Scaling up Link Prediction with Ensembles

Liang Duan[1] Charu Aggarwal[2] Shuai Ma[1,*] Renjun Hu[1] Jinpeng Huai[1]

[1]SKLSDE Lab, Beihang University, China [2]IBM T. J. Watson Research Center, USA
{duanliang, mashuai, hurenjun, huaijp}@buaa.edu.cn charu@us.ibm.com

ABSTRACT

A network with n nodes contains $O(n^2)$ possible links. Even for networks of modest size, it is often difficult to evaluate all pair-wise possibilities for links in a meaningful way. Furthermore, even though link prediction is closely related to missing value estimation problems, such as collaborative filtering, it is often difficult to use sophisticated models such as latent factor methods because of their computational complexity over very large networks. Due to this computational complexity, most known link prediction methods are designed for *evaluating* the link propensity over a *specified* subset of links, rather than for performing a global search over the entire networks. In practice, however, it is essential to perform an exhaustive search over the entire networks. In this paper, we propose an ensemble enabled approach to scaling up link prediction, which is able to decompose traditional link prediction problems into subproblems of smaller size. These subproblems are each solved with the use of latent factor models, which can be effectively implemented over networks of modest size. Furthermore, the ensemble enabled approach has several advantages in terms of performance. We show the advantage of using ensemble-based latent factor models with experiments on very large networks. Experimental results demonstrate the effectiveness and scalability of our approach.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications

Keywords

Link Prediction; Ensembles; Networks; Big Data

1. INTRODUCTION

The problem of *link prediction* or *link inference* is that of predicting the formation of future links in a dynamic and evolving network (see [7, 35, 36] for surveys). The link prediction problem has numerous applications, such as the recommendation of friends in a social network, the recommendation of images in a multimedia network, or the recommendation of collaborators in a scientific network, and, therefore, link prediction methods have been studied extensively because of their numerous applications in various network-centered domains.

Network Sizes	1 GHz	3 GHz	10 GHz
10^6 nodes	1000 sec.	333 sec.	100 sec.
10^7 nodes	27.8 hrs	9.3 hrs	2.78 hrs
10^8 nodes	$>$ 100 days	$>$ 35 days	$>$ 10 days
10^9 nodes	$>$ 10000 days	$>$ 3500 days	$>$ 1000 days

Table 1: The $O(n^2)$ problem in link prediction: Time required to allocate a *single machine cycle* to every node-pair possibility in networks of varying sizes and processors of various speeds.

Link prediction methods are often applied to very large networks, which are also sparse. The massive sizes of such networks can create challenges for the prediction process in spite of their sparsity. This is because the *search space* for the link prediction problem is of the size $O(n^2)$, where n is the number of nodes. Quadratic scalability can rapidly become untenable for larger networks. In fact, an often overlooked fact is that most *current link prediction algorithms evaluate the link propensities only over a subset of possibilities rather than exhaustively search for link propensities over the entire network, e.g., [23, 29].* In order to understand why this is the case, consider a network with 10^6 nodes. Note that a size such as 10^6 is not large at all by modern standards, and even common bibliographic networks such as DBLP now exceed this size. Even for this modest network, the number of *possibilities* for links is of the order of 10^{12}. Therefore, a 1GHz processor would require at least 10^3 seconds just to allocate one *machine cycle* to every pair of nodes. This implies that in order to determine the top-ranked link predictions over the *entire network*, the running time will be much larger than 10^3 seconds. It is instructive, therefore, to examine how this (lower bound on) running time scales with increasing network size. Table 1 shows the time requirements for allocating a single machine cycle to each pair-wise possibility. The running times in this table represent very optimistic lower bounds on the required times because link prediction algorithms are complex and require far more than a single machine cycle for processing a node-pair. Note that for larger networks, even the presented lower bounds on the running times are impractical.

It is noteworthy that most link prediction algorithms in the literature are not designed to search over the entire space of $O(n^2)$ possibilities. A closer examination of the relevant publications shows that even for networks of modest size, these algorithms perform benchmarking only by evaluating over a *sample of the possibilities* for links. This is only to be expected in light of the lower bounds shown in Table 1. In other words, the *complete ranking problem* for link prediction in very large networks remains largely unsolved

WSDM'16, February 22–25, 2016, San Francisco, CA, USA.

DOI: http://dx.doi.org/10.1145/2835776.2835815.

at least from a computational point of view. It is evident from the presented lower bounds in Table 1 that any ranking-based link prediction algorithm *must integrate search space pruning* within the prediction algorithm in order to even have any hope of exploring the $O(n^2)$ search space in a reasonable amount of time. The algorithmic design of most link prediction algorithms largely overlooks this basic requirement [7, 15].

The link prediction algorithms are classified into unsupervised and supervised methods. Unsupervised methods [13] typically use neighborhood measures such as the Adamic-Adar [1], and the Jaccard coefficient. Supervised methods [15] treat link prediction as a classification problem in which each node pair is treated as a test instance. Supervised methods are the state-of-the-art and generally provide more accurate results than unsupervised methods [15]. It is also noteworthy that most supervised methods evaluate link prediction algorithms by using only *a sample* of test links because of computational consideration. In real-world applications, it is often desirable to determine the *top-k* most relevant links for prediction *over all possibilities for test links*. This problem remains largely unsolved even for networks of any reasonable size.

The link prediction problem is also closely related to the missing value estimation problem, which is commonly used in collaborative filtering [3]. Just as collaborative filtering attempts to predict missing entries in a matrix of users and items, the link prediction problem attempts to predict missing entries in a node-node adjacency matrix. In fact, the missing value estimation framework seems to be a more compact and relevant model for the link prediction problem, as compared with the vanilla classification problem. Many of the modern methods for collaborative filtering use latent factor models [2, 17] such as SVD and NMF for predicting missing entries. These methods have been shown to be wildly successful at least within the domain of collaborative filtering [17]. In spite of the obvious similarity between link prediction and collaborative filtering and the obvious effectiveness of latent factor models, there are only a few methods [9] which attempt to use these models.

One of the reasons that latent factor models are rarely used in the link prediction domain is simply because of their complexity. In collaborative filtering applications, the item dimension is of the order of a few hundred thousand, whereas even the smallest networks in real-world settings contain more than a million nodes. Furthermore, collaborative filtering methods often perform the recommendation on a per-user basis rather than try to determine the top-k user-item pairs. The latter is particularly important in the context of link prediction. The factorization of a matrix of size $O(n^2)$ is not only computationally expensive, but also memory-intensive. As we will see later in this paper, one advantage of latent-factor models is that they are able to transform the adjacency matrix to a multidimensional space which can be searched efficiently *by pruning* large portions of the $O(n^2)$ search space in order to recommend the top-k possibilities. This is essential in such settings.

Contributions. In this paper, we explore an *ensemble-enabled approach* to achieving the aforementioned goals.

We show how to make latent factor models practical for link prediction by decomposing the search space into a set of smaller matrices. As a result, large parts of the $O(n^2)$ search space can be pruned without even having to evaluate the relevant node pairs. This provides an efficient approach for the top-k problem in link prediction. Furthermore, our problem decomposition method is an ensemble approach enabled with three structural bagging methods with performance guarantees, which has obvious *effectiveness* advantages. Note that the same bias-variance tradeoffs apply to the link-prediction problem, as they apply to the standard classification problem. Therefore, the use of an ensemble approach has obvious effectiveness advantages as well.

