Fast and Scalable Algorithms for Semi-supervised Link Prediction on Static and Dynamic Graphs

Rudy Raymond[1] and Hisashi Kashima[2]

[1] IBM Research – Tokyo
1623-14 Shimo-tsuruma, Yamato
Kanagawa 242-8502, Japan
raymond@jp.ibm.com
[2] Department of Mathematical Informatics
The University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8656, Japan
kashima@mist.i.u-tokyo.ac.jp

Abstract. Recent years have witnessed a widespread interest on methods using both link structure and node information for link prediction on graphs. One of the state-of-the-art methods is Link Propagation which is a new semi-supervised learning algorithm for link prediction on graphs based on the popularly-studied label propagation by exploiting information on similarities of links and nodes. Despite its efficiency and effectiveness compared to other methods, its applications were still limited due to the computational time and space constraints. In this paper, we propose fast and scalable algorithms for the Link Propagation by introducing efficient procedures to solve large linear equations that appear in the method. In particular, we show how to obtain a compact representation of the solution to the linear equations by using a non-trivial combination of techniques in linear algebra to construct algorithms that are also effective for link prediction on dynamic graphs. These enable us to apply the Link Propagation to large networks with more than 400,000 nodes. Experiments demonstrate that our approximation methods are scalable, fast, and their prediction qualities are comparably competitive.

1 Introduction

Many interactions in the real world can be expressed as networks that consist of a set of entities mapped to nodes and a set of links for the relationships between entities. Each entity may have additional information that influences its relationship with other entities. There are many natural examples of such networks, such as a network of webpages where the entities, relationships, and additional information are, respectively, the webpages, URL links, and the textual content of the pages. There are many other networks that exhibit similar properties; ranging from friendships and actions among people in Social Networking Service (SNS) to biological interactions among proteins. In many such networks the total

J.L. Balcázar et al. (Eds.): ECML PKDD 2010, Part III, LNAI 6323, pp. 131–147, 2010.

number of entities is large and the number of relationships per entity is often more than one.

Link mining is one of the most popular research topics that deals with extracting useful insights from such networks. It includes the *link prediction* problem, which is the problem of predicting the existence of unknown links between nodes using information from the known parts of the network. Link prediction has a broad range of applications. In marketing, users' ad-clicking actions might be predicted from the history of their actions and the relationships among their friends (or similar users) and the online ads. In social network analysis, users can be guided to contents of interest based on the friendship relations in their networks. Recent advances in storage technologies have made it possible to collect and store large amounts of information for link prediction. However, this also adds to the complexity of solving the link prediction problem.

Methods for the link prediction problem can be classified into two categories depending on the information used for prediction: link-information [13,14,19,28] and node-information [2,3,16] -based methods. Link-information-based methods such as matrix factorization [18] and link metrics [14] can predict the structures of networks from the observed parts of the networks. However, when only a small parts of the networks are known, the link-information-based methods work poorly. In contrast, node-information-based methods use node features as auxiliary information and can work well even in such cases.

Recently, a novel node-information-based link prediction method, which used a semi-supervised label propagation learning method to predict links (and hence, the method name *Link Propagation*) was proposed by the authors of [10]. Combined with information on node similarities, they devised a method for the Link Propagation by the conjugate gradient method and the *vec-tricks* techniques [24]. They also demonstrated that the qualities of the link prediction by the Link Propagation were competitive to those of other state-of-the-art methods. (See also Figure 1 in Section 3 for the comparison on the effectiveness of the exact Link Propagation by our method with existing state-of-the-art methods). However, like many other methods using node information, it has severe limitations in terms of computational time and space to handle networks with more than thousands of nodes, not to mention the difficulties in coping with link prediction on dynamic networks.

In this paper, we tackle these typical limitations of the node-information-based semi-supervised learning by proposing fast and scalable methods for the Link Propagation. Instead of using the conjugate gradient, our methods utilize the matrix factorization techniques, such as the Cholesky and eigen- decomposition. These enable us to exactly solve linear equations in the Link Propagation with less computational time and space. Moreover, the exact methods open the paths to define an approximate method for the Link Propagation that needs significantly less computational cost while maintaining accuracy. Our methods are also compatible with the vec-tricks techniques, which are useful to furtherly reduce the computational time. In addition, our methods can be applied for link prediction on dynamic networks. Recently, there is an emerging interest in the

study of dynamic network models (e.g., see [7]), and accordingly, there were many prior studies for link prediction and proximities on dynamic graphs, such as, in [9,21,22]. However, almost all of them are link-information based ones.

The contributions of this work are:

(1) We propose a new Link Propagation for developing a fast and scalable semi-supervised link prediction method based on the node information. The techniques used in the method also utilize matrix factorization and approximation. Thus, the new Link Propagation method can be considered as a kind of node-information-based link prediction that utilizes fast and scalable techniques in topological-based link prediction.
(2) We show that the new Link Propagation can also be used for efficient link prediction on large dynamic graphs whose edges evolve over time.
(3) We demonstrate experimental results on the effectiveness of the proposed method for link prediction in a large network with $400,000$ nodes and more. We show that despite using much less computational time and space, the prediction qualities of the approximated version of the Link Propagation are competitive to (and sometimes are better than) those of the exact ones.

The rest of the paper is organized as follows. Section 2 defines the link prediction problem that we consider in this paper. Section 3 reviews the Link Propagation method first proposed in [10]. In Section 4, the procedures for an exact and approximate Link Propagation methods on static and dynamic (time-evolving) graphs are presented. In Section 5, several experimental results to demonstrate the scalability of the proposed methods and the accuracies of their approximations are shown. Section 6 summarizes the related work, and Section 7 concludes this paper with some discussion and promising future work.

2 Link Prediction Problem

The link prediction problem is usually described as a task to predict how likely a link exists between an arbitrary *pair* of nodes in a graph or network.

Let us denote two subsets of nodes of a network by $X \equiv \{x_1, x_2, \ldots, x_M\}$ and $Y \equiv \{y_1, y_2, \ldots, y_N\}$. Some or all of X and Y may be identical depending on the applications. We denote the number of nodes by $M \equiv |X|$ and $N \equiv |Y|$.

Our goal is to predict how likely a link exists for arbitrary node pair $(x_i, y_j) \in X \times Y$, which we refer to as the *link strength*. Namely, we want to output a matrix F, each of whose element $[\mathsf{F}]_{i,j}$ is the link strength between x_i and y_j. A large value of link strength indicates high confidence in the existence of the link, and a small value indicates high confidence in the absence of the link.

We define another $M \times N$ matrix F^* to represent the observed parts of the links of the network. Each element of F^* is defined as

$$[\mathsf{F}^*]_{i,j} \equiv \begin{cases} 1 & \text{if there exists a link between } (i,j), \\ 0 & \text{otherwise (link status is unknown for } (i,j)). \end{cases}$$

To estimate the link strength for the pairs whose link status is unknown (i.e. for the elements of F^* filled with zeros), F^* plays the role of the target values given in a training dataset in (positive-and-unlabelled) supervised learning.

