extensions of two existing graphs kernels, namely, the direct product kernel and the shortest-path kernel, with significant improvements in accuracy. For the direct product kernel, our extension also provides significant computational benefits. Moreover, we design and implement the algorithms of our method to work in a distributed fashion using the GraphLab framework, ensuring high scalability.

Categories and Subject Descriptors

H.3.3 [**Information Systems**]: [Information Search and Retrieval]

General Terms

Algorithms, Experimentation, Theory

Keywords

entity resolution, entity disambiguation, graph kernels, support vector machines

CIKM '13, Oct 27 - Nov 1, 2013, San Francisco, CA, USA

1. INTRODUCTION

Modern data mining techniques and large data sources like news media allow for extraction of vast amounts of information about entities, such as people and companies. For example, a user might be interested to know which cities TED talks curator Chris Anderson is visiting this year. An automated reply to such a query requires extraction of names and places from texts. This task is made difficult by the existence of Chris's namesake, former Wired Magazine editor-in-chief Chris Anderson. The two individuals work in overlapping fields, and deciding whom is referred to in a news article may be difficult, even when considering context. Resolution of ambiguities in data is a well-studied problem and methods for entity resolution [6, 5, 15, 27], entity matching [7, 24] and entity disambiguation [11, 12, 13, 23] are all aimed towards associating references in text sources with the correct underlying entity. These methods typically make use of similarities in names [6, 5], meta-data [7] or source information [23], to decide which entities underly which references. *Relational entity disambiguation* makes use of network structure between entities [5, 4, 3, 23], sometimes together with additional information.

Big data analysis increasingly faces the challenge of how to preserve user anonymity [1]. While a number of privacy preserving mechanisms have been studied [14], the practical applications are still at a nascent stage [20]. Often, a simple approach to address this privacy concern is using anonymization at source, prior to subsequent data mining, by assigning pseudorandom identifiers to entities. In this setting, existing techniques [7, 11, 12, 13, 24, 15, 27, 5, 6], building on similarity of entity attributes, are rendered inapplicable, while the method presented in this paper is inherently well-suited. To the best of our knowledge, very little work has been done in this setting. Moreover, our approach learns the nature of ambiguous nodes from the data, making it suitable for more advanced anonymization techniques such as *k-Degree* anonymization [20].

We investigate a form of relational entity disambiguation in which the entities have been assigned an anonymized (possibly ambiguous) identifier, and network structure is the only information available. A node in the network represents the identifier of one or several entities, and an edge a relation between identifiers, such as co-occurrence in articles. We define an *ambiguous node*, in such a network, to be one representing several underlying entities. Such entities are assumed to have been assigned the same identifier in the data mining process, possibly because of sharing the same name.

A node representing only one entity is called *unambiguous*. We target a scenario where the original data is inaccessible or expensive to access. Such situations occur for example when large amounts of texts are parsed in an online fashion, not to be looked at again.

We develop a novel formulation of *relational entity disambiguation* as a graph classification problem of detecting ambiguities in anonymized data with network structure. We solve the problem leveraging kernel methods and a highly scalable SVM implementation. Given a graph with some nodes labeled as ambiguous or unambiguous, our method trains a classifier based on graph kernels using the neighborhoods of labeled nodes as input. The method is designed to operate in a distributed scheme and is implemented in the distributed computation framework GraphLab[1], allowing for web-scale data to be processed. While scalable entity disambiguation has been addressed previously [7], existing methods rely on entity attribute information, and do not lend themselves to the anonymized setting.

Further, we extend two existing graph kernels, designing a domain-specific adaptation of the shortest-path kernel [8], and a fast approximate algorithm for the direct product kernel [17] for unlabeled graphs. We show theoretical and empirical computational benefits of the extension compared to the original kernels. We evaluate our method on two real-world datasets, comparing the performance of different graph kernels. Our extensions are shown to be both significantly faster and more accurate than the original kernels. We also show that implementing our methods in GraphLab gives a significant speedup in our experiments.

We compare our approach to a state-of-the-art method [23] at the task of detecting anonymized, ambiguous nodes on a well-studied, public dataset. Our experiments show that our method is significantly better than the state-of-the-art, either in terms of speed or accuracy or both. Moreover, our method demands less information since it does not require knowledge of data sources.

The main contributions of this paper can be summarized as follows:

- We develop a novel formulation of *relational entity disambiguation* as a graph classification problem, suitable for anonymized data with a network structure.
- Our method leverages kernel methods and a scalable SVM algorithm. Further, our approach is designed to operate in a distributed fashion on GraphLab to ensure high scalability.
- We extend two existing graph kernels, showing theoretical computational benefits and improved classification accuracy on real-world data.
- We show that our method is significantly better than state-of-the-art in terms of either speed or accuracy or both.

The remainder of this paper is organized as follows. In Section 2, we survey related research. In Section 3, we define the problem of anonymized relational entity disambiguation, briefly covering necessary background, and present our approach. In Section 4, we make extensions to existing graph kernels for increased accuracy and scalability. Section 5 presents results of experiments on real-world data. In Section 6 we conclude and discuss future work.

[1] `http://graphlab.org/`

Notation.

We use lower-case bold letters $\mathbf{a} = [a_1, \ldots, a_n]^T$ to denote vectors and a_i denotes the i-th element of the vector. We use upper-case letters $A, B, C, \ldots$ to denote matrices with A_{ij} referring to the element at the i-th row and j-th column of A. Let A^n denote the n-th power of A. Let $\mathbf{a}^{(n)}$ denote the n-th vector in a set of vectors. The Euclidean norm of $\mathbf{a}$ is denoted by $\|\mathbf{a}\|$. Let $\mathbb{1}(\cdot)$ denote the indicator function and $\lfloor \cdot \rfloor$ the floor function. We use $\mathbf{e} = [1, \ldots, 1]^T$ to denote a vector of all 1's of the appropriate size. We let $\mathbf{e}_i = [0, \ldots, 1, \ldots, 0]^T$ denote a vector with the i-th element set to 1 and all other elements zero. We let $A \otimes B = C$ (of size $mp \times nq$) denote the Kronecker product [10] of matrices A (of size $m \times n$) and B (of size $p \times q$). We let $G = (V, E)$ denote a graph with vertex set $V = \{v_1, \ldots, v_n\}$ and edge set $E = \{e_k : e_k = (v_i, v_j) \Leftrightarrow v_i \sim v_j,\ v_i, v_i \in V\}$, $A(G)$ the adjacency matrix of G and $\Delta(G)$ the maximum degree of G. Let $V^{(i)} \times V^{(j)}$ denote the Cartesian product of vertex sets $V^{(i)}$ and $V^{(j)}$. We let $S_{ij}^{(G)}$ denote the shortest path from node i to j in graph G. $K(G^{(i)}, G^{(j)}) = \langle \phi(G^{(i)}), \phi(G^{(j)}) \rangle = K_{ij}$ denotes a kernel [26] on graphs $G^{(i)}$, $G^{(j)}$, where $\phi : \mathcal{X} \mapsto \mathcal{H}$ is a map into a Hilbert space $\mathcal{H}$ [26]. Whenever clear from context, we will omit G.