Using real-life datasets, we show that our ensemble-enabled approach for link prediction is both effective and efficient. For instance, (1) on Friendster with 15 million nodes and 1 billion edges, our approach could finish in an hour, while direct NMF, AA [1] and BIGCLAM [33] could not finish in a day, and (2) our approach improves the accuracy by $(18\%, 39\%, 33\%)$ (resp. $(4\%, 10\%, 18\%)$ and $(16\%, 11\%, 38\%)$) over direct NMF, AA and BIGCLAM on YouTube, Flickr and Wikipedia, respectively.

Organization. This paper is organized as follows. In the next section, we provide the basic framework for the approach and describe the efficient use of latent factor models for link prediction. Section 3 discusses how latent factor models can be augmented with ensembles to provide more effective and efficient results. Section 4 presents and discusses the experimental results, followed by related work in Section 5 and conclusions in Section 6.

2. LATENT FACTOR MODEL FOR SCALABLE LINK PREDICTION

We assume that $G = (N, A)$ is an undirected network containing node set N and edge set A. The node set N contains n nodes and the edge set A contains m edges. Furthermore, the $n \times n$ weight matrix $W = [w_{ij}]_{n \times n}$ contains the weights of the edges in A. The weight matrix is useful in representing the strengths of network connections in many real-world settings such as the number of publications between a pair of co-authors in a bibliographic network. The matrix is sparse, and many its entries are 0, which could be interpreted either as absence of links or as missing entries. While we assume that an undirected network is available, the approach can also be generalized to directed networks. The ranking problem for link prediction is formally stated as follows:

DEFINITION 1. *Given a network $G = (N, A)$ with node set N and edge set A, the ranking problem for link prediction is to determine the top-k node-pair recommendations such that these node pairs are not included in A.*

Note that this problem definition requires us to consider the entire search space of $O(n^2)$ possibilities, rather than a sample of the node pairs in the network.

Latent factor models work by associating a low dimensional factor with each row and column of the network. However, since link prediction is (predominantly) studied only for undirected networks, which have symmetric weight matrices, it suffices to associate an r-dimensional latent factor $\overline{F_i}$ with the ith node in the network. The value of r is the *rank* of the factorization. This is an issue, which we will discuss slightly later. The weight of a link between nodes i and j is defined by the use of the dot product between the factors of nodes i and j. In other words, for the weight matrix $W = [w_{ij}]_{n \times n}$, we would like to achieve the following:

$$w_{ij} \approx \overline{F_i} \cdot \overline{F_j}, \ \ \forall i, j \in \{1, \ldots, n\}. \tag{1}$$

This condition can be directly written in matrix form. Let F be an $n \times r$ matrix, in which the ith row is the row vector $\overline{F_i}$. Then, the aforementioned condition of Equation (1) can be written as follows:

$$W \approx F F^T. \tag{2}$$

An important question arises as to whether entries in the matrix W corresponding to the absence of links should be treated as incomplete entries or whether they should be treated as zero, with the possibility of being incorrect. When latent factor models are used in collaborative filtering, such entries are typically treated as

missing entries. However, unlike the absence of a rating, the absence of a link is indeed useful information *in the aggregate*, even though some node pairs have the propensity to form links *in the future*. Therefore, we argue that, unlike collaborative filtering, W should be treated as a completely specified matrix, but with noisy entries. Therefore, in the link prediction problem, latent factor models should be viewed as a way of *correcting* noisy entries with zero values, rather than strictly as a missing value estimation problem. Such assumptions also simplify the algorithmic development of latent factor models for link prediction. The idea here is that when we approximately factorize W into the form FF^T, the positive values of entries in FF^T can be viewed as the *predictions* of noisy 0-entries in W.

A second important question arises as to the choice of the latent factor model that must be used for prediction. There are many choices available for factorizing an adjacency matrix, especially when it is symmetric. Even a straightforward diagonalization of the matrix provides a reasonable factorization. We choose *non-negative* matrix factorization not only because of its interpretability advantages but also because it facilitates the $O(n^2)$ search phase of the prediction by providing a non-negative and sparse representation for each node.

We would like to determine the matrix F such that the Frobenius norm of $(W - FF^T)$ is minimized. This problem is referred to as symmetric NMF, and an efficient solution is proposed in [14], where F can be determined by starting with random nonnegative entries in $(0, 1)$, and using the following multiplicative update rule:

$$F_{ij} \leftarrow F_{ij}(1 - \beta + \beta \frac{(WF)_{ij}}{(FF^TF)_{ij}}), \tag{3}$$

in which β is a constant in $(0, 1]$ [28].

Discussions. Let us examine the computational complexity of the update Equation (3). The matrix FF^TF can be fully materialized with $O(r^2 \cdot n)$ matrix multiplications, and the matrix WF can be computed in $O(m \cdot r)$ multiplications by observing that the sparse matrix W has only $2m$ non-zero entries corresponding to the number of edges. Therefore, each update takes $O(n \cdot r^2 + m \cdot r)$ time.

This remains quite high for large networks, which motivates us to develop fast searching techniques to speed up the process.

2.1 Efficient Top-K Prediction Searching

An advantage of the nonnegative factorization approach is that it enables an efficient search phase, which is generally not possible with most other link prediction methods. The value of $\overline{F_i} \cdot \overline{F_j}$ in Equation (1) provides a prediction value for the links. The goal of the search phase is to return the top-k links with the largest prediction values. In real-world settings, the matrix F is often nonnegative and sparse. This non-negativity and sparsity are particularly useful in enabling an efficient approach. In order to speed up the search, we define the notion of ϵ-approximate top-k predictions, denoted as top-(ϵ, k) predictions.

DEFINITION 2 (TOP-(ϵ, k) PREDICTIONS). *A set L of predicted links is a top-(ϵ, k) prediction, if the cardinality of L is k, and the k-th best value of $\overline{F_i} \cdot \overline{F_j}$ for a link $(i, j) \in L$ is at most ϵ less than the k-th best value of $\overline{F_h} \cdot \overline{F_l}$ over any link (h, l) in the network.*

Intuitively, this definition allows a qualitative tolerance of ϵ in the top-k returned links. Allowing such a tolerance significantly helps in speeding up the search process by pruning large portions of the search space, which is particularly important in an $O(n^2)$ search space of link predictions.

The first step is to create a new $n \times r$ matrix S, which is obtained by sorting the columns of F in a descending order. An inverted list is associated with each of the r latent variables containing the node identifiers of F arranged according to the sorted order of S. The r inverted lists can also be represented as an $n \times r$ matrix R. Let the number of rows in the p-th column of S ($p \in [1, r]$), for which the value of S_{ip} is greater than $\sqrt{\epsilon/r}$ be f_p, and for which the value of S_{ip} is greater than 0 be f'_p, respectively.

Then the following nested loop is executed for each (say p-th) column of S:

for each $i = 1$ to f_p **do**
  for each $j = i + 1$ to f'_p **do**
    if $S_{ip} \cdot S_{jp} < \epsilon/r$ **then**
      break inner loop;
    else increase the score of node-pair (R_{ip}, R_{jp}) by an amount of $S_{ip} \cdot S_{jp}$
  endfor
endfor

This nested loop is designed to track the relevant subset of node pairs from which one can determine the top-(ϵ, k) predictions. The nested loop typically requires much less time than $O(n^2)$ time because large portions of the search space are pruned. First, depending on the value of ϵ, the value of f_p is much less than n. This is particularly true if many entries of the factorized matrix F are close to 0. Furthermore, the inner loop is often terminated early. The value of ϵ therefore provides the user a way to set the trade-off between accuracy and efficiency. A hash-table is maintained which tracks all the pairs (R_{ip}, R_{jp}) encountered so far in the nested loop. Because of the pruning, the hash table usually needs to track a miniscule set of the $O(n^2)$ node-pairs in order to determine the ones that truly satisfy the top-(ϵ, k) requirement. In the process, we exclude the links which have already been represented with non-zero entries in W because such links are always likely to have the largest prediction values, which further reduces the searching space.