As side information for link prediction, we assume that we are also given similarity matrices W_X and W_Y among the nodes in X and Y, respectively. Those matrices are non-negative and symmetric.

In summary, the input and output of the link prediction problem discussed in this paper is defined as follows.
[**Input**]: A matrix F^* representing the known parts of the graph, and two symmetric non-negative matrices W_X and W_Y for two node sets X and Y.
[**Output**]: A matrix F representing the link strength of all node pairs in $X \times Y$.

3 Review of the Link Propagation Method

In this section, we review the formulation of the Link Propagation introduced in [10]. The Link Propagation applies the idea of the *label propagation* methods [29,31] to link prediction. The label propagation methods are usually used for predicting the labels of unlabeled nodes by using the *label propagation principle*: "Two nodes that are similar to each other are likely to have the same label". Modifying the label propagation principle, the Link Propagation states the analogous version of the inference principle as "Two node pairs that are similar to each other are likely to have the same link strength".

Applying the label propagation principle to pairs of nodes, we obtain the following objective function to minimize (notice that $\mathbf{vec}\,(\mathsf{F})$ is the vectorization operation of matrix F)

$$J(\mathsf{F}) = \frac{\sigma}{2}\mathbf{vec}\,(\mathsf{F})^\top \mathsf{L}\mathbf{vec}\,(\mathsf{F}) + \frac{1}{2} \parallel \mathbf{vec}\,(\mathsf{F}) - \mathbf{vec}\,(\mathsf{F}^*) \parallel_2^2, \tag{1}$$

where the first term indicates that the two link strength values $[\mathsf{F}]_{i,j}$ and $[\mathsf{F}]_{\ell,m}$ for the two pairs should be close to each other if the similarity between the two pairs is large. The second term is the loss function that fits the predictions F to their target values F^* for the known parts of the network. It also plays a role as a regularization term to prevent the predictions from being too far from zero, and for numerical stability. The $\sigma > 0$ is a regularization parameter which balances the two terms of Eq. (1). The $MN \times MN$ matrix L is a *Laplacian matrix*. For the Link Propagation, the previous work [10] recommended using the *Kronecker product Laplacian* defined as

$$\mathsf{L} \equiv \mathsf{D}_Y \otimes \mathsf{D}_X - \mathsf{W}_Y \otimes \mathsf{W}_X, \tag{2}$$

or the *Kronecker sum Laplacian* defined as

$$\mathsf{L} \equiv \mathsf{D}_Y \oplus \mathsf{D}_X - \mathsf{W}_Y \oplus \mathsf{W}_X = \mathsf{L}_Y \oplus \mathsf{L}_X, \tag{3}$$

where D_X is a diagonal matrix whose diagonal elements are $[\mathsf{D}_X]_{i,i} := \sum_j [\mathsf{W}_X]_{i,j}$, and $\mathsf{L}_X \equiv \mathsf{D}_X - \mathsf{W}_X$ is the Laplacian matrix for W_X. The matrices D_Y and L_Y

are defined similarly. Note that the $\otimes$ operator indicates the Kronecker product and the $\oplus$ operator indicates the Kronecker sum, whose definitions can be found in [12]. Here are the ideas behind the Kronecker product and the Kronecker sum Laplacians. Let us consider two pairs of nodes (x_i, y_j) and (x_ℓ, y_m). The Kronecker product Laplacian indicates that the two pairs are similar to each other if x_i and x_ℓ are similar to each other as well as y_j and y_m. This is basically similar to the pair-wise similarity used in kernel methods [2,3,16]. In constrast, the Kronecker sum Laplacian indicates that the two pairs are similar to each other if x_i and x_ℓ are identical and y_j and y_m are similar to each other, or x_i and x_ℓ are similar to each other and y_j and y_m are identical.

Minimizing Eq. (1) with respect to $\mathbf{vec}(\mathsf{F})$, we obtain the system of linear equations

$$(\sigma \mathsf{L} + \mathsf{I})\, \mathbf{vec}(\mathsf{F}) = \mathbf{vec}(\mathsf{F}^*)\,. \tag{4}$$

Using the elements of F for link prediction, the accuracy of the Link Propagation is shown to be competitive with the other state-of-the-art node-information-based methods such as the pairwise SVM [2,3,16] and the metric learning [23]. Figure 1 shows a comparison of AUC values by several methods on medium-size graphs. They are, SVD as a link-based method (with rank= 6), the pairwise SVM (with $C = 1$), metric learning (with rank= 6), and the Link Propagation method with the Kronecker product (**KP**) similarity and the Kronecker sum (**KS**) similarity (with $\sigma = 0.01$) on predicting metabolic networks [26][1]. The similarity matrices (or kernel matrices) were constructed by taking average of the three given matrices named 'phylogenetic', 'expression', and 'localization', and the Laplacian (or kernel matrices) were normalized. The hyperparameters were tuned appropriately. The AUC values were evaluated by 5-fold cross validation with 10% training data. We can see from the figure that the link-based method (SVD) performs poorly because the training data is sparse, and the Link Propagation (especially KS) achieves the best performance.

The authors of [10] proposed a conjugate gradient-based method for solving Eq. (4). However one unavoidable drawback of the conjugate gradient-based method is its large memory requirements for storing the matrices such as W_X, F^*, and F due to the nature of Kronecker operators. The matrix F is inherently dense even when the others are sparse, and in many cases, only parts of its elements are needed for the link prediction (e.g., on some small subset of users and ads). In the hereafter, we will develop exact and approximate solutions for the Link Propagation to overcome the limitation and furtherly widen its applications on large-size and dynamic graphs.

4 Fast and Scalable Solutions for the Link Propagation

In this section, we propose novel methods for the Link Propagation that require much less memory than those based on the conjugate gradient method. We first show an exact method and then, extending it, present an approximate method for the Link Propagation.