2. RELATED WORK

Entity Resolution.

A large family of techniques are devoted to *entity resolution*, the process in which references are matched with their underlying entities. This problem has two difficulties, 1) the same identifier may be used for several entities (e.g. Chris Anderson (TED) and Chris Anderson (WIRED)), and 2) the same entity may be referred to using several identifiers (e.g. Chris Anderson, Mr. Anderson). Bhattacharya and Getoor approach the problem using a probabilistic graphical model [6] and hierarchical clustering [5]. In both methods, similarity between identifiers is used, which makes them unusable in the anonymized setting. This notion can be generalized to include methods using any set of entity attributes in the resolution process [7, 11, 12, 13, 24, 15, 27]. In this paper, entities are anonymized and have no such attributes. Further, we approach only the first difficulty stated above, rendering the above methods unsuitable for direct comparison.

Relational Entity Disambiguation.

A specialized problem, related to entity resolution, is *relational entity disambiguation*. Here, various kinds of network structure between entities are exploited. Bekkerman and MacCallum [3] use link structure of personal web pages to disambiguate people in social networks. Bhattacharya and Getoor [5] use author lists of research papers, forming a network, and name similarity, to disambiguate authors. Both of these methods however, leverage information that is not available in our setting.

Malin [23] approaches the problem of disambiguating entities based on network structure alone. In his setting, entities are related through a set of sources. His canonical

example is that of entities being actors and sources being movies. Two actors are deemed connected if they appear in the same movie. Malin makes two attempts to solve the problem, one using hierarchical clustering, and one based on random walks. Since Malin's random walk method performed the best in his experiments, and that to the best of our knowledge no other methods are applicable to our problem, we use Malin's random walk method for a state-of-the-art comparison. We denote Malin's random walk method MALIN.

Each entity (actor) has an identifier (name) which may or not be ambiguous. For each identifier of interest, MALIN creates a graph in which the distinguished identifier is assigned one node (here called source node) for every source in which it appears, and every other identifier is assigned a single node. An edge is placed between two nodes if the underlying identifiers co-occur in a source, the weight of the edges corresponds to the number of times the identifiers have co-occurred. Then, a series of random walks are made from each of the source nodes for the identifier of interest, to estimate the probability of reaching another source node. These probabilities are then used to decide, using a simple threshold, whether two different source nodes refer to the same underlying entity. MALIN defines the similarity measure between two source nodes a and b, as the average of the probability of a random walk starting in a reaching b and a walk starting in b reaching a. If the similarity measure of two source nodes is higher than the threshold, then these source nodes are deemed to refer to the same entity.

MALIN allows for anonymized data, but demands knowledge of in which sources entities co-occur, e.g. a list of names for each movie in the dataset. This is a restriction on the type of datasets that can be used with the method since general networks do not have this type of source information. For example, most social networks have weighted relations connecting people, but no notion of sources. This means that MALIN uses more information than the method presented in this paper. In the experiments section we will see that this makes MALIN inapplicable to datasets where the sources are not observed. Further, the output of MALIN is a clustering of source nodes. The clusters represent the predicted underlying entities, associating each source node with only one entity. When using MALIN for comparison, we label any identifier, in which all source nodes become part of a single cluster, as *unambiguous* and identifiers where the source nodes are part of more than one cluster as *ambiguous*.

Graph Classification.

This work attempts to classify nodes in a network as ambiguous or unambiguous. Building a classifier on nodes demands for a way of comparing them. While there are kernels for straight-forward node comparison in graphs, such as the diffusion kernel [19], these are defined on the full entity graph and consequently prohibitively computationally expensive for real-world graphs. Instead, we may compare the neighborhood structure of nodes, resulting in a graph classification (rather than node classification) problem.

Common graph kernels include shortest-path kernels [8], direct product kernels [17] and graphlet kernels [31]. We evaluate the performance of all of them for the entity disambiguation task in our experiments. Moreover, we make extensions to the shortest-path and direct product kernels and evaluate them in terms of computation speed and classification accuracy. While graph kernels have been used for e.g. protein structure prediction [9] or character recognition [2], to the best of our knowledge, no existing work uses this approach for the problem of relational entity disambiguation.

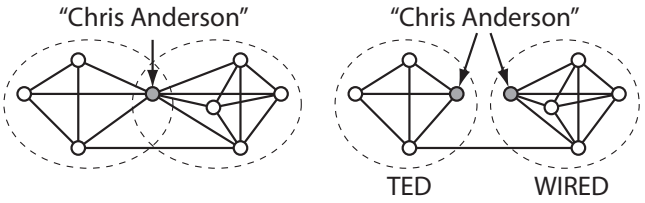

Figure 1: (Left) Local neighborhood (fictive) of the ambiguous vertex Chris Anderson. (Right) Correct splitting of the vertex into its two true underlying entities.

For completeness, we note that several other graph kernels exist such as Weisfeiler-Lehman [30] kernels and Fast Subtree kernels [29]. We limit our scope, however to the three kernels mentioned above.

3. OUR APPROACH

We let the term *entity* refer to a person or a company etc. while an *identifier* is a name or a label. If several entities have the same identifier, we say that the identifier is *ambiguous*. While a single entity may have several identifiers, we do not address this here; we focus only on ambiguities. In our setting, entities have hidden relationships which are observed through co-occuring identifiers such as names mentioned in news articles. We let an *identifier graph* be the graph with one node for each identifier and an edge between every pair of co-occuring identifiers. Edges are weighted by the significance of the relationships, such as number of co-occurrences.

This setting leads us to the definition of anonymized relational entity disambiguation as the following classification problem.

PROBLEM DEFINITION 1. *Given an undirected identifier graph $G = (V, E)$ with edge weights $w_{ij} \in \mathbb{R}^+$ and training data $S = \{(v_i, y_i) : 1 \leq i \leq m, v_i \in V, y_i \in \{\pm 1\}\}$ that labels certain nodes as ambiguous $(+)$ or unambiguous $(-)$, classify new nodes as $+1$ or -1. Each node of G may refer to a single entity or several underlying entities. The weight of an edge signifies the importance of the connection between two nodes.*

Our problem is illustrated by the example of Chris Anderson, stated in the introduction and further explained by Figure 1. In the figure, *two* individuals called Chris Anderson have been assigned only *one*, common identifier and thus *one* common node in the graph. This error creates a strong connection between the two communities, TED and WIRED, something we would not expect in reality. It is exactly this type of unexpected property of the network that we aim to capture with our classifier. Although this example involves only people, we would like to emphasize that nodes can represent any type of entity; an equally troublesome example would be that of the two *cities* Paris, France and Paris, Texas.