It remains to show that this procedure truly does find a valid set of top-(ϵ, k) link predictions. The reason that the procedure works correctly and efficiently is because of nonnegativity and sparsity.

PROPOSITION 1. *The nested loop method finds a valid set of top-(ϵ, k) predictions.*

PROOF. The main part of the proof is to show that any dot product is underestimated by at most ϵ. The aforementioned pseudocode containing the nested loop is executed r times, once for each latent component. Therefore, it suffices to show that the contribution of each nested loop is underestimated by at most ϵ/r. There are two sources of underestimation:

1. The outer loop does not consider rows i for which $S_{ip} < \sqrt{\epsilon/r}$. This effectively prunes the products between pairs (i, j) for which both S_{ip} and S_{jp} are less than $\sqrt{\epsilon/r}$. Therefore, the underestimation because of the ignoring of this pair is at most $\sqrt{\epsilon/r} \times \sqrt{\epsilon/r} = \epsilon/r$.

2. The second case is when the inner loop is terminated early. The early termination condition in this case is that the product is at most ϵ/r.

Therefore, in both these mutually exclusive cases, the underestimation is at most ϵ/r. Therefore, over all latent components the aggregate underestimation is at most $(\epsilon/r) \times r = \epsilon$.

This completes the proof. □

Discussions. While the basic matrix factorization method is able to allow us to provide efficiency and pruning to the search process, it is still not quite as fast as one may need for large networks. The

main problem arises as a result of the factorization process itself, which can require as much as $O(r \cdot (m+n \cdot r))$. Typically, the number r of latent factors varies from the orders of a few ten to a few hundred [30, 31]. For sparse networks, in which the node degree is less than r, the $O(nr^2)$ term might be the bottleneck. The required number of latent components r is often expected to increase with network size. In order to handle this computational problem, we propose the method of *ensemble decomposition*, which provides both efficiency *and* effectiveness advantages.

3. STRUCTURAL BAGGING METHODS

Since the link prediction problem scales worse than linearly with the network size, it is generally more efficient to solve smaller problems multiple times rather than solve a single large problem. The structural bagging approach provides an effective method to decompose the link prediction problem into smaller pieces that are solved independently. Furthermore, the aggregated results from multiple models often provide a robustness to the decomposition process [38]. In the following, we introduce three different ways for the bagging decomposition.

3.1 Random Node Bagging

Random node bagging is the simplest form of structural bagging. The basic idea of random node bagging is to iteratively apply the following three steps:

(1) Select a random set of nodes N_r from the graph G corresponding to a fraction f of the nodes in the network. Determine the node set $N_s \supseteq N_r$, corresponding to all nodes adjacent to N_r.

(2) Construct a reduced adjacency matrix W_s from the node set N_s by using the subgraph induced on N_s of G, referred to as an ensemble component or simply an ensemble, to select the relevant $|N_s|$ rows and columns of W.

(3) Apply the symmetric NMF method in Section 2 to the reduced matrix W_s. Use the pruning search process in Section 2.1 to determine the predictions of all pairs of nodes of N_s.

The main efficiency advantage of this approach is because of the smaller sizes of the matrices in the factorization. Furthermore, because of the smaller size of the induced subgraph in each ensemble component, the number of latent factors r, which is required, is also smaller. This will generally translate into efficiency advantages. In many cases, when the number of nodes is very large, it may be impractical to solve the entire problem in main memory. In such cases, the use of ensemble approach decomposes the problem into smaller memory-resident components.

The main problem with random node sampling is that it does not attempt to sample more *relevant* regions of the network which are more likely to contain possible links. Other forms of sampling are likely to be more effective in this context.

3.2 Edge Bagging

Edge bagging is designed to sample more *relevant* regions of the graph. After all, real-world networks are sparse and most of the $O(n^2)$ possibilities for edges are usually not populated. By sampling densely populated regions of the network, many node pairs will not be considered at all, but these node pairs are often not relevant to begin with.

The edge bagging approach proceeds as follows:

(1) Let N_s be a node set containing a single randomly chosen node. Nodes which are adjacent to N_s are randomly selected and added to N_s. In the event that no node is adjacent to N_s, a randomly chosen node from a different connected component is added to N_s. The procedure is repeated until N_s contains at least a fraction f of the total number of nodes.

(2) Construct a reduced adjacency matrix W_s from the node set N_s by using the subgraph induced on N_s of G, i.e., an ensemble, to select the relevant $|N_s|$ rows and columns of W.

(3) Apply the symmetric NMF method in Section 2 to the reduced matrix W_s. Use the pruning search process in Section 2.1 to determine the predictions of all pairs of nodes.

This method of growing the sampled node set with edge sampling is likely to select dense components from the network. Such dense components are more likely to contain random node pairs. Unlike the previous case where every node pair is considered with high probability, many node pairs will not be considered at all. However, such node pairs will typically not be present in the same dense component. Therefore, such node pairs are likely to be irrelevant, and in this way the approach already prunes unimportant node pairs during the process of ensemble construction.

3.3 Biased Edge Bagging

While the edge bagging procedure is effective at discovering dense components, it does have a drawback. Its main drawback is that it selectively includes nodes with high degrees within the resulting components. Therefore, the same high-degree nodes are very likely to be included in all the ensemble components. As a result, it often becomes more difficult to make robust predictions between low-degree nodes.

In biased edge bagging, exactly the same procedure is used as the case of edge bagging. The only difference is that when the node set N_s is grown, a random adjacent node is not selected. Rather, an adjacent node, which was selected the least number of times in previous ensemble components, is used. Ties are broken randomly.

This approach ensures that each node is selected with an approximately similar number of times across various ensemble components, and it prevents the repeated selection of high-degree nodes. Note that the bias in the edge bagging process makes that the vast majority node pairs will not be considered. However, such node pairs will usually be in components that are not as well connected. Therefore, such nodes are far less likely to form links. In most practical applications, one only needs to recommend a small number of node pairs for prediction. Therefore, it is reasonable to ignore such node pairs in the prediction process.

3.4 Using Link Prediction Characteristics

Different from existing sampling methods [32], our bagging methods should be designed in particular for link prediction. Motivated by the observation that most of all new links in social networks span within very short distances, typically closing triangles, which has been justified in [37], we develop three structural bagging approaches such that *a node is always sampled together with all its neighbors, which guarantees the possibility of forming triangles*. To achieve this, we revise the previous bagging methods as follows.

(1) For random node bagging, when a node is selected uniformly at random from the network G, the node together with all of its neighbors are put into the node set N_s.

(2) For edge and biased edge bagging, when a node adjacent to N_s is selected and added to N_s, all of its neighbors are put into the node set N_s together.

3.5 Bound of Node Bagging Ensembles

Observe that even each ensemble component is much smaller, multiple samples are required. To meaningfully rank the various node pairs, each node pair needs to be included in the ensemble components with performance guarantees. What is the required number of samples to ensure that each node pair is included at least μ times? Clearly, this number depends on the sampling fraction f. We next present a probabilistic bound on the expected number of times that a node pair is included as follows.

PROPOSITION 2. *The expected times of each node pair included in μ/f^2 ensemble components is at least μ.*

PROOF. Since each ensemble component includes a node with probability at least f, it follows that each node pair is included with probability f^2. Furthermore, all ensemble component are independent of each other. Let X be the times of each node pair is included in all ensemble components, and the expected value $E(X)$ of X is equal to $b \times f^2$, where b is the number of ensemble components. For $E(X) \geq \mu$, we have $b \geq \mu/f^2$. □

Note that the above bound holds only for the original random node bagging method, and it provides a theoretical guarantee. Indeed, we could do much better in practice. For instance, the setting of $\mu = 0.1$ and $f = 0.1$ already achieves a pretty good result, as shown by our experimental study in Section 4.