[1] `http://web.kuicr.kyoto-u.ac.jp/supp/yoshi/ismb05/`

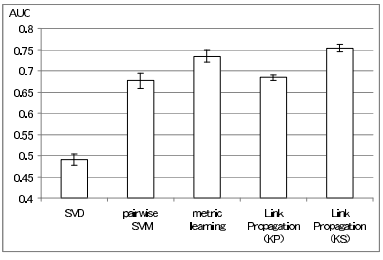

Fig. 1. The accuracies of the Link Propagation methods (KS & KP) vs. those of other methods

As a matter of convenience in deriving the solution, we use the normalized versions of the Laplacian matrices in Eqs. (2) and (3). The normalized Kronecker product Laplacian matrix is given as

$$\begin{aligned} \mathsf{L} &\equiv \mathsf{I} - (\mathsf{D}_Y \otimes \mathsf{D}_X)^{-\frac{1}{2}} (\mathsf{W}_Y \otimes \mathsf{W}_X) (\mathsf{D}_Y \otimes \mathsf{D}_X)^{-\frac{1}{2}} \\ &= \mathsf{I} - \left(\mathsf{D}_Y^{-\frac{1}{2}} \mathsf{W}_Y \mathsf{D}_Y^{-\frac{1}{2}}\right) \otimes \left(\mathsf{D}_X^{-\frac{1}{2}} \mathsf{W}_X \mathsf{D}_X^{-\frac{1}{2}}\right), \end{aligned} \tag{5}$$

while the normalized Kronecker sum Laplacian matrix is given as

$$\begin{aligned} \mathsf{L} &\equiv \left(\mathsf{I} - \left(\mathsf{I} - \mathsf{D}_Y^{-\frac{1}{2}} \mathsf{W}_Y \mathsf{D}_Y^{-\frac{1}{2}}\right) \oplus \left(\mathsf{I} - \mathsf{D}_X^{-\frac{1}{2}} \mathsf{W}_X \mathsf{D}_X^{-\frac{1}{2}}\right)\right) \\ &= 3\mathsf{I} - \left(\mathsf{D}_Y^{-\frac{1}{2}} \mathsf{W}_Y \mathsf{D}_Y^{-\frac{1}{2}}\right) \oplus \left(\mathsf{D}_X^{-\frac{1}{2}} \mathsf{W}_X \mathsf{D}_X^{-\frac{1}{2}}\right). \end{aligned} \tag{6}$$

Both of the normalized Laplacians in Eqs. (5) and (6) can be written unifiedly in a general form as

$$\mathsf{L} \equiv c\mathsf{I} - \left(\mathsf{D}_Y^{-\frac{1}{2}} \mathsf{W}_Y \mathsf{D}_Y^{-\frac{1}{2}}\right) \circledast \left(\mathsf{D}_X^{-\frac{1}{2}} \mathsf{W}_X \mathsf{D}_X^{-\frac{1}{2}}\right).$$

Notice that the operator $\circledast \in \{\otimes, \oplus\}$ corresponds to the Kronecker product $\otimes$, or, the Kronecker sum $\oplus$ operator. In this paper, we can consider either of them by appropriately setting the value of c (which is 1 for the Kronecker product, and 3 for the Kronecker sum).

For simplicity, let us write:

$$\tilde{\mathsf{W}}_X \equiv \mathsf{D}_X^{-\frac{1}{2}} \mathsf{W}_X \mathsf{D}_X^{-\frac{1}{2}}, \quad \tilde{\mathsf{W}}_Y \equiv \mathsf{D}_Y^{-\frac{1}{2}} \mathsf{W}_Y \mathsf{D}_Y^{-\frac{1}{2}}, \tag{7}$$

to obtain the following solution of Eq. (4):

$$\mathbf{vec}\,(\mathsf{F}) = \left((1 + c\sigma)\,\mathsf{I} - \sigma \tilde{\mathsf{W}}_Y \circledast \tilde{\mathsf{W}}_X\right)^{-1} \mathbf{vec}\,(\mathsf{F}^*). \tag{8}$$

4.1 An Exact Link Propagation

Here we explain an algorithm to compute the exact solution of Eq. (8). The algorithm is the foundation to develop a scalable and faster Link Propagation.

One of the key techniques in this paper is a direct method to efficiently compute the inverse matrix on the right-hand side of Eq. (8) that involves Kronecker operators. In what follows, we will show how to obtain the exact and approximate inverse by exploiting the following well-known theorem (see. e.g., [12]) on the eigenvalues and eigenvectors of the Kronecker sum and product.

Theorem 1. *Let the eigenvalues of the matrices* $\tilde{\mathsf{W}}_X$ *and* $\tilde{\mathsf{W}}_Y$ *be, respectively,* $\{{\lambda_X}^{(i)}\}_{i=1}^{M}$ *and* $\{{\lambda_Y}^{(j)}\}_{j=1}^{N}$*, with the corresponding eigenvectors are given as, respectively, the column of the matrices* V_X *and* V_Y*. Then the eigenvectors of the Kronecker product* $\tilde{\mathsf{W}}_Y \otimes \tilde{\mathsf{W}}_X$ *and the Kronecker sum* $\tilde{\mathsf{W}}_Y \oplus \tilde{\mathsf{W}}_X$ *are the same (column vectors of)* $\mathsf{V}_Y \otimes \mathsf{V}_X$*, while the eigenvalues are* $\{{\lambda_X}^{(i)}{\lambda_Y}^{(j)}\}_{(i,j)=(1,1)}^{M,N}$ *for the Kronecker product, and* $\{{\lambda_X}^{(i)} + {\lambda_Y}^{(j)}\}$ *for the Kronecker sum.*

Since the similarity matrices are positive semidefinite matrices, they can be eigendecomposed as follows (those of $\tilde{\mathsf{W}}_Y$ are omitted):

$$\tilde{\mathsf{W}}_X = \mathsf{V}_X \mathsf{diag}\left({\lambda_X}^{(1)}, {\lambda_X}^{(2)}, \ldots, {\lambda_X}^{(M)}\right) \mathsf{V}_X^\top, \tag{9}$$

Now let us define the matrix $\mathsf{\Lambda}$ to be either

$$[\mathsf{\Lambda}]_{i,j} \equiv {\lambda_X}^{(i)}{\lambda_Y}^{(j)}, \text{ or } [\mathsf{\Lambda}]_{i,j} \equiv {\lambda_X}^{(i)} + {\lambda_Y}^{(j)}, \tag{10}$$

where the former is for the Kronecker product and the latter is for the Kronecker sum. Then by Theorem 1, we can write the inverse matrix in Eq. (8) as

$$\left((1+c\sigma)\,\mathsf{I} - \sigma\tilde{\mathsf{W}}_Y \circledast \tilde{\mathsf{W}}_X\right)^{-1} = \left((1+c\sigma)\,\mathsf{I} - \sigma\mathsf{V}\mathsf{diag}\left(\mathbf{vec}\left(\mathsf{\Lambda}\right)\right)\mathsf{V}^\top\right)^{-1}, \tag{11}$$

where $\mathsf{V} = \mathsf{V}_Y \otimes \mathsf{V}_X$. Since, it holds that $\mathsf{V}\,\mathsf{V}^\top = \mathsf{I}$, Eq. (11) can be transformed into

$$\left((1+c\sigma)\,\mathsf{I} - \sigma\mathsf{V}\mathsf{diag}\left(\mathbf{vec}\left(\mathsf{\Lambda}\right)\right)\mathsf{V}^\top\right)^{-1} = \mathsf{V}\left((1+c\sigma)\,\mathsf{I} - \sigma\mathsf{diag}\left(\mathbf{vec}\left(\mathsf{\Lambda}\right)\right)\right)^{-1}\mathsf{V}^\top.$$