Algorithm 1 DETECTAMB.NODES($G = (V, E), \kappa$)

Input: $G = (V, E)$
Input: κ - Neighborhood size.
for $v_i \in V$ **do**
  Set $G^{(i)} = \mathcal{N}_\kappa^{(i)}$ according to (1)
end for
Compute graph kernel matrix K_{ij}, $\forall G^{(i)}, G^{(j)}$
Train an SVM with K and labels $\{y_i : y_i \in \{\pm 1\}\}$.
Output: SVM classifier.

Method Overview.

We approach the problem of identifying ambiguous nodes with the intuition that the neighborhoods of ambiguous nodes have structure different from those of unambiguous nodes. While Figure 1 is only an example, it illustrates the concept of two communities having fused around an ambiguous node. This motivates us to build a classifier using features of the neighborhood structure. In a graph $G = (V, E)$, we define the neighborhood $\mathcal{N}_G^{(i)}$ of a node $v_i \in V$ to be the subgraph of G induced by set of nodes connected to v_i through an edge. This notion can easily be extended to larger neighborhoods by considering neighbors of neighbors. In general, we define the κ-neighborhood $\mathcal{N}_{G,\kappa}^{(i)}$ of $v_i \in V$ as the subgraph induced by the set of nodes $\{v_j\} \subseteq V$ for which there exists a path from v_i to v_j with $s(v_i, v_j) \leq \kappa$ edges. Leaving out the subscript G for convenience, we have,

$$\begin{aligned} V_\kappa^{(i)} &= \{v_i\} \cup \{v_j \in V : s(v_i, v_j) \leq \kappa\} \\ E_\kappa^{(i)} &= \{(v_p, v_q) : (v_p, v_q) \in E \wedge v_p, v_q \in V_\kappa^{(i)}\} \\ \mathcal{N}_\kappa^{(i)} &= (V_\kappa^{(i)}, E_\kappa^{(i)}) \end{aligned} \quad (1)$$

With $\kappa = 1$ we recover the common neighborhood consisting of the distinguished node and its immediate neighbors.

Our method consists of three steps, illustrated in Figure 2. First, an identifier graph $G = (V, E)$ is constructed from the raw data. Each node represents one identifier that may correspond to one or several underlying entities. Some of the nodes are labeled + (ambiguous) or − (unambiguous) respectively. An edge between two nodes is present if the two corresponding identifiers are related in some way, and the edge weight represents the strength of the relation. For example, two identifiers may be related by co-occurring in the same article. The number of such co-occurrences is the weight of the edge. Note that we do not assume to have access to information about in *which* articles two identifiers co-occurred, only that they co-occurred in *some* articles. In the second stage, we extract the local neighborhoods of the nodes that we want to classify. For these local neighborhoods we then compute one of several graph kernels K in order to measure similarity between nodes. New nodes are labeled using a standard classifier. The overall approach is summarized in Algorithm 1.

Next, we describe a selection of graph kernels that are applicable to our approach.

3.1 Graph Kernels

Direct Product Kernel.

The direct product (DP) kernel compares two graphs based on the number of walks they have in common [17]. The common walks are computed by creating the *direct product graph* [17] $G_\times = (V_\times, E_\times)$ of two graphs $G^{(1)} = (V^{(1)}, E^{(1)})$ and $G^{(2)} = (V^{(2)}, E^{(2)})$, defined for unlabeled[2] graphs [8, 17] as:

$$V_\times = \{(v_1, w_1) \in V^{(1)} \times V^{(2)}\} \quad (2)$$

$$E_\times = \{((v_1, w_1), (v_2, w_2)) \in V_\times \times V_\times : (v_1, v_2) \in E^{(1)} \wedge (w_1, w_2) \in E^{(2)}\} \quad (3)$$

The kernel value is computed using

$$K_{DP}(G^{(1)}, G^{(2)}) = \sum_{i,j=1}^{|V_\times|} \left[\sum_{n=0}^{\infty} \lambda^n A(G_\times)^n\right]_{ij} \quad (4)$$

where $\lambda < \Delta(G_\times)^{-1} \in \mathbb{R}^+$ is a decay factor for making the sum converge.

Graphlet Kernel.

The graphlet (GL) kernel compares graphs through their distributions of *graphlets* of size $3 - 5$ [31], i.e. induced subgraphs of $k \in \{3, 4, 5\}$ nodes. As exhaustive enumeration of all graphlets is infeasible, the graphlet distribution is approximated using sampling, and by considering a finite number of values of k. Shervashidze et al. [31] show that for $\delta > 0$, $\epsilon > 0$,

$$n = \left\lceil \frac{2(\log 2 \cdot a + \log(\frac{1}{\delta}))}{\epsilon^2} \right\rceil \quad (5)$$

samples suffice to ensure that $P\{\|D - \hat{D}^n\|_1 \geq \epsilon\} \leq \delta$, where D is the true distribution and $\hat{D}^n$ is the approximate distribution using n samples.

Shortest Path Kernel.

The shortest path (SP) kernel compares graphs based on the similarity of their shortest paths [8]. All the shortest paths in a graph can be obtained in $\mathcal{O}(n^3)$ time using the Floyd-Warshall algorithm [16]. We let S_{ij}^G denote the shortest *distance* between nodes v_i and v_j in graph G. For unweighted graphs, the shortest distance between v_i and v_j is the number of steps on the shortest $v_i \to v_j$ path, and for weighted graphs it is the sum of all weights on the minimum weight $v_i \to v_j$ path. The shortest-path kernel is defined as:

$$K_{SP}(G^{(1)}, G^{(2)}) = \sum_{i,j,p,q} k_{walk}^{(1)}(S_{ij}^{G^{(1)}}, S_{pq}^{G^{(2)}}) \quad (6)$$

where $k_{walk}^{(1)}$ is a positive definite kernel on edge walks of length 1.

4. KERNEL EXTENSIONS

In this section, we design simple extensions of the DP and SP kernels, which we show can be computed quickly by explicitly knowing the kernel mapping ϕ.