3.6 Ensemble Enabled Top-K Predictions

We now explain the complete framework for top-k predictions enabled with ensembles, shown as follows.

given network $G(N, A)$ and parameters μ and f.
let Γ be empty;
repeat μ/f^2 times **do**
  let N_s be a sampled ensemble of G with at least $f \cdot n$ nodes;
  Compute $W_s \approx F_s \cdot F_s^T$ using NMF;
  let Γ' be top-k largest value node pairs (i, j) in $\{\overline{F_{s,i}} \cdot \overline{F_{s,j}}\}$;
  let Γ be top-k largest value node pairs (i, j) in $\Gamma' \cup \Gamma$;
return the top-k node pairs Γ not included in A.

To ensure that a node pair appears in the ensemble components at least μ expected times, μ/f^2 ensemble components are considered in total. For each time, an ensemble component N_s is sampled by one of the above node, edge and biased edge bagging methods, the symmetric NMF method in Section 2 is used to the reduced matrix W_s, and the aforementioned pruning search process in Section 2.1 is used to determine the predictions of all pairs of nodes. If a node pair appears in multiple ensemble components and has multiple predicted values, the maximum predicted value is considered. And, hence, only the top-k predicted links are maintained for each ensemble component. At the end of the process, the top-k predictions in all μ/f^2 ensemble components are returned.

4. EXPERIMENTAL STUDY

We next present an extensive experimental study of our ensemble-enabled approach for link prediction. Using real-life datasets, we conducted two sets of experiments to evaluate: (1) the effectiveness and efficiency of our approach vs. conventional methods AA [1] and BIGCLAM [33] and (2) the impacts of various factors.

4.1 Experimental settings

We first present our experimental settings.

Datasets. We used the following real-life network datasets, which are from the Koblenz Network Collection [1].

[1]http://konect.uni-koblenz.de/

Datasets	Date	Nodes	Edges
YouTube	2006-12-09 — 2007-02-22	1,503,841	3,691,893
	2007-02-23 — 2007-07-22	1,503,841	806,213
Flickr	2006-11-01 — 2006-11-30	1,580,291	13,341,698
	2006-12-01 — 2007-05-17	1,580,291	3,942,599
Wikipedia	2001-02-19 — 2006-10-31	1,682,759	28,100,011
	2006-11-01 — 2007-04-05	1,682,759	5,856,896

Table 2: Training and ground truth data. The data in the first time slot is the training data and the remaining is the ground truth data.

Parameters	Descriptions	Default Values
β	Coefficient in NMF update rule	0.5
$iter$	Number of iterations for NMF	50
r	Number of latent factors	10
ϵ	Tolerance of top-(ϵ, k) prediction	1
k	Number of links returned by top-(ϵ, k) prediction	1×10^5 / 1×10^6
μ	Expected appearing times of each node pair in ensemble components	0.1
f	Fraction of the number of nodes to be selected for an ensemble component	0.1
n	Number of nodes	See Table 2

Table 3: Parameters used in the experiments. Note that we set $k = 1 \times 10^5$ for YouTube and $k = 1 \times 10^6$ for other datasets.

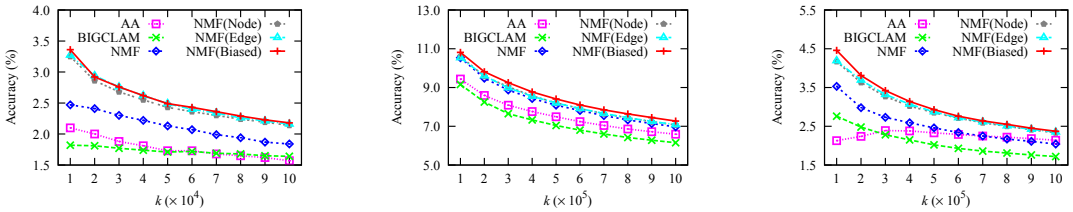
(1) YouTube is a 7 month friendship network of YouTube users with $3,223,589$ nodes and $9,375,374$ undirected edges.
(2) Flickr is a 6 month friendship connections of Flickr users with $2,302,925$ nodes and $33,140,017$ directed edges.
(3) Wikipedia is a 6 year hyperlink network of the English Wikipedia with $1,870,709$ nodes and $39,953,145$ directed edges.
(4) Twitter is the follower network from Twitter with $41,652,230$ nodes and $1,468,365,182$ directed edges.
(5) Friendster is the friendship network of the Friendster with $68,349,466$ nodes and $2,586,147,869$ directed edges.

(1) YouTube, Flickr and Wikipedia contain timestamps of edge arrivals. For each of these datasets, the latest five month part is treated as its ground truth data for testing the accuracy, and the remaining part is treated as its training data, shown in Table 2. To test the scalability, we further generated five subnetworks with increasing sizes for each dataset, using the breadth first search started from the node with the largest degree. (2) Twitter and Friendster do not have timestamps, and are only used for the scalability test. (3) It does not make much sense to predict links for users who appear in the ground truth data, but not in the training data. Hence, we removed these users from the ground truth data. Moreover, since our link prediction methods focus on predicting links on undirected graphs, we ignored the direction of edges in the directed graphs.

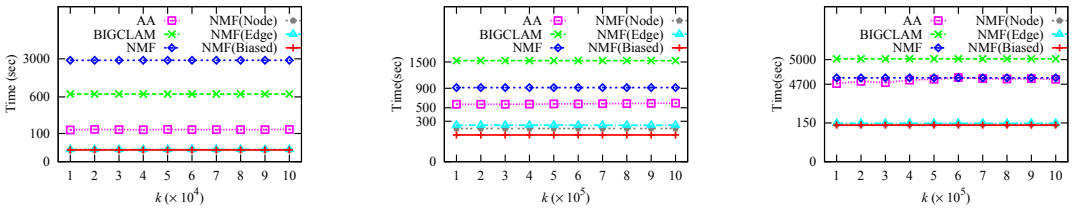
Algorithms for comparison. We have carefully chosen a couple of algorithms to compare with our ensemble-enabled approach.
(1) Adamic/Adar (AA) [1]: AA is a popular neighborhood based method that produces a score for each link (u, v), defined as below:

$$score(u, v) = \sum_{z \in N(u) \cap N(v)} \frac{1}{log|N(z)|},$$

where $N(u)$ is the set of neighbors of node u. Lu [35] showed that AA performs well on a range of networks because it only concerns 2-hop neighbors and reduces much of search space. Therefore, we implemented a top-k link prediction method by searching the k largest AA score links. The complexity of this method is $O(nd^2log(k))$, where d is the average degree of networks. Another popular link prediction method is Katz [34], which is based on the ensemble of all paths. However, its complexity is $O(n^3)$,