Notice that $((1+c\sigma)\,\mathsf{I} - \sigma\mathsf{diag}\left(\mathbf{vec}\left(\mathsf{\Lambda}\right)\right))$ is a diagonal matrix whose inverse can be easily calculated as the following matrix D

$$[\mathsf{D}]_{i,j} \equiv \left(1 + \sigma\left(c - [\mathsf{\Lambda}]_{i,j}\right)\right)^{-1}. \tag{12}$$

This gives us the the solution of Eq. (8) as $\mathbf{vec}\left(\mathsf{F}\right) = \mathsf{V}\ \mathsf{diag}\left(\mathbf{vec}\left(\mathsf{D}\right)\right)\mathsf{V}^\top\mathbf{vec}\left(\mathsf{F}^*\right)$. We can further simplify this equation as

$$\mathbf{vec}\left(\mathsf{F}\right) = \mathsf{V}\ \mathsf{diag}\left(\mathbf{vec}\left(\mathsf{D}\right)\right)\mathbf{vec}\left(\mathsf{V}_X^\top\mathsf{F}^*\mathsf{V}_Y\right) = \mathbf{vec}\left(\mathsf{V}_X\left(\mathsf{D} * \left(\mathsf{V}_X^\top\mathsf{F}^*\mathsf{V}_Y\right)\right)\mathsf{V}_Y^\top\right),$$

where in the derivation we used the "vec-tricks" techniques [12,24] as

$$\left(\mathsf{V}_Y \otimes \mathsf{V}_X\right)\mathbf{vec}\left(\mathsf{F}^*\right) = \mathbf{vec}\left(\mathsf{V}_X\mathsf{F}^*\mathsf{V}_Y^\top\right), \tag{13}$$

and $\mathsf{D} * \mathsf{A}$ is the Hadamard product of matrices D and A. Taking out the **vec** operation, we can describe the solution as

$$\mathsf{F} = \mathsf{V}_X\left(\mathsf{D} * \left(\mathsf{V}_X^\top\mathsf{F}^*\mathsf{V}_Y\right)\right)\mathsf{V}_Y^\top. \tag{14}$$

Algorithm 1. Exact Link Propagation. Input: $(\mathsf{W}_X, \mathsf{W}_Y, \mathsf{F}^*, \sigma)$.

1: Compute the normalized similarity matrices $\tilde{\mathsf{W}}_X$ and $\tilde{\mathsf{W}}_Y$ as Eq. (7)
2: Compute the eigendecomposition of $\tilde{\mathsf{W}}_X$ and $\tilde{\mathsf{W}}_Y$ as in Eq. (9)
3: Compute the elements of the matrix D as in Eq. (12), where Eq. (10) is used for the Kronecker product or sum
4: Compute the solution from Eq. (14)

The complete procedure to compute the exact solution matrix F is summarized in Algorithm 1.

Although computing the exact solution of the Link Propagation with Algorithm 1 requires intensive matrix operations, in the *worst-case* analysis it is more efficient than the one with the conjugate gradient in [10]. Algorithm 1 requires only $O(M^3 + N^3 + M^2N + N^2M + MN|\mathsf{F}^*|)$ computational time. In detail, Step 1 needs $O(M^2 + N^2)$ time, while Step 2 and Step 3 require $O(M^3 + N^3)$ and $O(MN)$ time, respectively. Computing all elements of F in Step 4 takes $O(M^2N + MN^2 + MN|\mathsf{F}^*|)$ time, which means approximately N or M operations is required for computing each element of F. When $M = N$, Algorithm 1 only requires $O(N^3)$ time in contrast to $O(N^5)$ time of the conjugate gradient method. The space complexity is $O(N^2)$, which is the same with the conjugate gradient method[2]. However, the exact solution in Algorithm 1 is potentially more useful because it opens a path to apply rich techniques of linear algebra, such as matrix approximation and factorization, so that the time and space complexities can be reduced while the accuracy levels are maintained. Moreover, it also enables us to design a fast incremental method for the Link Propagation in dynamic graphs without recomputing from scratch. We will briefly explain the techniques in the following sections.

4.2 An Approximate Link Propagation

Here we present an approximate solution of the Link Propagation that mitigates the large computational time and space complexities of the exact solution.

For nodes on the order of millions, not only F, but also storing the similarity matrices in main memory is already prohibitive. Therefore, we need to consider more scalable and faster ways to compute the approximations of the Link Propagation. For this purpose, we can use any matrix approximation technique whose computation process does not require storing all elements of the matrix in the main memory. One of the approximation techniques used in this paper is the *incomplete Cholesky decomposition*. However, note that other matrix approximation techniques, such as those based on singular value decomposition, could also be used. Thus, we can obtain the low-rank approximation of the similarity matrices W_X and W_Y as

$$\mathsf{W}_X \approx \mathsf{G}_X \mathsf{G}_X^\top, \quad \mathsf{W}_Y \approx \mathsf{G}_Y \mathsf{G}_Y^\top. \tag{15}$$

[2] Without the vec-tricks, the time and space complexities of the conjugate gradient are $O(N^6)$ and $O(N^4)$, respectively.

Here, G_X and G_Y are $M \times \bar{M}$ and $N \times \bar{N}$ matrices, respectively, where $\bar{M}$ and $\bar{N}$ are the parameters for the approximation ranks whose values can be set appropriately by the users. The total computational cost of obtaining such matrices by the incomplete Cholesky decomposition is $O(M\bar{M}^2 + N\bar{N}^2)$. Moreover, the sum of each row of the approximate matrices can be computed from G_X and G_Y by $\mathsf{D}_X = \mathsf{diag}\left(\mathsf{G}_X\mathsf{G}_X^\top \mathbf{1}\right)$ and $\mathsf{D}_Y = \mathsf{diag}\left(\mathsf{G}_Y\mathsf{G}_Y^\top \mathbf{1}\right)$. Thus, by defining the following matrices:

$$\tilde{\mathsf{G}}_X \equiv \mathsf{D}_X^{-\frac{1}{2}}\mathsf{G}_X, \quad \tilde{\mathsf{G}}_Y \equiv \mathsf{D}_X^{-\frac{1}{2}}\mathsf{G}_Y, \tag{16}$$

we can write the normalized similarity matrices as

$$\tilde{\mathsf{W}}_X \approx \mathsf{D}_X^{-\frac{1}{2}}\mathsf{G}_X\mathsf{G}_X^\top\mathsf{D}_X^{-\frac{1}{2}} = \tilde{\mathsf{G}}_X\tilde{\mathsf{G}}_X^\top, \quad \tilde{\mathsf{W}}_Y \approx \mathsf{D}_Y^{-\frac{1}{2}}\mathsf{G}_Y\mathsf{G}_Y^\top\mathsf{D}_Y^{-\frac{1}{2}} = \tilde{\mathsf{G}}_Y\tilde{\mathsf{G}}_Y^\top.$$