4.1 Truncated Direct Product Kernel

For unlabeled graphs, the DP kernel (4) can be decomposed into components that can be calculated independently for each graph. For our proof, we need first to note that [32]

$$A(G_\times) = A(G^{(1)}) \otimes A(G^{(2)}) \quad (7)$$

[2]For labeled graphs, see Gärtner et al. [17]

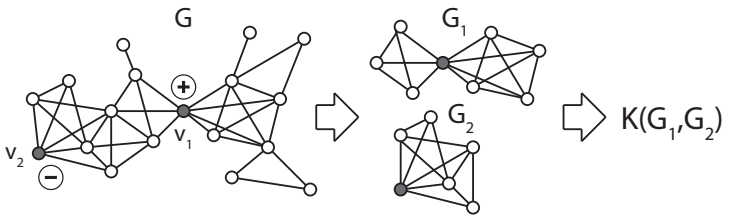

Figure 2: Method overview. In a three-step process, our approach considers a partially labeled co-occurrence graph (left), extracts local neighborhoods of nodes of interest (middle) and computes the kernel values of these nodes (right). The kernel values are then used for classification. In the figure, only labeled nodes are selected, as would be the case when training the classifier.

Algorithm 2 $\mathbf{u} = \textsc{NumRandWalk}(G = (V, E), K)$

Input: $G = (V, E)$
Input: K - Maximum walk length.
Initialize $t_i^{(0)} = 1,\ \forall i : v_i \in V$; $u_0 = |V|$
for $n \in \{1, \ldots, K\}$ **do**
  Set $t_i^{(n)} = \sum_{j:(v_j,v_i)\in E} t_j^{(n-1)},\ \forall i : v_i \in V$
  Set $u_n = \sum_{i:v_i\in V} t_i^{(n)},\ \forall i : v_i \in V$
end for
Output: $\mathbf{u}$ - Random walk counts.

Kronecker product powers have the property that [10](p.775)

$$(A \otimes B)^n = A^n \otimes B^n \tag{8}$$

From the definition of the Kronecker product we get

$$\sum_{i,j} [A \otimes B]_{ij} = \mathbf{e}^T A \mathbf{e}\mathbf{e}^T B \mathbf{e} \tag{9}$$

Now, we can derive a useful representation of the DP kernel as follows:

$$\begin{aligned} K_{DP}(G^{(1)}, G^{(2)}) &= \sum_{i,j=1}^{|V_\times|} \left[\sum_{n=0}^{\infty} \lambda^n A(G_\times)^n\right]_{ij} \\ &= \sum_{n=0}^{\infty} (\lambda^{\frac{n}{2}} u_n^{(1)})(\lambda^{\frac{n}{2}} u_n^{(2)}) \end{aligned} \tag{10}$$

where $u_n^{(i)} = \mathbf{e}^T A(G^{(i)})^n \mathbf{e}$. The last expression of (10) can be written as an inner product $\langle \phi(G^{(1)}), \phi(G^{(2)}) \rangle$, making it a valid kernel [26]. We approximate (10) by only considering walks up to a finite length K, setting the kernel value of all longer paths to zero. This makes the dimension of the feature vectors ϕ finite, and trivially, the kernel is still valid. We call this the *truncated direct product* (TDP) kernel. For clarity, the kernel is defined as:

$$K_{TDP}(G^{(1)}, G^{(2)}) = \sum_{n=0}^{K} (\lambda^{\frac{n}{2}} u_n^{(1)})(\lambda^{\frac{n}{2}} u_n^{(2)}) \tag{11}$$

The vector $\mathbf{u}^{(i)} = [u_0^{(i)}, \ldots, u_K^{(i)}]^T$ for graph $G^{(i)}$ can be calculated using Algorithm 2. Note that making the sum in (10) finite allows us to set $\lambda > \Delta(G_\times)^{-1}$, making the TDP kernel defined where the DP kernel is not. As the average degree of a graph is $2\frac{|E|}{|V|}$, the time complexity for calculating the TDP kernel becomes $\mathcal{O}(K|E|)$.

For general graphs, exact approaches take $\mathcal{O}(|V|^6)$ [17] and $\mathcal{O}(|V|^3)$ [32] time to calculate the DP kernel for exact solutions. If the graph is sparse, i.e. has $\mathcal{O}(|V|)$ edges, then the calculation can be done in $\mathcal{O}(|V|^2)$ [32] time. For an approximate solution in the general case, the kernel can be calculated in $\mathcal{O}(|V|^2)$ [18] time. Therefore, our approach is advantageous if $K \ll |V|$ and the graph is sparse.

Long walks (large K) tend to result in a phenomenon known as *tottering* in which the walks of the DP kernel will go back and forth along the same nodes, over and over. Tottering reduces the expressivity of the kernel as the same cycles of nodes will be visited repeatedly [22]. This suggests that for good generalization performance, as suggested empirically by Borgwardt and Kriegel [8], K should be small.

Distinguished Vertex.

The TDP kernel can be modified further to suit to our specific classification problem. We note that the graphs we are comparing are in fact pointed graphs, in that they are the neighborhood of one distinguished vertex. Instead of counting the number of random walks from all vertices, we can choose to only count walks from the distinguished vertex in the middle of the local neighborhood graph (i.e. the vertex "Chris Anderson" in Figure 1). This enables us to collect more specific information concerning only the local neighborhood of the vertex of interest. Thus, we create a distinguished vertex modification of the TDP kernel as follows:

$$K_{TDP_d}(G^{(1)}, G^{(2)}) = \sum_{n=0}^{K} (\lambda^{\frac{n}{2}} u_n^{(1)})(\lambda^{\frac{n}{2}} u_n^{(2)}) \tag{12}$$

where $u_n^{(i)} = \mathbf{e}_d^T A(G^{(i)})^n \mathbf{e}$ if the distinguished vertex is that of index d. Essentially, only the definition of the vector $\mathbf{u}$ has changed. Algorithm 2 can be easily modified to account for walks only from the distinguished vertex by setting $t_i^{(0)} = 1$ only for the distinguished vertex in the initialization step of the algorithm, and 0 for all other vertices.

We make a note that this definition is a special case of the alternative definition of the DP kernel in Vishwanathan et al. [32]. They incorporate starting and stopping probabilities $p_\times$ and $q_\times$ for the random walks, giving the following graph kernel:

$$K(G^{(1)}, G^{(2)}) = \sum_{n=0}^{\infty} \mu(n) q_\times^T W_\times^n p_\times \tag{13}$$

where μ is a discrete measure and $W_\times$ is the weight matrix associated with $A(G_\times)$. By ignoring edge weights and setting $\mu(n) = \lambda^n$, $p_\times = \mathbf{e}$ and $q_\times = \mathbf{e}_d$, we recover (12) with $K = \infty$.