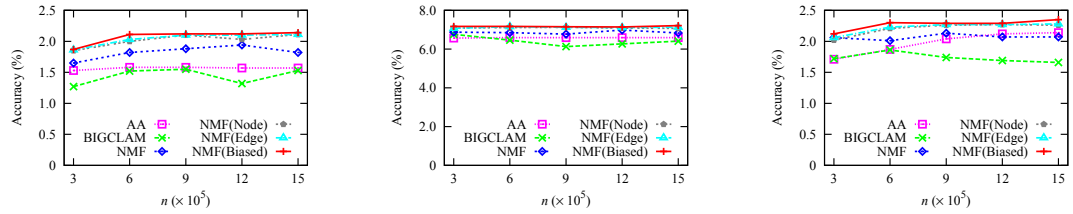


(a) YouTube (b) Flickr (c) Wikipedia

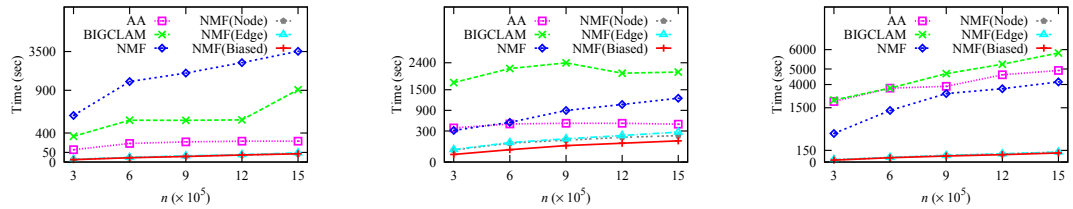


(d) YouTube (e) Flickr (f) Wikipedia

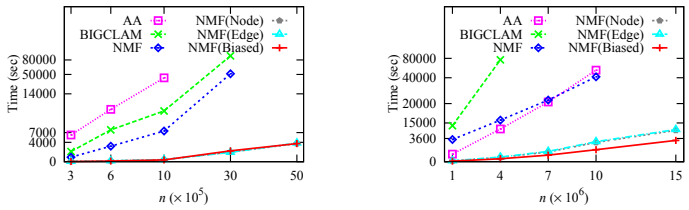
Figure 1: Accuracy and efficiency comparison: with respect to the number k of predicted links.



(a) YouTube (b) Flickr (c) Wikipedia



(d) YouTube (e) Flickr (f) Wikipedia



(g) Twitter (h) Friendster

Figure 2: Accuracy and efficiency comparison: with respect to the network sizes.

and does not work on large networks with millions of nodes. Thus, we did not choose Katz for comparison in the experiments.

(2) Cluster Affiliation Model for Big Networks (BIGCLAM) [33]: Yang and Leskovec developed this probabilistic generative model for networks based on community affiliations. An ingredient of BIGCLAM is based on the fact that, when people share multiple community affiliations, the links between them stem for one dominant reason. This means that more communities a node pair shares, the higher the probability of the node pair being connected is. Let F be a nonnegative matrix where F_{uc} is the degree of the node u belongs to the community c. Give F, the BIGCLAM generates a graph $G(N, A)$ by creating edge (u, v) between a pair of nodes $u, v \in N$ with the probability

$$p(u, v) = 1 - exp(-F_u \cdot F_v),$$

where F_u is a weight vector for node u. Viewing the probability $p(u, v)$ as a score for the link (u, v), it is reasonable to predict links based on BIGCLAM. The complexity of BIGCLAM is $O(nd(r + d))$, where d is the average degree of networks. In addition, this

model is not designed to search the entire space of $O(n^2)$, and we revised it by our top-(ϵ, k) method to predict links.

Implementation. We implemented all algorithms including AA, BIGCLAM, link prediction method in Section 2 (NMF), NMF with random node bagging (NMF(Node)), NMF with edge bagging (NMF(Edge)) and NMF with biased edge bagging (NMF(Biased)) using C/C++ with no parallelization.

All experiments were conducted on a machine with 2 Intel Xeon E5–2630 2.4GHz CPUs and 64 GB of Memory, running 64 bit Windows 7 professional system. Each experiment was repeated 5 times, and the average is reported here.

4.2 Experimental Results

We next present our findings. In all the experiments, we fixed r to $(50, 30, 50)$ (resp. $(40, 20, 20)$) for NMF (resp. BIGCLAM) on YouTube, Flickr and Wikipedia, respectively. For three bagging methods, we fixed $r = 10$ by default (See Exp-2.3 for more details about the setting of r). The other parameters with their descriptions and default values are presented in Table 3.

4.2.1 Comparison with AA and BIGCLAM

In the first set of tests, we evaluated the effectiveness and efficiency of our methods compared with AA and BIGCLAM. Given one of top-k link prediction methods, denoted by x, its effectiveness of link prediction is evaluated with the following measure:

$$accuracy(x) = \frac{\#\text{ of correctly predicted links}}{\text{the number } k \text{ of predicted links}}.$$

Exp-1.1: Impacts of k. To evaluate the impacts of the number k of predicted links, we varied k from 1×10^4 to 1×10^5 on YouTube (resp. from 1×10^5 to 1×10^6 on Flickr and Wikipedia) and fixed other parameters to their default values. The results of accuracy and running time are reported in Figure 1(a), 1(b) and 1(c) and Figure 1(d), 1(e) and 1(f), respectively.

The accuracy results tell us that (a) NMF(Biased) outperforms other methods on all datasets, (b) three bagging methods always have higher accuracy than NMF, AA and BIGCLAM, (c) NMF is more accurate than AA and BIGCLAM, and (d) the accuracy of all methods decreases with the increment of k. Indeed, NMF(Biased) improves the accuracy by $(18\%, 39\%, 33\%)$ (resp. $(4\%, 10\%, 18\%)$ and $(16\%, 11\%, 38\%)$) over NMF, AA and BIGCLAM on YouTube, Flickr and Wikipedia, respectively. This verifies the effectiveness of our bagging methods.

The running time results tell us that (a) three bagging methods are much faster than NMF, AA and BIGCLAM, (b) the running time of all methods is insensitive to the increase of k, except AA whose complexity is $O(nd^2log(k))$. Indeed, the three bagging methods finished the prediction in 300 seconds on the three datasets. Furthermore, NMF(Biased) is $(70, 2.7, 14)$ (resp. $(4.5, 2.7, 8)$ and $(33, 33, 36)$) times faster than NMF, AA and BIGCLAM on YouTube, Flickr and Wikipedia, respectively. This verifies the efficiency of our bagging methods.

Note that the accuracy of all methods is not high, even the best accuracy (NMF(Biased) on Flickr when $k = 1 \times 10^5$) is less than 12%. The reason is that there are more than 1×10^{12} possible links in the search space of each dataset, but less than 1×10^7 links in the ground truth. In addition, NMF is slower than AA on three datasets because r is fixed to at least 30, which is consistent to the $O(nr^2)$ complexity of NMF. The running time of AA is about 100 seconds on YouTube, while more than 500 seconds on Flickr and 4700 seconds on Wikipedia. This is because that the average degree of these datasets are 5, 17 and 33, and AA takes more time with the increase of the degree of networks. The running time of BIGCLAM is also sensitive to the degree of networks because its complexity is $O(nd(r+d))$. As a result, it runs faster than NMF on YouTube when the degree is 5, while it takes more time on Flickr and Wikipedia when the degree is increased.

Exp-1.2: Impacts of network sizes. To evaluate the impacts of network sizes, we varied the number of nodes n from 3×10^5 to 1.5×10^6 on YouTube, Flickr and Wikipedia (resp. from 3×10^5 to 5×10^6 on Twitter and from 1×10^6 to 1.5×10^7 on Friendster). Since Twitter and Friendster do not contain ground truth for choosing the value of r, we fixed r on these datasets to the average value of its default values. Hence, on these two datasets, we fixed $r = 43$ for NMF (resp. $r = 27$ for BIGCLAM and $r = 10$ for bagging methods). The other parameters are fixed to their default values. The results of accuracy and running time are reported in Figure 2(a), 2(b) and 2(c) and Figure 2(d), 2(e), 2(f), 2(g) and 2(h), respectively. Note that there are *some missing plots* for NMF, AA and BIGCLAM in the Figure 2(g) and 2(h) because their running time is beyond 24 hours.

The accuracy results tell us that (a) NMF(Biased) obtains the highest accuracy on all datasets, (b) the three bagging methods perform better than NMF, AA and BIGCLAM, and (c) NMF has higher accuracy than AA and BIGCLAM. This means that our methods are accurate and robust with the increase of network sizes.