Notice that since $\bar{M} \ll M$ and $\bar{N} \ll N$, the eigendecomposition of $\tilde{\mathsf{G}}_X^\top\tilde{\mathsf{G}}_X$ and $\tilde{\mathsf{G}}_Y^\top\tilde{\mathsf{G}}_Y$ can be performed easily to obtain the following eigenvalues (which are the same as those of the approximate similarity matrices) and eigenvectors satisfying (those of $\tilde{\mathsf{G}}_Y^\top\tilde{\mathsf{G}}_Y$ are omitted)

$$\tilde{\mathsf{G}}_X^\top\tilde{\mathsf{G}}_X = \bar{\mathsf{U}}_X\mathsf{diag}\left(\bar{\lambda}_X^{(1)}, \bar{\lambda}_X^{(2)}, \ldots, \bar{\lambda}_X^{(\bar{M})}\right)\bar{\mathsf{U}}_X^\top. \tag{17}$$

Now, the eigenvectors of the approximate similarity matrix $\tilde{\mathsf{W}}_X$ (and similarly for $\tilde{\mathsf{W}}_Y$) can be computed from the equation

$$\bar{\mathsf{V}}_X \equiv \tilde{\mathsf{G}}_X\bar{\mathsf{U}}_X\mathsf{diag}\left(\bar{\lambda}_X^{(1)}, \bar{\lambda}_X^{(2)}, \ldots, \bar{\lambda}_X^{(\bar{M})}\right)^{-\frac{1}{2}}. \tag{18}$$

Similar to the exact case, by Eqs. (15) and (18), letting $\bar{\mathsf{V}} = \bar{\mathsf{V}}_Y \otimes \bar{\mathsf{V}}_X$, we can write the inverse matrix in Eq. (8) as

$$\left((1+c\sigma)\,\mathsf{I} - \sigma\tilde{\mathsf{W}}_Y \circledast \tilde{\mathsf{W}}_X\right)^{-1} = \left((1+c\sigma)\,\mathsf{I} - \sigma\bar{\mathsf{V}}\ \mathsf{diag}\left(\mathbf{vec}\left(\bar{\Lambda}\right)\right)\bar{\mathsf{V}}^\top\right)^{-1}. \tag{19}$$

Notice that the elements of the matrix $\bar{\Lambda}$ are appropriately defined as in Eq. (10) for the Kronecker product or sum as in the exact case. However, differing from the exact case, we have $\left(\bar{\mathsf{V}}_Y \otimes \bar{\mathsf{V}}_X\right)\left(\bar{\mathsf{V}}_Y \otimes \bar{\mathsf{V}}_X\right)^\top \neq \mathsf{I}$. Fortunately, by using the Woodbury equation (see, e.g., [4]) and $\bar{\mathsf{V}}^\top\ \bar{\mathsf{V}} = \mathsf{I}$, we can compute the inverse matrix in Eq. (19), as follows.

$$\left((1+c\sigma)\,\mathsf{I} - \sigma\tilde{\mathsf{W}}_Y \circledast \tilde{\mathsf{W}}_X\right)^{-1}$$
$$= \frac{\mathsf{I}}{1+c\sigma} + \frac{\bar{\mathsf{V}}}{(1+c\sigma)^2}\left(\frac{\mathsf{diag}\left(\mathbf{vec}\left(\bar{\Lambda}\right)\right)^{-1}}{\sigma} - \frac{\mathsf{I}}{1+c\sigma}\right)^{-1}\bar{\mathsf{V}}^\top.$$

Defining the matrix $\bar{\mathsf{D}}$ as

$$[\bar{\mathsf{D}}]_{i,j} \equiv \left(\frac{1}{\sigma[\bar{\Lambda}]_{i,j}} - \frac{1}{1+c\sigma}\right)^{-1} = \frac{\sigma\,(1+c\sigma)\,[\bar{\Lambda}]_{i,j}}{1+c\sigma-\sigma[\bar{\Lambda}]_{i,j}}, \tag{20}$$

Algorithm 2. Approximate Link Propagation. Input: $(\mathsf{W}_X, \mathsf{W}_Y, \mathsf{F}^*, \sigma)$.

1: Compute the low-rank approximation matrices of W_X and W_Y as in Eq. (15)
2: Compute the normalized matrices, $\tilde{\mathsf{G}}_X$ and $\tilde{\mathsf{G}}_Y$, as in Eq. (16)
3: Compute the eigendecomposition of $\tilde{\mathsf{G}}_X^\top \tilde{\mathsf{G}}_X$ and $\tilde{\mathsf{G}}_Y^\top \tilde{\mathsf{G}}_Y$ as in Eq. (17)
4: Compute the eigenvectors of $\tilde{\mathsf{G}}_X \tilde{\mathsf{G}}_X^\top$ and $\tilde{\mathsf{G}}_Y \tilde{\mathsf{G}}_Y^\top$ according to Eq. (18)
5: Compute the elements of the matrix $\bar{\mathsf{D}}$ according to Eq.(20), where the values of elements of $\bar{\Lambda}$ are adjusted according to that in the Kronecker sum or product
6: Using the core solution in Eq. (22), compute the elements of F according to Eq. (21)

we can compute the approximate solution of the Link Propagation as

$$\mathbf{vec}\left(\mathsf{F}\right) = \frac{1}{1+c\sigma}\mathbf{vec}\left(\mathsf{F}^*\right) + \frac{1}{\left(1+c\sigma\right)^2}\bar{\mathsf{V}}\ \mathsf{diag}\left(\mathbf{vec}\left(\bar{\mathsf{D}}\right)\right)\bar{\mathsf{V}}^\top \mathbf{vec}\left(\mathsf{F}^*\right).$$

By using the vec-trick techniques and taking-out the **vec** operation, the solution matrix F is efficiently obtained as

$$\mathsf{F} = \frac{1}{1+c\sigma}\mathsf{F}^* + \frac{1}{\left(1+c\sigma\right)^2}\bar{\mathsf{V}}_X\left(\bar{\mathsf{D}} * \left(\bar{\mathsf{V}}_X^\top \mathsf{F}^* \bar{\mathsf{V}}_Y\right)\right)\bar{\mathsf{V}}_Y^\top. \tag{21}$$