4.2 Binned Shortest Distance Kernel

We seek to construct a computationally efficient version of the SP kernel for weighted graphs. Consider using the indicator function for $k^{(1)}_{walk}$ in the shortest-path kernel as in Borgwardt and Kriegel [8]. Inserting this into (6) leaves us with,

$$K_{SPI}(G^{(1)}, G^{(2)}) = \sum_{i,j,p,q} \mathbb{1}\left[S^{G^{(1)}}_{ij} = S^{G^{(2)}}_{pq}\right] \quad (14)$$

For unweighted or integer weighted graphs, S_{ij} are integers, making the indicator function a reasonable choice of kernel. However, for real weighted graphs, this formulation is less sensible as it involves comparing real numbers for equality. Therefore, we design a simple heuristic extension of (14) for general weighted graphs with the idea of comparing rounded-off values of the real-valued weights. Formally, we define a function $h : \mathbb{R}^+ \mapsto \{1, 2, \ldots, M\}$ that maps distance values to a finite set of M bins. This gives us the definition of the *binned shortest distance* (BSD) kernel as:

$$K_{BSD}(G^{(1)}, G^{(2)}) = \sum_{k=1}^{M} \sum_{(i,j)\in E^{(1)}} \theta^{G^{(1)}}_{ijk} \sum_{(p,q)\in E^{(2)}} \theta^{G^{(2)}}_{pqk} \quad (15)$$

where $\theta^G_{ijk} = \mathbb{1}(h(S^G_{i,j}) = k)$. This definition can be thought of as a relaxation of (14) in which the indicator has value 1 if the compared values are similar enough. Equation (15) can easily be written as the inner product $\langle \phi(G^{(1)}, \phi(G^{(2)}) \rangle$, showing that it is a valid kernel [26].

Binning.

By applying different types of binning (choosing the function h), the kernel can be made to consider similar distances to be equal, as opposed to the indicator version of the SP kernel (14). We believe this is advantageous in weighted graphs, as we will show empirically in our experiments. A simple binning function is the linear binning defined by

$$h(S_{ij}) = \left\lfloor \frac{MS_{ij}}{S_{max} + \epsilon} \right\rfloor + 1 \quad (16)$$

where $S_{max} = \max_{G,i,j} S^G_{ij}$ is the maximum shortest distance S^G_{ij} encountered in the dataset and $\epsilon \in \mathbb{R}^+$ is a small constant.

4.3 Fast Computation

Since the kernel mappings ϕ are explicitly known for the TDP and BSD kernels, we avoid expensive computation of the kernel values K_{ij} for each *pair* of graphs. In fact, we do not need to compute the Gram matrix K at all, since explicitly knowing ϕ allows us to use a fast linear SVM solver, making our approach in Section 3 usable in $\mathcal{O}(mT)$ time. This procedure is shown in Algorithm 3. We use Pegasos [28], a state-of-the-art iterative subgradient method for training the SVM classifier. Pegasos has the property of being independent of the training set size m when using linear kernels, which is to our advantage, as previously described. Note that this independence only holds when ϕ is known, as the Pegasos algorithm becomes directly dependent on m when using kernels in the general case [28].

We make the computation of ϕ scalable by using GraphLab [21], a library for doing distributed computation. By following the restrictions of the GraphLab framework, scalable computation on graphs can be easily achieved. Since we explicitly know the mapping ϕ, we can easily design algorithms that fit the GraphLab framework for computing ϕ for each graph in an efficient manner.

Algorithm 3 DETECTAMB.NODESFAST($G = (V, E), \kappa$)

Input: $G = (V, E)$
Input: κ - Neighborhood size.
for $v_i \in V$ **do**
  Set $G^{(i)} = \mathcal{N}^{(i)}_\kappa$ according to (1)
end for
Compute feature mappings $\phi(G^{(i)})$, $\forall G^{(i)}$
Train a linear SVM with labels $\{y_i : y_i \in \{\pm 1\}\}$.
Output: SVM classifier.

4.4 Normalization

We can make the TDP and BSD kernels less sensitive to the graph size by normalizing the feature vectors, giving the kernels the form,

$$K_{ij} = \frac{\phi(G^{(i)})\phi(G^{(j)})}{\| \phi(G^{(i)})\| \| \phi(G^{(j)})\|} \quad (17)$$

This definition of the TDP and BSD kernels is equivalent to the cosine similarity measure [25]. We show in our experiments that normalization indeed helps increase classification performance.

5. EXPERIMENTS

We evaluate our approach by comparing it, using a large selection of graph kernels, to the approach of Malin [23], denoted MALIN. We address two questions, 1) how well have our extensions improved existing graph kernels, 2) how well does our method fair against a state-of-the-art method. The experiments are conducted on two real-world datasets, one of which is readily available.

5.1 Recorded Future News Data

We investigate a proprietary dataset (RF) from Recorded Future[3], a company specializing in web intelligence and predictive analytics. The data has been gathered using automatic processing of articles from news and social media. When an entity, e.g. a person, place or company, was found in an article, it was assigned a (possibly ambiguous) canonical identifier.

The RF dataset contains information from around 10 million articles. The identifier graph of the set has a node for each identifier in the set of articles and an edge (v_i, v_j) if identifiers v_i and v_j have co-occurred in an article. Each edge is associated with a weight w_{ij} equal to the number of times i and j have co-occurred. The graph contains 2,155,893 nodes and 13,935,815 edges and has 91 nodes that have been manually labeled as either ambiguous (+1) or unambiguous (−1). The average size for each local neighborhood graph of the labeled nodes was 267 nodes and 5,830 edges. 39.6% of the labeled nodes have a positive label. Note that in this dataset we do not have access to the actual articles that is the cause of the relations, only the fact that certain names co-occurred in a number of articles, making Malin's random walk method inapplicable to this dataset.

[3] `http://www.recordedfuture.com/`

5.2 Internet Movie Database

The Internet Movie Database[4] (IMDb) is an online database gathering information about the televised entertainment industry, including movies, actors and production personnel.

Following Malin [23], we artificially introduced ambiguities by treating all actors with the same last name as one, merging them into a single node corresponding to a single identifier e.g. "Smith". An edge was added between two nodes if the corresponding identifiers were part of the same cast list in a movie. Edges were weighted by the total number of times a pair of nodes starred in movies together. Additionally, we only included movies with more than one actor, thus ensuring that every node is adjacent to at least one other node. Actors who had not starred together with any other actor were removed from the dataset. Just like in Malin [23], we only considered movies between 1994-2003. It is important to note that our dataset is not the exact same as the one used in Malin [23]. This is due to the fact that the IMDb database has been updated since. The dataset used in by Malin [23] had ∼37,000 movies and ∼180,000 distinct entities, while our dataset consists of 52,004 movies and 2,272,504 distinct entities.

When all actors with the same last name had been merged, the resulting graph contained 150,072 nodes, corresponding to the distinct last names, and 9,283,233 edges. In Malin [23] they reported ∼85,000 distinct last names. We created a training set by randomly selecting 100 identifiers, such that half of them were ambiguous. The average size for each local neighborhood graph of the labeled nodes was 156 nodes and 12,028 edges. We denote this dataset IMDB1.