The running time results tell us that (a) bagging methods are much faster than other methods, (b) the running time of all methods increase nearly linearly with the increase of n. For instance, NMF(Biased) speeds up NMF, AA and BIGCLAM for around $(20, 107, 43)$ (resp. $(31, 21, 175)$) times on Twitter (resp. Friendster) and is thus essential for making our bagging methods scalable to large networks. Note that NMF is slower than BIGCLAM on YouTube but faster on other datasets because BIGCLAM requires more time with the increase of the degree of networks, which is consistent with the complexity analysis. In addition, the running time of BIGCLAM fluctuates when n between 6×10^5 and 1.2×10^6 on YouTube and Flickr because the number of the iterations for its nonnegative matrix factorization method is not fixed.

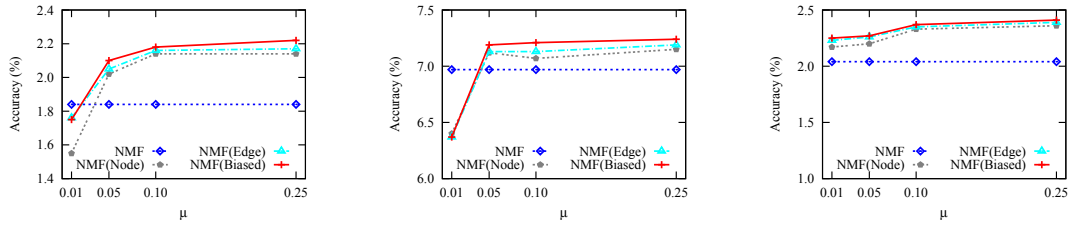
4.2.2 Impacts of Various Parameters

In the second set of tests, we evaluated the impacts of parameters on the accuracy and running time of NMF, NMF(Node), NMF(Edge), NMF(Biased) and BIGCLAM. We first tested the impacts of μ and f in bagging methods. We then tested the impacts of r and ϵ.

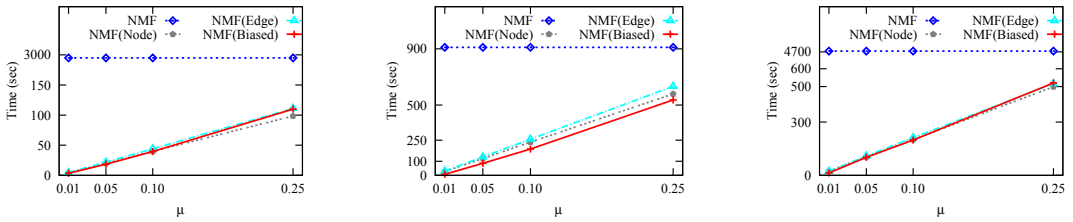
Exp-2.1: Impacts of μ. To evaluate the impacts of μ, we varied μ from 0.01 to 0.25 and fixed other parameters to their default values. The accuracy and running time results are reported in Figure 3(a), 3(b) and 3(c) and Figure 3(d), 3(e) and 3(f), respectively. We also plotted the accuracy and running time of NMF for comparison.

The results tell us that (a) bagging methods have higher accuracy than NMF when μ is large enough, (b) the accuracy of bagging methods increases with the increment of μ and becomes stable when μ is greater than 0.1. This means that the accuracy of bagging methods would increase and become stable with the increase of the number of ensemble components. Furthermore, (c) the running time of bagging methods increases linearly with the increase of μ since μ/f^2 ensemble components had been generated in each bagging method. Note that, when $\mu = 0.1$, the accuracy of bagging methods is becoming stable and the running time of them is less than NMF. Therefore, we fixed $\mu = 0.1$ by default.

Exp-2.2: Impacts of f. To evaluate the impacts of f, we varied f from 0.02 to 0.5 and fixed other parameters to their default values. The accuracy and running time results are reported in Figure 4(a), 4(b) and 4(c) and Figure 4(d), 4(e) and 4(f), respectively.

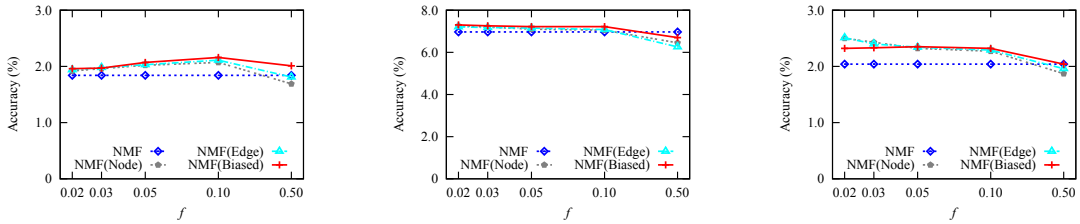


(a) YouTube (b) Flickr (c) Wikipedia

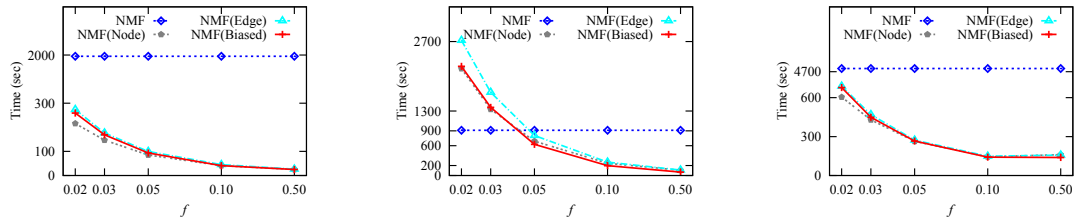


(d) YouTube (e) Flickr (f) Wikipedia

Figure 3: Accuracy and efficiency comparison: with respect to the expected appearing times μ.



(a) YouTube (b) Flickr (c) Wikipedia



(d) YouTube (e) Flickr (f) Wikipedia

Figure 4: Accuracy and efficiency comparison: with respect to the fraction f

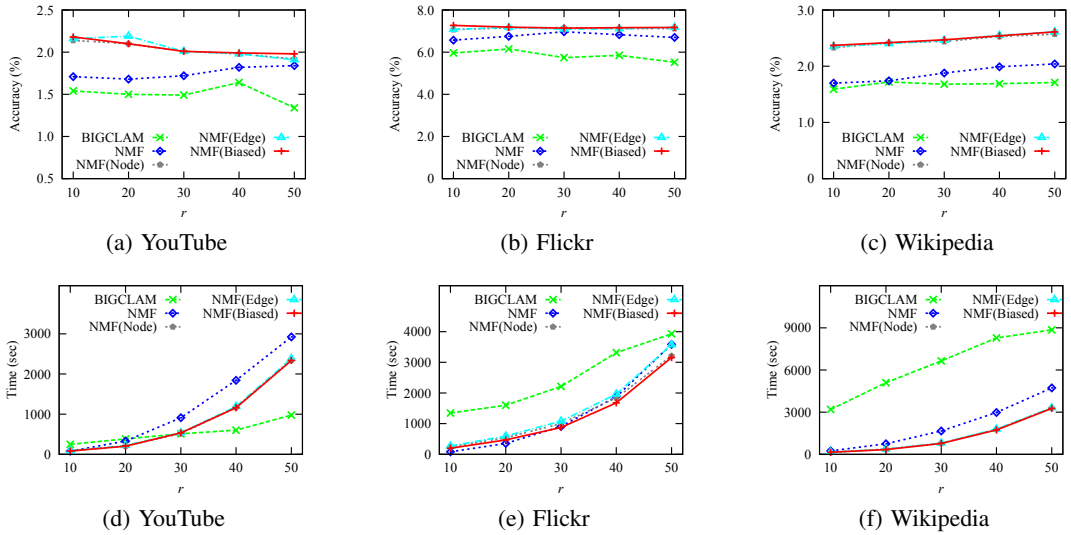
The results tell us that (a) bagging methods have higher accuracy than NMF when f between 0.02 and 0.1, (b) the accuracy of bagging methods decreases with the increase of f, and (c) the running time of bagging methods increases nearly linearly with the decrease of f. Note that the complexity of bagging methods is $O((n_1r^2 + n_1d_1r) * \mu/f^2)$, where $n_1 = nf$ and d_1 is the average degree of each ensemble component. Since bagging methods are likely to select dense components, d_1 may be greater than r and the complexity is $O(nd_1r\mu/f)$. Thus, the running time of bagging methods increases with the decrease of f. Keeping the accuracy of bagging methods better than that of NMF, we fixed $f = 0.1$ by default to achieve a better efficiency.