Since F is a large matrix, storing all of its elements is prohibitively expensive. Instead, we can store its compact representation by computing and storing the smaller *core matrix*

$$\bar{\mathsf{D}} * \left(\bar{\mathsf{V}}_X^\top \mathsf{F}^* \bar{\mathsf{V}}_Y\right), \tag{22}$$

along with the eigenvectors of the approximate similarity matrices. This information is sufficient to compute the elements of F on demand. We summarize the procedure to compute the approximate solution of the Link Propagation in Algorithm 2. Two major advantages of Algorithm 2 are its using much less computational space and time, and its capability to compactly represent the elements of F by storing the core matrix and the eigenvectors of the low-rank similarity matrices whose sizes are mostly linear in the number of nodes (we will show later in the experiments that low-rank approximation matrices are sufficient). That is, its space complexity is only $O(\bar{M}\bar{N} + \bar{M}M + \bar{N}N)$, where the first term is due to the core matrix, and the last two terms are due to the eigenvectors. With regards to its computational complexity, we can see that up to Step 5, Algorithm 2 requires $O(M\bar{M}^2 + N\bar{N}^2 + \bar{M}^3 + \bar{N}^3 + |\mathsf{F}^*|\bar{M}\bar{N})$ time: Step 1 and Step 4 need $O(M\bar{M}^2 + N\bar{N}^2)$ time, Step 2 requires $O(M\bar{M} + N\bar{N})$ time, and Step 3 takes $O(M\bar{M}^2 + N\bar{N}^2 + \bar{M}^3 + \bar{N}^3)$ time, while the computation of the core matrix in Step 5 and Step 6 need $O(\bar{M}\bar{N}|\mathsf{F}^*|)$ time. To compute all elements of F in Step 6, we need $O(\bar{M}\bar{N}M + \bar{N}MN)$ time, which implies constant computation time per element for small $\bar{M}$ and $\bar{N}$. Indeed, in the experiments we could choose some small values for $\bar{M}$ and $\bar{N}$ to obtain satisfactory link prediction results.

4.3 Link Prediction on Dynamic Graphs

Here we show another desirable aspect of the exact and approximated solutions of the Link Propagation that enables an efficient adjusment of the link prediction scores to cope with addition and deletion of edges in the networks.

Changes in graphs (which result in the change of values of F^*) occur quite frequently, and hence timely updating the link prediction scores is crucial. For example, consider the graph whose node set X is the subset of users and node set Y is the subset of ads, where the similarity matrices are computed from the users' profiles, tags, etc. In this setting, the edges represent the users' clicking actions. New clicks can be regarded as addition, while non-clicking actions can be regarded as subtraction of the corresponding elements in F^*. Changes in the link prediction scores due to the addition and substraction of elements of F^* can be computed straightforwardly in our methods.

Let us denote the parts of F^* that changed as $\Delta\mathsf{F}^*$, namely, the new matrix is $\tilde{\mathsf{F}^*} = \mathsf{F}^* + \Delta\mathsf{F}^*$. It is clear from Eqs. (21) and (14), that the incremental updates of the prediction scores for all elements of F can be performed by updating the smaller core matrix in Eq. (22) (also Eq. (14)) as

$$\bar{\mathsf{D}} * \left(\bar{\mathsf{V}}_X^\top \Delta\mathsf{F}^* \bar{\mathsf{V}}_Y\right), \tag{23}$$

that implies the time complexity of the incremental update of the core matrix is $O(\bar{M}\bar{N}|\Delta\mathsf{F}^*|)$, which is proportional to the number of changes in F^*. Notice that aparts from the approximated similarity matrices, the incremental updates in our methods are exact, i.e., their accuracies are the same with those of the compute-from-scratch methods. The limitation of the incremental update presented here is that it requires the same set of nodes and similarity matrices.

5 Experiments

In this section, we present some experimental results for predicting the links between pair of nodes (*pair-wise link prediction*) using exact and approximate solutions of the Link Propagation. The purpose of the experiments is to show that the approximated Link Propagation performs quite well inspite of requiring significantly less time and space than the exact one. We first describe the datasets of the experiments.

5.1 Datasets

We tested the exact and approximate Link Propagation on two types of network datasets available from the Web Spam Challenge archive[3]. The summary of the datasets is listed in Tables 1 and 2. The Link Propagation was performed when $X = Y$.

The first type of datasets were created from the network of 400,000 web pages in a webgraph (called *Web400K* dataset) whose links correspond to the hyperlinks of webpages. For each webpage, there is a sparse feature vector that contains the frequency of words in the web page. The (x, y) element of the similarity matrix W in the experiment is the inner product of the feature vectors of the corresponding webpages x and y. From this dataset, we created two datasets each of which contains exactly 1,000 and 3,000 nodes of Web400K (denoted by

[3] `http://webspam.lip6.fr/wiki/pmwiki.php?n=Main.PhaseIITrainningCorpora`

Table 1. Datasets from the web graph

	Web1K	Web3K	Web400K
#Nodes	1,000	3,000	400,000
#Links	20,214	38,453	13,068,666

Table 2. Datasets from the host graph

	Host1K	Host3K	Host9K
#Nodes	1,000	3,000	9,000
#Links	6,223	156,411	505,809

Web1K, and *Web3K*, respectively) that are reachable and closest to the node with id "1" (That is, starting from node 1, we add its neighbors, by prioritizing the node with lower id, to be the members of the dataset and repeat this process until we find exactly $k \in \{1000, 3000\}$ nodes. If at any time we cannot add a node because the network is disconnected, we add the node with the lowest id that is not in the current dataset, and repeat). We avoid extracting random subgraphs for ease of reproducibility of the results.

The second type of datasets were created from the network of 9,000 hosts in a hostgraph (called *Host9K* dataset) whose links correspond to the existences of linked pages between pages in the hosts. For each host, there is a sparse feature vector that represents a normalized tf-idf vector over the content of its pages. The feature vectors were used to build the similarity matrix similarly as in the webgraph. In our experiments, we again created two small graphs from the hostgraph that consist of, respectively, simply the first 1,000 and 3,000 nodes in the hostgraph. We call the datasets *Host1K* and *Host3K*, respectively. We evaluated the effectiveness and efficiency of the exact and approximate Link Propagation by measuring their time complexities and accuracies. Time complexity means the amount of time for an algorithm to be ready to output a value of prediction and not the amount of time to compute all elements of F (whose size is very large). The accuracies were evaluated by AUC, area under the ROC curve, which is commonly used in supervised learning. We randomly selected $\lambda = 10\%, 75\%$ of the links in the datasets for at least three times for each dataset, and used them as training data, and evaluated averaged AUC values on all other pair of nodes. The standard deviations of AUC values in our experiments were small.

5.2 Implementation

In all experiments, we set $\sigma = 0.001$, used the Laplacian Sum for all graphs because of the page limit. We evaluated the AUC values and measured the computational time for ten times for datasets other than the Host3K and Web3K graphs (which are just repeated for three times because of the time limitation). All algorithms were implemented in 64-bit Java using Colt© packages matrix operations. They were tested on several (virtual) machines running Linux and Microsoft® Windows XP®. The computation times were measured on instances executed on an IBM IntelliStation® running 64-bit Red Hat Enterprise Linux 5 with an Intel® Core 2 Quad CPU Q6600@2.40-GHz CPU and 8-GB RAM.