In IMDB1 there exists several identifiers that appear in one movie only. In fact, out of the 100 identifiers chosen, 38 was part of only one source. Out of these 38 identifiers, 36 were non-ambiguous, while 2 were ambiguous. Consistent with intuition, it is very unlikely that an actor appearing in only one movie is ambiguous. This, perhaps undesired, property of the dataset led us to create a second set based on the IMDb data. This dataset contains the same data, the only difference being that the 100 identifiers, chosen for evaluation, are required to be a part of more than one movie. In the resulting set, the average number of nodes and edges in each local neighborhood were 279 and 24,550 respectively. We denote this alternative dataset IMDB2.

5.3 Experimental Setup

We evaluate the performance of five different graph kernels (GL, DP, SP, TDP, BSD), including extensions, within our approach using 10-fold cross validation with the Pegasos SVM solver [28]. For all kernels we use only the 1-neighborhood ($\kappa = 1$) for comparing nodes. This reduces computation time, but is also motivated from the intuition that the larger the neighborhood included, the less specific it is to the node of interest.

We transform all edge weights w_{ij} with a function $\tau : \mathbb{R}^+ \mapsto \mathbb{R}^+, \tau' < 0$ so that strong connections (many co-occurrences) become small weights, in the belief that shortest paths should go along strong connections. In our experiments, we try logarithmic and inverse edge transforms, defined as: $\tau_{\log}(w_{ij}) = \ln w_{max} - \ln w_{ij}$ and $\tau_{\text{inv}}(w_{ij}) = w_{ij}^{-1}$, where w_{max} is the maximum edge weight in the graph. Note that the transformations are applied prior to the computation of the distances S_{ij}^G with the Floyd-Warshall algorithm. The transforms only affect the BSD and SP kernels, as the other kernels do not consider edge weights. For the BSD kernel, we use only linear binning, as defined in (16).

We believe that using linear binning is sufficient, since the edge transform τ already preprocesses the edges in a non-linear manner. The logarithmic edge transform allows our kernel to capture the intuition that there exist many weak connections and few very strong connections. Achieving good binning resolution in both cases motivates the use of the logarithm. Figure 3, depicting the weight distribution of the two datasets, supports this intuition.

For the TDP kernel, we perform a grid search over the parameters, setting $\lambda \in \{0.2, 0.4, 0.6, 0.8\}$ and $K = \lfloor \ln_\lambda 10^{-5} \rfloor$, giving $K \in \{7, 12, 22, 51\}$. For the BSD kernel, we try setting the number of bins $M \in \{5, 10, 25, 50\}$. As both the above kernels can be normalized, we try enabling and disabling normalization for these. For the GL kernel, we used $\epsilon = \delta = 0.05$. The parameter λ of the DP kernel was set for each dataset so that $\lambda < \Delta(G_\times)^{-1}$, giving $\lambda = 0.00067$ for RF and $\lambda = 0.00098$ for IMDB1. The DP kernel on IMDB2 did not finish within 48 hours and was left out because of this. For all kernels, we optimize the Pegasos SVM parameters using cross-validation.

For MALIN, we started 100 and 1000 random walks from each source of the node being classified, and for each walk we did 50 steps. Early tests showed that increasing the number of steps did not increase performance. Note that in Malin [23], only 100 walks were used, albeit on a different dataset. On IMDB1 and IMDB2, however, more walks could be needed, as they are of larger size. We then tested for the best threshold from the set $\{0.00, 0.01, \ldots, 1.00\}$.

Kernel	RF	IMDB1	IMDB2
GL	81.0	84.0	**71.0**
SP	73.0	77.0	60.0
BSD_n	**82.0**	82.0	61.0
BSD	78.0	78.0	61.0
DP	54.9	37.0	-
TDP_n	75.0	74.0	62.0
TDP	75.0	65.0	62.0
TDP_{dn}	76.0	77.0	65.0
TDP_d	72.0	76.0	55.0
MALIN		**86.0**	64.0

Table 1: Classification accuracy (%) on 10-fold cross-validation. The subscript n stands for normalization and d for the distinguished vertex version of the TDP kernel. The DP kernel on IMDB2 did not finish within in 48 hours and was left out. Note that MALIN is not applicable to the RF dataset as explained in Section 5.1.

5.4 Results

The best results of our experiments, in terms of classification accuracy on the three datasets, are shown in Table 1. The GL, SP and DP kernels are the original kernels as described in Section 3. The DP kernel was computed using the Sylvester equation method of Vishwanathan et al. [32] and the SP kernel used the indicator function as in (14). The BSD and TDP kernels are the extensions of the SP and DP kernels respectively, proposed in Section 4. In Table 2, the

[4] `http://www.imdb.com/interfaces`

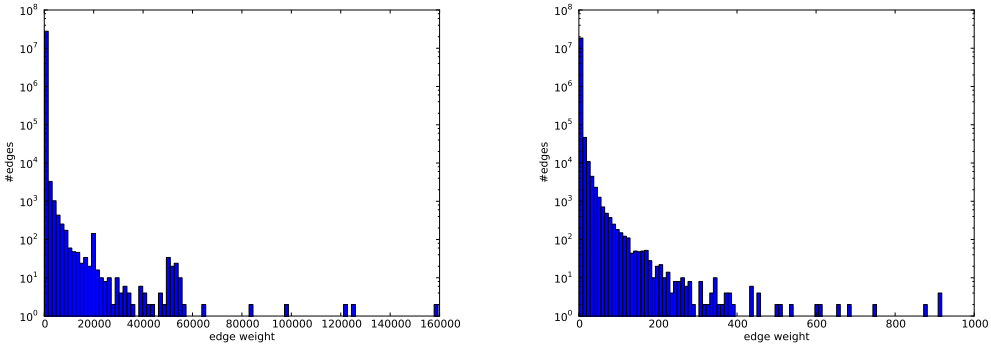

Figure 3: Histogram of edge weights in the RF (left) and IMDB (right) datasets, with the number of edges on the y-axis and edge weight magnitude on the x-axis. Note that the y-axis is on a logarithmic scale.

Kernel†	RF	IMDB1	IMDB2
GL	424	1254	1221
SP	1010	5247	14151
BSD_n	1721	6706	19325
BSD	1690	8274	12143
DP	14980	131329	-
TDP_n	3	9	25
TDP	3	9	24
TDP_{dn}	8	9	19
TDP_d	8	9	25
MALIN		2787	49362

Table 2: Wall clock time for detecting ambiguities for all training examples, measured in seconds[5]. The subscript n stands for normalization and d for the distinguished vertex version of the TDP kernel. † The GL kernel was implemented in MATLAB. All others were run in C++ using GraphLab. The DP kernel on IMDB2 did not finish within in 48 hours and was left out. Note that MALIN is not applicable to the RF dataset as explained in Section 5.1.

wall-clock times needed for detecting ambiguities using the various methods are presented.