Exp-2.3: Impacts of r. To evaluate the impacts of r, we varied r from 10 to 50 and fixed other parameters to their default values. The accuracy and running time results are reported in Figure 5(a), 5(b) and 5(c) and Figure 5(d), 5(e) and 5(f), respectively.

The results tell us that (a) NMF(Biased) obtains the best accuracy compared with other methods, (b) bagging methods have higher accuracy than NMF and BIGCLAM, (c) the accuracy of NMF increases slightly with the increase of r and is always greater than that of BIGCLAM, and (d) the running time of NMF is increased quadratically with the increase of r since its complexity is $O(nr^2)$. A similar trend of running time is also found for bagging methods. To obtain the highest accuracy, we fixed r to $(50, 30, 50)$ (resp. $(40, 20, 20)$) for NMF (resp. BIGCLAM) on YouTube, Flickr and Wikipedia, respectively. Note that, when $r = 10$, the accuracy of bagging methods is greater than the best accuracy of NMF. Hence, we fixed $r = 10$ by default for bagging methods.

Exp-2.4: Impacts of ϵ. To evaluate the impacts of ϵ, we varied ϵ from 0.5 to 1.0 and fixed other parameters to their default values. The accuracy and running time results are reported in Figure 6(a), 6(b) and 6(c) and Figure 6(d), 6(e) and 6(f), respectively.

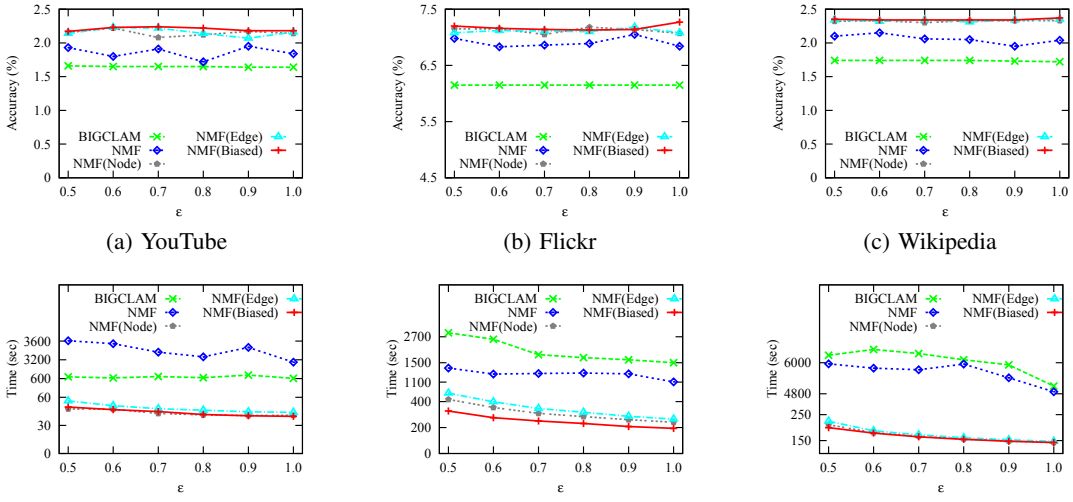
The results tell us that (a) the accuracy of all methods is stable with the increase of ϵ, which means that the accuracy of our methods is insensitive to ϵ, and (b) the running time of all methods is decreased with the increase of ϵ because the larger ϵ reduces more search space. Thus, our top-(ϵ, k) method is reasonable for link prediction. Since the accuracy is insensitive to ϵ, we fixed $\epsilon = 1$ by default to achieve a better efficiency.



(a) YouTube (b) Flickr (c) Wikipedia

(d) YouTube (e) Flickr (f) Wikipedia

Figure 5: Accuracy and efficiency comparison: with respect to the number r of latent factors.



(a) YouTube (b) Flickr (c) Wikipedia

(d) YouTube (e) Flickr (f) Wikipedia

Figure 6: Accuracy and efficiency comparison: with respect to the tolerance ϵ of top-(ϵ, k) prediction.

Summary. From these experimental results on real-life social network datasets, we find the following. (1) NMF is able to predict links and can be sped up by the top-(ϵ, k) process to explore the $O(n^2)$ search space. It is more accurate than AA and BIGCLAM, which verifies the effectiveness of NMF. Moreover, NMF runs faster than BIGCLAM on dense networks, e.g., Flickr and Wikipedia, which is consistent with the complexity analysis that BIGCLAM is sensitive to the degree of networks. (2) However, the running time of NMF is increased quadratically with the increase of r. This might be a bottleneck for large networks. By decomposing the link prediction problem into smaller pieces and solving them independently, our bagging methods are more efficient and scale well with the size and density of large networks, e.g., NMF(Biased) finished in an hour on Friendster with 1.5×10^7 nodes and 1.0×10^9 edges, while NMF, AA and BIGCLAM could not finish in a day. Further, NMF(Biased) speeds up NMF, AA and BIGCLAM for around $(20, 107, 43)$ (resp. $(31, 21, 175)$) times on Twitter (resp. Friendster). (3) Combining link prediction characteristics, our bagging methods also provide the accuracy advantages for link prediction, e.g., NMF(Biased) is faster than the other methods, and improves the accuracy by $(18\%, 39\%, 33\%)$ (resp. $(4\%, 10\%, 18\%)$ and $(16\%, 11\%, 38\%)$) over NMF, AA and BIGCLAM on YouTube, Flickr and Wikipedia, respectively.

5. RELATED WORK

The link prediction problem has been studied extensively in the data mining and machine learning community [13, 35], which falls into unsupervised and supervised methods [15]. Unsupervised methods often assign scores to potential links based on the topology of the given graphs: (a) Adamic/Adar [1] is a common neighbor based method; (b) Katz [34] is a path based method which sums over all paths between two nodes, and there are also other path based methods, such as Local Path and Random Walk with Restart [35]; And (c) [12, 13] investigates the low rank approximation methods by generating a small rank matrix to approximate the initial adjacency matrix. Supervised methods [8, 15, 16] typically treat link prediction as a classification problem, e.g., supervised matrix factorization and random walk based approaches [5, 9]

Recently, several models for link prediction have been proposed, such as community affiliation models [25,33], stochastic topic mod-

els [39], negative link prediction models [40], statistical relational models [6, 10, 11, 22, 26] and Markov models [27]. Moreover, link prediction has also been studied for mining missing hyperlinks [4,41]. While some recent work has focused on the heterogeneous [18–21,24] and temporal [5,23] scenarios, these methods are not essentially designed to search the entire space of $O(n^2)$ possibilities. Indeed, they are often not able to prune the search space of possibilities, and are mostly designed to evaluate the link prediction propensities of a subset of node pairs.