Table 3. The AUC scores of link prediction on the webgraph by 10% sampling

Dataset	Exact	$\bar{M} = 20$	$\bar{M} = 50$	$\bar{M} = 100$
Web1K	0.892	0.854	0.930	**0.944**
Web3K	0.914	0.769	0.921	**0.956**
Web400K	NA	0.875	0.876	**0.894**

Table 4. The AUC scores of link prediction on the webgraph by 75% sampling

Dataset	Exact	$\bar{M} = 20$	$\bar{M} = 50$	$\bar{M} = 100$
Web1K	0.926	0.858	0.934	**0.954**
Web3K	0.934	0.771	0.925	**0.960**
Web400K	NA	0.893	0.888	**0.908**

5.3 Experimental Results on Static Graphs

Here we present the accuracies and the computational time of the exact and approximated Link Propagation method for the link prediction on the webgraphs and hostgraphs when some percentage of links are given as training samples.

Tables 3 and 4 show the AUC values by the exact and approximate Link Propagation on webgraphs, when, respectively, 10% and 75% of links were used for training. The tables show a surprising result that at low values of the approximation rank $\hat{M}$, the approximate method can sometimes produce better AUC values than the exact method that at $\hat{M} = 50$ the AUC values of the approximate Link Propagation were better on the Web1K and Web3K graphs. The AUC value of the exact method on the Web400K graph is not available within our limited resources (denoted by *NA* in the tables), but the AUC values of the approximate method were sufficiently high.

Tables 5 and 6 summarize the AUC values on the hostgraphs similarly as in the previous tables. From the tables, we can see that the AUC values of the approximate method increased as the the values of $\hat{M}$ and the number of sample links increased. However, the AUC values were less than the corresponding AUC values of the exact method (For the same reason, we could not compute the AUC value of the exact method on the Host9K). Unlike what we have observed from the webgraphs, they are never better than those of the exact ones. This might reflect the properties of similarity matrices: the feature vectors of nodes in the webgraphs correspond to pages and tend to be of low rank, while the feature vectors of hosts in the hostgraphs correspond to accumulation of feature vectors of various pages and thus, tend to be of high rank and hard to approximate. The comparison of the computation time for the exact and approximate methods for each graph in the datasets is shown in Figure 2. The approximate method is efficient enough that we could compute the link prediction over the Web400K and Host9K graphs within reasonable time. For example, from the figure we can see that the link prediction on the Web400K graph, whose size is more than one hundred times of that of the Web3K graph can be performed within almost half of the time spent on the Web3K graph to perform the exact Link Propagation. On the Host3K graph, the approximation method can be 50 times faster than the exact one while retaining the accuracy level.

5.4 Experimental Results on Dynamic Graphs

Here we present experimental results for another contribution of our method for the link prediction on dynamic graphs as summarized in Figure 3. We simulated

Table 5. The AUC scores of link prediction on the hostgraph by 10% sampling

Dataset	Exact	$\bar{M} = 20$	$\bar{M} = 50$	$\bar{M} = 100$
Host1K	**0.829**	0.644	0.685	0.711
Host3K	**0.910**	0.610	0.655	0.686
Host9K	NA	0.629	0.560	**0.638**

Table 6. The AUC scores of link prediction on the hostgraph by 75% sampling

Dataset	Exact	$\bar{M} = 20$	$\bar{M} = 50$	$\bar{M} = 100$
Host1K	**0.876**	0.649	0.703	0.741
Host3K	**0.938**	0.611	0.660	0.694
Host9K	NA	0.618	0.570	**0.646**

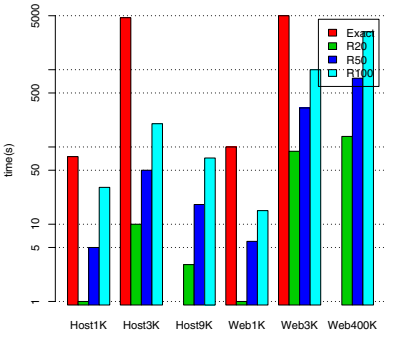

Fig. 2. Comparison of computational time by the exact method, rank-20 approximation (R20), rank-50 approximation (R50) and rank-100 approximation (R100)

the changes of edges by first performing the Link Propagation with 10% of links of the corresponding graph and then incrementally adding 10%, 20%, 30%, 40%, and 65% of the rest of the links at a time. Notice that the effect of deletion of edges is similar as implied by Eq. (23) and therefore omitted.

The left part of Figure 3 shows the computational time with regards to the number of links used in the training on the Host1K and Host3K graphs. The value at 10% links denotes the computational time used for computing the core matrix, the most dominant part, from scratch, while the rest of the values denote those from incrementally updating the core matrix computed from scratch with 10% of links until sufficient portion of links are added. From the figure we can see that on a large-size graph like the Host3K, incrementally updating the core matrix due to addition (deletion) of 10% links is about 10 times faster than computing it from scratch.

The right part of Figure 3 shows the corresponding AUCs where we can notice that the more samples are available, the higher the prediction qualities are. Thus, even for the exact Link Propagation, we only need computing from scratch once (mainly for reading and decomposing the similarity matrices) and then update the core matrix as needed when edges are added or deleted.

6 Related Work

The link prediction problem has been thoroughly studied in the context of predicting biological networks such as protein-protein interaction networks and gene

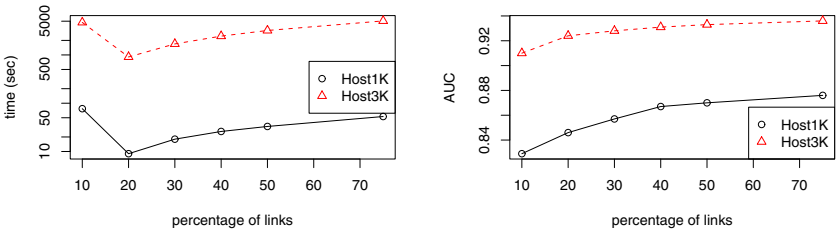

Fig. 3. Computational time (left) and AUCs (right) for the link prediction with incremental addition of links. The initial link prediction was computed with 10% links of the corresponding graph, and the rest was performed with incremental updates.

regulatory networks in the bioinformatics area, and also in the context of link mining [6] in the data mining community.

In bioinformatics, several node-information-based approaches have been proposed, such as an EM-based [11] and dimension-reduction-based approaches [23,26]. The pair-wise kernel with which we compared our method in Figure 1 was proposed for predicting protein-protein interactions [3]. Interestingly, the same kernel was independently proposed for entity resolution [16] and collaborative filtering [2]. In the data mining community, the link prediction problem is being studied as one of the fundamental tasks for the link mining. There are several methods that utilize structural information only such as link metrics (e.g. [14]). Matrix factorization approaches [13,19] are also grouped into topological-information-based methods. There are also supervised learning methods using node information as well as topological information such as [8,15]. Several previous works, e.g. [17,20], that apply the framework of statistical relational learning to link prediction also exist. A similar framework using the so-called exponential random graph model is used in social network analysis [1]. Recently, sophisticated generative models of networks from a Bayesian perspective have been proposed [5,27].