Preliminary results indicated that the performance did not improve with $\kappa > 1$. Further experiments with $\kappa > 1$ were not performed because of this fact and the intuition that the larger the size of the extracted neighborhoods, the less they will describe the specific characteristics of the central vertex. When using larger neighborhoods, the graph sizes increase rapidly, reaching close to the size of the original identifier graph already for $\kappa = 2$. It was thus infeasible to run these experiments within reasonable time.

We proceed to compare the results of our method with MALIN based on the results in Table 1. In the RF dataset, there is no source information available, making it is impossible to run experiments for MALIN on this dataset.

We now summarize the parameters giving the best performance during experiments. The TDP kernel performed best with $\lambda = 0.2$ for the RF dataset, $\lambda = 0.8$ on the IMDB1 dataset and $\lambda = 0.6$ on the IMDB2 dataset. The SP kernel performed best with inverse edge transform on the RF dataset and logarithmic edge transform on the IMDB1 and IMDB2 datasets. The BSD kernel with logarithmic edge transform on the RF dataset gave the best result, using 10 linear bins. On the IMDB1 and IMDB2 datasets, the BSD kernel with logarithmic edge transform and 25 and 5 linear bins, respectively, gave the best result.

For MALIN, on IMDB1, the best number of walks was 100, and the best threshold 0.00. On IMDB2, MALIN got the best accuracy with 1000 walks and the threshold being 0.17.

Our approach (GL, BSD) gets comparable results to MALIN on IMDB1, while being significantly faster (GL), see Table 2. Our approach outperforms MALIN by a wide margin on IMDB2. IMDB1 contains identifiers that only appeared in one movie, which in our setting means that we have very little information to base the classification on. Also, MALIN predicts all single-source nodes to be non-ambiguous by default. Since most of these nodes actually are non-ambiguous, this means that MALIN works quite well on the IMDB1 dataset. In fact, any classifier that has access to the sources would be able to get a very good result on IMDB1 by classifying all nodes with more than one source as ambiguous and the rest as non-ambiguous. Our method however does not have access to the sources and still gives similar performance, with many of our kernels taking significantly smaller amount of time. This simple classification method, for classifiers that have access to the sources, is however not very useful on the more challenging dataset IMDB2, and thus MALIN gets a worse result on that dataset. We also note that MALIN takes significantly longer time to run on the IMDB2 dataset, compared to IMDB1. This is because of the fact that IMDB2 contains more sources per node on average, increasing the number of walks performed by MALIN. Also, MALIN got the best result with 1000 walks per source node, instead of 100 like on IMDB1, this also increased the running time significantly. With 100 walks MALIN got only 56% accuracy and took 3,957 seconds to run.

In our graph kernel-based approach, the time complexity for training the classifier is independent of the number of sources the data contained. Although the computation time will increase if the number of edges increases, increasing the number of sources will not increase the computation time unless it introduces new edges into the graph. New edges will be added if a source introduces a new pair of identifiers not

[5]8 threads on a cluster with 5 computation nodes and 8 Intel Xeon 2.4 GHz cores per node was used.

previously encountered together in a source. New sources without any new pairs of identifiers will then only increase the weight of edges and therefore not affect the computation time of our method. This behavior is in contrast to MALIN, which increases linearly in computation time with the number of sources. Since the number of sources potentially can be very big in real world datasets, this time dependence on the number of sources could in some cases be a problem. If we assume a scenario where we have access to a very large amount of sources, then as long as a new source does not introduce a new edge, including that source in the dataset will not increase the computation time of our graph kernel approach, giving us an advantage over MALIN in this setting. When it comes to running time, the GL and TDP kernels finished faster than MALIN on both datasets, with the TDP kernel being particularly noteworthy.

On a final note, given the speed of our method, it could be used as a potential preprocessing step, whereafter a clustering method such as MALIN could be run on the filtered smaller selection of nodes, dividing them into their correct underlying entities. This, of course, assumes a scenario where the sources are observed. It would not work on the RF dataset, for which further investigation needs to be done.

Evaluation of Kernel Extensions.

In terms of accuracy, the BSD and GL kernels outperform the other kernels when it comes to the RF dataset. On the IMDB1 dataset GL performs the best of the kernels, with the normalized BSD kernel close behind. On the more challenging dataset IMDB2, the GL kernel performed the best.

We note that both of our extended kernels outperformed their original counterparts. For the DP kernel, this is especially noteworthy due to the timing results of Table 2. The TDP kernel variations all perform significantly better to a fraction of the computational cost. A possible reason for why the TDP kernel would perform better than the DP kernel is *tottering*, as described in Section 4. Since calculating the DP kernel is the same as calculating all walks of length up to infinity, tottering actually outweighs the relevant walks, making the DP kernel perform worse than a kernel that only calculates walks up to a certain length, i.e. the TDP kernel.

Using normalization helped the BSD and TDP kernels get better or equal accuracy on all datasets, indicating that the normalization might help the kernels become less sensitive to graph size. We also note that the BSD kernel outperforms the SP kernel on both datasets, indicating that binning seems to help the kernel get a better classification accuracy on weighted graphs.

The experiments with the distinguished node for the TDP kernel (TDP_{dn}), gave a slight increase in performance, but not enough to say that the TDP_{dn} kernel is better than the TDP_{n} kernel.

The experiments indicate that the edge transform $\tau_{\log}$ generally performs better than τ_{inv}. The only exceptions to this was the BSD kernel on the IMDB2 dataset and the SP kernel on the RF dataset.

Speedup.

As the BSD and TDP kernels can be calculated algorithmically an order of magnitude faster than naïve pairwise kernel computation, it is uninteresting to compare this increase in computation speed. Indeed, the TDP kernel runs much faster than any of the other approaches in our experiments. However, with a running time of a few seconds, investigating the scalability of the TDP kernel would require a larger dataset. Rather, we focus on investigating how well the computation of shortest distances for the BSD kernel scales when using GraphLab. The experiments were carried out on the RF dataset. A cluster with 5 computation nodes with 8 Intel Xeon 2.4 GHz cores and 23.5 GB RAM per node was used. Figure 4 shows a comparison with linear speedup. Speedup was defined as $\frac{t_1}{t_k}$ where t_k is the time required to precompute the graph kernels using k CPUs. In the figure, *serial* means running GraphLab on each local neighborhood graph separately, whereas *batch* denotes putting all local neighborhoods in one big graph, on which GraphLab is run just once. *Spread* denotes running the threads evenly divided on several physical machines, whereas *tight* denotes running as many threads per machine as there are CPU cores. We see that the batch-tight computation mode gets the best speedup, increasing almost linearly up until 8 CPUs.