Our method is related to NMF proposed in [30], which has been successfully used for collaborative filtering [17]. Since the adjacency matrix in our approach is symmetric, we adopt the symmetric NMF method [28]. Our work is also related to bagging predictor [38] that generates an aggregated predictor based on multiple bootstrap samples. Different from the bootstrap sampling methods, we focus on sampling subgraphs from large networks. Although a variety of graph sampling techniques were introduced in [32], our approach combines link prediction characteristics [37] with graph sampling methods to achieve high link prediction accuracy.

6. CONCLUSIONS AND FUTURE WORK

We have proposed an ensemble-enabled approach for top-k link prediction, which scales up link prediction on very large social networks. By decomposing a large network into smaller pieces, the bagging methods are more scalable to large networks with over 15 million nodes and 1 billion edges. We develop three bagging methods that are designed in particular for link prediction, and our bagging methods also provide better accuracy and scalability. Finally, we have experimentally verified that our ensemble-enabled approach is much more accurate and scalable than existing methods AA [1] and BIGCLAM [33].

A couple of topics need further investigation. First, we are to develop distributed approaches scalable on networks with billions of nodes, in a way similar to [31]. Second, we are to study personalized recommendations using our link prediction approach.

Acknowledgments. This work is supported in part by 973 program (No. 2014CB340300), NSFC (No. 61322207) and Special Funds of Beijing Municipal Science & Technology Commission. For any correspondence, please refer to Shuai Ma.

7. REFERENCES

[1] L. Adamic and E. Adar. Friends and neighbors on the web. *Social Networks*, 25, pp. 211–230, 2001.

[2] C. Aggarwal and S. Parthasarathy. Mining massively incomplete data sets by conceptual reconstruction. *KDD*, 2001.

[3] G. Adomavicius, and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 17(6), pp. 734–749, 2005.

[4] S. F. Adafre and M. Rijke. Discovering missing links in Wikipedia. *KDD Workshop on Link Discovery*, 2005.

[5] L. Backstrom, and J. Leskovec. Supervised random walks: predicting and recommending links in social networks. *WSDM*, 2011.

[6] M. Bilgic, G. Namata and L. Getoor. Combining collective classification and link prediction. *ICDM Workshop on Mining Graphs and Complex Structures*, 2007.

[7] L. Getoor and C. Diehl. Link mining: A survey. *SIGKDD Exploration*, pp. 3–12, 2005.

[8] J. R. Doppa, J. Yu, P. Tadepalli and L. Getoor. Chance constrained programs for link prediction. *NIPS Workshop on Analyzing Networks and Learning with Graphs*, 2009.

[9] A. K. Menon, and C. Elkan. Link prediction via matrix factorization. *Machine Learning and Knowledge Discovery in Databases*, pp. 437–452, 2011.

[10] L. Getoor, N. Friedman, D. Koller and B. Taskar. Learning probabilistic models of relational structure. *ICML*, 2001.

[11] L. Getoor, N. Friedman, D. Koller and B. Taskar. Learning probabilistic models of link structure. *Journal of Machine Learning Research*, 3, pp. 679–707, 2002.

[12] J. Kunegis and A. Lommatzsch. Learning Spectral Graph Transformations for Link Prediction. *ICML*, 2009.

[13] D. Liben-Nowell and J. Kleinberg. The link prediction problem for social networks. *Journal of the American society for information science and technology*, 58(7), pp. 1019–1031, 2007.

[14] B. Long, Z. Zhang, and P. Yu. Co-clustering by block value decomposition. *KDD*, 2005.

[15] R. Lichtenwalter, J. Lussier, and N. Chawla. New perspectives and methods in link prediction. *KDD*, 2010.

[16] R. Lichenwater and N. Chawla. Vertex Collocation Profiles: Subgraph Counting for Link Analysis and Prediction. *WWW*, 2012.

[17] B. Liu. Web Data Mining. *Springer*, 2010.

[18] G. Qi, C. Aggarwal, and T. Huang. Link Prediction across Networks by Cross-Network Biased Sampling. *ICDE*, 2013.

[19] Y. Sun, R. Barber, M. Gupta, C. Aggarwal, J. Han. Co-author Relationship Prediction in Heterogeneous Bibliographic Networks. *ASONAM*, 2011.

[20] Y. Sun, J. Han, C. Aggarwal, N. Chawla. When will it happen – Relationship Prediction in Heterogeneous Information Networks. *WSDM*, 2012.

[21] J. Tang, T. Lou, and J. Kleinberg. Inferring social ties across heterogenous networks. *WSDM*, 2012.

[22] B. Taskar, M. F. Wong, P. Abbeel and D. Koller. Link prediction in relational data. *NIPS*, 2003.

[23] D. Wang, D. Pedreschi, C. Song, F. Giannotti, and A.-L. Barabasi. Human mobility, social ties, and link prediction. *KDD*, 2011.

[24] Y. Yang, N. Chawla, Y. Sun, and J. Han. Predicting Links in Multi-Relational and Heterogeneous Networks. *ICDM*, 2012.

[25] T. Yang, R. Jin, Y. Chi, and S. Zhu. Combining link and content for community detection: a discriminative approach. *KDD*, 2009.

[26] K. Yu, W. Chu, S. Yu, V. Tresp and Z. Xu. Stochastic relational models for discriminative link prediction. *NIPS*, 2006.

[27] J. Zhu, J. Hong and G. Hughes. Using Markov models for web site link prediction. *HyperText*, 2002.

[28] C. Ding, X. He and H. D. Simon. On the Equivalence of Nonnegative Matrix Factorization and Spectral Clustering. *SDM*, 2005.

[29] C. Lee, M. Pham, N. Kim, M. K. Jeong, D. K. J. Lin and W. Art. A Novel Link Prediction Approach for Scale-free Networks. *WWW*, 2014.

[30] Lee, Daniel D. and Seung, H. Sebastian. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401, pp. 788–791, 1999.

[31] Chao Liu, Hung-chih Yang, Jinliang Fan, Li-Wei He and Yi-Min Wang. Distributed Nonnegative Matrix Factorization for Web-Scale Dyadic Data Analysis on MapReduce. *WWW*, 2010.

[32] Ahmed, Nesreen K. and Neville, Jennifer and Kompella, Ramana. Network Sampling: From Static to Streaming Graphs. *Transactions on Knowledge Discovery from Data (TKDD)*, 8(2), pp. 7:1–7:56 2014.

[33] Jaewon Yang and Jure Leskovec. Overlapping Community Detection at Scale: A Nonnegative Matrix Factorization Approach. *WSDM*, 2013.

[34] L. Katz. A New Status Index Derived from Sociometric Analysis. *Psychometrika*, 18(1), pp. 39–43, 1953.

[35] Linyuan Lu and Tao Zhou. Link Prediction in Complex Networks: A Survey. *Physica A*, pp. 1150–1170, 2011.

[36] Mohammad Al Hasan and Mohammed J. Zaki. A survey of link prediction in social networks. *Social Network Data Analytics*, 2011.

[37] J. Leskovec, L. Backstrom, R. Kumar and A. Tomkins. Microscopic Evolution of Social Networks. *KDD*, 2008.

[38] Leo Breiman. Bagging Predictors. *Machine Learning*, 24(2), pp. 123–140, 1996.

[39] N. Barbieri, F. Bonchi and G. Manco. Who to Follow and Why: Link Prediction with Explanations. *KDD*, 2014.

[40] J. Tang, S. Chang, C. Aggarwal and H. Liu. Negative Link Prediction in Social Media. *WSDM*, 2015.

[41] R. West, A. Paranjape and J. Leskovec. Mining Missing Hyperlinks from Human Navigation Traces: A Case Study of Wikipedia. *WWW*, 2015.