The matrix "vec-trick" in Eq. (13) was first proposed by Vishwanathan et al. [24] for accelerating the computation of the graph kernels. The label propagation technique on graphs was also used in [30], where a batch and incremental method for finding a good low-dimensional latent-space embedding of documents utilizing side information from multiple graphs was proposed. The use of Woodbury equation for handling dynamic bipartite graphs can also be found in [22] (and its conference version). Notice that unlike those in [22], our methods can be used beyond pair-wise link prediction.

7 Discussion and Concluding Remarks

We show fast and scalable semi-supervised learning algorithms for link prediction on static and dynamic graphs using both link information and node information by devising novel methods for exact and approximate Link Propagation. The applications of our methods to large networks with more than $400,000$ nodes

demonstrated that our methods are scalable and their approximation are quite good. The proposed algorithms avoid the use of the conjugate gradient method and directly solve the huge linear equations by (approximating) the matrix inverse, which also allow us to perform efficient leave-one-out estimation [25] for determining the hyper-parameters in the Link Propagation. This will be investigated in the future. Finally, we should also note that although the methods presented in this paper were described for pair-wise link prediction, they can be extended for triplets, quadlets, etc. This can be done by using higher order tensors. We omit the details due to the space limitation.

References

1. Anderson, C.J., Wasserman, S., Crouch, B.: A p^* primer: logit models for social networks. Social Networks 21, 37–66 (1999)
2. Basilico, J., Hofmann, T.: Unifying collaborative and content-based filtering. In: ICML (2004)
3. Ben-Hur, A., Noble, W.S.: Kernel methods for predicting protein-protein interactions. Bioinformatics 21(suppl. 1), i38–i46 (2005)
4. Bishop, C.: Pattern Recognition and Machine Learning. Springer, Heidelberg (2006)
5. Chu, W., Sindhwani, V., Ghahramani, Z., Keerthi, S.: Relational learning with Gaussian processes. In: NIPS (2007)
6. Getoor, L., Diehl, C.P.: Link mining: a survey. SIGKDD Explorations 7(2), 3–12 (2005)
7. Hanneke, S., Xing, E.: Discrete temporal models of social networks. In: Airoldi, E.M., Blei, D.M., Fienberg, S.E., Goldenberg, A., Xing, E.P., Zheng, A.X. (eds.) ICML 2006. LNCS, vol. 4503, pp. 115–125. Springer, Heidelberg (2007)
8. Hasan, M.A., Chaoji, V., Salem, S., Zaki, M.: Link prediction using supervised learning. In: LinkKDD (2005)
9. Hayashi, K., Hirayama, J., Ishii, S.: Dynamic Exponential Family Matrix Factorization. In: Theeramunkong, T., Kijsirikul, B., Cercone, N., Ho, T.-B. (eds.) PAKDD 2009. LNCS, vol. 5476, pp. 452–462. Springer, Heidelberg (2009)
10. Kashima, H., Kato, T., Yamanishi, Y., Sugiyama, M., Tsuda, K.: Link propagation: A fast semi-supervised learning algorithm for link prediction. In: SDM (2009)
11. Kato, T., Tsuda, K., Asai, K.: Selective integration of multiple biological data for supervised network inference. Bioinformatics 21(10), 2488–2495 (2005)
12. Laub, A.J.: Matrix Analysis for Scientists and Engineers. Society for Industrial and Applied Mathematics (2005)
13. Lee, D., Seung, H.: Algorithms for non-negative matrix factorization. In: NIPS, pp. 556–562 (2001)
14. Liben-Nowell, D., Kleinberg, J.: The link prediction problem for social networks. In: CIKM, pp. 556–559 (2004)
15. O'Madadhain, J., Hutchins, J., Smyth, P.: Prediction and ranking algorithms for event-based network data. SIGKDD Explorations 7(2), 23–30 (2005)
16. Oyama, S., Manning, C.D.: Using feature conjunctions across examples for learning pairwise classifiers. In: Boulicaut, J.-F., Esposito, F., Giannotti, F., Pedreschi, D. (eds.) ECML 2004. LNCS (LNAI), vol. 3201, pp. 322–333. Springer, Heidelberg (2004)

17. Popescul, A., Ungar, L.H.: Statistical relational learning for link prediction. In: IJCAI Workshop on Learning Statistical Models from Relational Data (2003)
18. Srebro, N.: Learning with Matrix Factorization. PhD thesis, Massachusetts Institute of Technology (2004)
19. Srebro, N., Rennie, J., Jaakkola, T.: Maximum-margin matrix factorization. In: NIPS, pp. 1329–1336 (2005)
20. Taskar, B., Wong, M., Abbeel, P., Koller, D.: Link prediction in relational data. In: NIPS (2004)
21. Tong, H., Papadimitriou, S., Sun, J., Yu, P.S., Faloutsos, C.: Colibri: fast mining of large static and dynamic graphs. In: KDD, pp. 686–694 (2008)
22. Tong, H., Papadimitriou, S., Yu, P.S., Faloutsos, C.: Fast monitoring proximity and centrality on time-evolving bipartite graphs. Statistical Analysis and Data Mining 1(3), 142–156 (2008)
23. Vert, J.-P., Yamanishi, Y.: Supervised graph inference. In: NIPS (2005)
24. Vishwanathan, S.V.N., Borgwardt, K., Schraudolph, N.: Fast computation of graph kernels. In: NIPS (2007)
25. Wu, M., Schölkopf, B.: Transductive classification via local learning regularization. In: AISTATS (2007)
26. Yamanishi, Y., Vert, J.-P., Kanehisa, M.: Supervised enzyme network inference from the integration of genomic data and chemical information. Bioinformatics 21, i468–i477 (2005)
27. Yu, K., Chu, W., Yu, S., Tresp, V., Xu, Z.: Stochastic relational models for discriminative link prediction. In: NIPS (2007)
28. Zan Huang, H.C., Li, X.: Link prediction approach to collaborative filtering. In: JCDL (2005)
29. Zhou, D., Bousquet, O., Weston, J., Schölkopf, B.: Learning with local and global consistency. In: NIPS, pp. 321–328 (2004)
30. Zhou, D., Zhu, S., Yu, K., Song, X., Tseng, B.L., Zha, H., Giles, C.L.: Learning multiple graphs for document recommendations. In: WWW '08, pp. 141–150. ACM, New York (2008)
31. Zhu, X., Ghahramani, Z., Lafferty, J.: Semi-supervised learning using Gaussian fields and harmonic functions. In: ICML (2003)