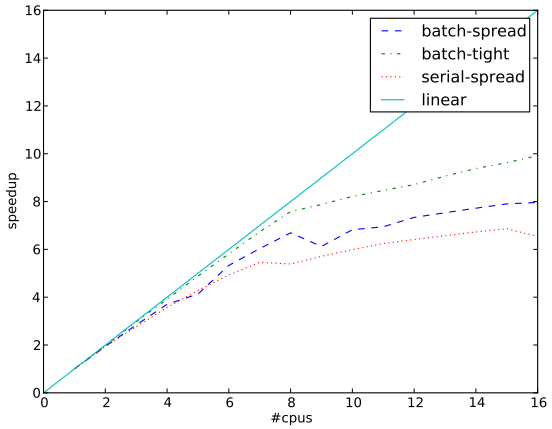

Figure 4: Speedup of BSD kernel precomputation as the number of CPUs increase on the RF dataset. Linear speedup is shown by the solid line.

6. CONCLUSIONS AND FUTURE WORK

In this paper we have developed a novel formulation of anonymized relational entity disambiguation as a graph classification problem. We have devised a method for detecting ambiguities in graph data based on local neighborhoods using graph kernels. Our approach was compared to a state-of-the-art method, showing significantly better performance in terms of either accuracy or speed or both.

We have shown how to enable fast computation of the direct product (DP) kernel [17] by extending it to a truncated direct product (TDP) kernel that outperformed the DP kernel in our experiments. We have also presented a simple extension of the shortest-path kernel [8], creating the binned shortest distance (BSD) kernel, as a way of measuring similarity between general weighted graphs. The BSD kernel was shown to give good classification results, outperforming the base shortest-path kernel on weighted graphs, which motivates the need for binning. Normalization of the TDP and BSD kernels also helped boost performance. We

show a significant speedup in the computation of the kernels when using GraphLab and by explicitly knowing the kernel mapping ϕ.

Future research should examine the possibility of applying our method in a semi-supervised setting, by considering unlabeled samples. Other aspects we will look into is the problem of actually associating ambiguous identifiers with their underlying entities. It should be noted that our method can be used as a fast indicator of which identifiers to examine more closely for ambiguity. Larger datasets should be examined, in order to further investigate the scalability of our method. Given the good performance of the GL kernel in our experiments, it is of great interest to attempt at designing an extension to the GL kernel, which considers weighted graphs.

7. ACKNOWLEDGMENTS

We would like to thank Recorded Future, a software company, and their employees for their insight and help and for providing data. We would also like to thank Bhavishya Goel and Jacob Lidman for help with initial setup of the GraphLab framework.

8. REFERENCES

[1] C. C. Aggarwal and S. Y. Philip. *A condensation approach to privacy preserving data mining*. Springer, 2004.

[2] F. R. Bach. Graph kernels between point clouds. In *Proc. of ICML*, 2008.

[3] R. Bekkerman and A. McCallum. Disambiguating web appearances of people in a social network. In *Proc. of WWW*, 2005.

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

[5] I. Bhattacharya and L. Getoor. Entity resolutions in graphs. In *Mining Graph Data*. Wiley, 2006.

[6] I. Bhattacharya and L. Getoor. A latent dirichlet model for unsupervised entity resolution. In *Proc. of SDM*, 2006.

[7] C. Böhm, G. de Melo, F. Naumann, and G. Weikum. Linda: distributed web-of-data-scale entity matching. In *Proc. of CIKM*, 2012.

[8] K. M. Borgwardt and H.-P. Kriegel. Shortest-path kernels on graphs. In *Prof. of ICDM*, 2005.

[9] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. Vishwanathan, A. J. Smola, and H.-P. Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl 1):i47–i56, 2005.

[10] J. Brewer. Kronecker products and matrix calculus in system theory. *Circuits and Systems, IEEE Trans. on*, 25(9):772–781, 1978.

[11] R. Bunescu and M. Pasca. Using encyclopedic knowledge for named entity disambiguation. In *Proc. of EACL*, 2006.

[12] S. Cucerzan. Large-scale named entity disambiguation based on wikipedia data. In *Proc. of EMNLP-CoNLL*, 2007.

[13] C. P. Diehl, L. Getoor, and G. Namata. Name reference resolution in organizational email archives. In *Prof. of SDM*, 2006.

[14] C. Dwork. Differential privacy: A survey of results. In *Proc. of TAMC*. 2008.

[15] T. Elsayed, D. W. Oard, and G. Namata. Resolving personal names in email using context expansion. In *Proc. of ACL*, 2008.

[16] R. W. Floyd. Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345, 1962.

[17] T. Gärtner, P. Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. *Learning Theory and Kernel Machines*, pages 129–143, 2003.

[18] U. Kang, H. Tong, and J. Sun. Fast random walk graph kernel. In *Proc. of SDM*, 2012.

[19] R. I. Kondor and J. Lafferty. Diffusion kernels on graphs and other discrete structures. In *Proc. of ICML*, 2002.

[20] K. Liu, K. Das, T. Grandison, and H. Kargupta. Privacy-preserving data analysis on graphs and social networks. In H. Kargupta, J. Han, P. S. Yu, R. Motwani, and V. Kumar, editors, *Next Generation of Data Mining*. Chapman and Hall/CRC, 2008.

[21] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein. Distributed graphlab: a framework for machine learning and data mining in the cloud. *Proc. of VLDB*, 2012.

[22] P. Mahé, N. Ueda, T. Akutsu, J.-L. Perret, and J.-P. Vert. Extensions of marginalized graph kernels. In *Proc. of ICML*, 2004.

[23] B. Malin. Unsupervised name disambiguation via social network similarity. In *Workshop on link analysis, counterterrorism, and security*, volume 1401, pages 93–102, 2005.

[24] V. Rastogi, N. N. Dalvi, and M. N. Garofalakis. Large-scale collective entity matching. *PVLDB*, 4(4):208–218, 2011.

[25] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA, 1986.

[26] B. Schölkopf and A. J. Smola. *Learning with kernels: Support vector machines, regularization, optimization, and beyond*. MIT press, 2001.

[27] P. Sen. Collective context-aware topic models for entity disambiguation. In *Proc. of WWW*, 2012.

[28] S. Shalev-Shwartz, Y. Singer, N. Srebro, and A. Cotter. Pegasos: primal estimated sub-gradient solver for SVM. *Mathematical Programming*, 127(1):3–30, 2011.

[29] N. Shervashidze and K. M. Borgwardt. Fast subtree kernels on graphs. In Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, editors, *NIPS*, pages 1660–1668. Curran Associates, Inc., 2009.

[30] N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn, and K. M. Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12:2539–2561, 2011.

[31] N. Shervashidze, S. Vishwanathan, T. Petri, K. Mehlhorn, and K. M. Borgwardt. Efficient graphlet kernels for large graph comparison. In *Proc. of AISTATS*, 2009.

[32] S. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt. Graph kernels. *JMLR*, 2010.