# An Ensemble Approach to Link Prediction

Liang Duan, Shuai Ma*, Charu Aggarwal, Tiejun Ma, Jinpeng Huai

**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 pairwise possibilities for links in a meaningful way. Further, even though link prediction is closely related to missing value estimation problems, it is often difficult to use sophisticated models such as latent factor methods because of their computational complexity on large networks. Hence, most known link prediction methods are designed for *evaluating* the link propensity on a *specified* subset of links, rather than on the entire networks. In practice, however, it is essential to perform an exhaustive search over the entire networks. In this article, we propose an ensemble enabled approach to scaling up link prediction, by decomposing traditional link prediction problems into subproblems of smaller size. These subproblems are each solved with latent factor models, which can be effectively implemented on networks of modest size. By incorporating with the characteristics of link prediction, the ensemble approach further reduces the sizes of subproblems without sacrificing its prediction accuracy. The ensemble enabled approach has several advantages in terms of performance, and our experimental results demonstrate the effectiveness and scalability of our approach.

**Index Terms**—Link prediction, NMF, ensembles, social 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 [15], [17], [31] for surveys). The link prediction problem has numerous applications, such as the recommendation of friends in a social network [6], [8], [36], the recommendation of images in a multimedia network [2], or the recommendation of collaborators in a scientific network [19], [24], and, therefore, link prediction methods have been extensively studied because of their numerous applications in various network-centered domains.

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.,* [20], [38], [43], [46].

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 time in this table represent very optimistic lower bounds on the required time 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 time are impractical.

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.

| 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 |

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 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 [15], [25].

The link prediction algorithms are classified into unsupervised and supervised methods. Unsupervised methods [24] typically use neighborhood measures such as the Adamic-Adar [1] and the Jaccard coefficient [24]. Supervised methods [25] treat the link prediction problem 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

- *L. Duan, S. Ma (correspondence) and J. Huai are with SKLSDE lab, School of Computer Science and Engineering, Beihang University, China. E-mail: {duanliang, mashuai, huaijp}@buaa.edu.cn*
- *C. Aggarwal is with IBM Thomas J. Watson Research Center, USA. E-mail: charu@us.ibm.com*
- *T. Ma is with Centre for Risk Research, Department of Decision Analytics and Risk, University of Southampton, UK. E-mail: tiejun.ma@soton.ac.uk*

*Manuscript received XXX, 2016; revised XXX, 2017.*

than unsupervised methods [25]. 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 [2]. 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 [3], [26] 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 [26]. 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 [32] which attempt to use these models.

One of the reasons that latent factor models are rarely used for link prediction is simply due to their complexity. In collaborative filtering applications, the item dimension is of the order of a few hundred thousand, whereas even the smallest real-world networks 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 will be seen later in this article, 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 a top-$k$ setting.

**Contributions**. To this end, we explore an *ensemble 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. An optimizing method is also provided for speeding up the search process when a threshold is available. This provides an efficient approach for the top-$k$ problem in link prediction.

Our problem decomposition method is an ensemble approach enabled with three structural bagging methods with performance guarantees, which has obvious *robustness* advantages. Note that the bagging methods combine the outputs of several predictors may reduce the overall risk of making a particularly poor prediction [9]. By incorporating with the characteristics of link prediction, the bagging methods maintain high prediction accuracy while reducing the network size via graph sampling techniques.

Using real-life datasets, we finally conduct an extensive experimental study, and show that our ensemble approach for link prediction is both effective and efficient. Indeed, (1) on Friendster with 15 million nodes and 1 billion edges, our approach could finish in two hours, while existing methods direct NMF, AA [1], RA [44] and BIGCLAM [40] could not finish in a day, and (2) our approach in general improves the accuracy on (Digg, YouTube, Wikipedia) compared with direct NMF, AA, RA and BIGCLAM.

**Organization**. This article 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 Link Prediction

As pointed out in Section 1, it is typically time consuming and memory intensive for latent factor models to search the top-$k$ possible links from the $O(n^2)$ search space. In this section, we first provide an efficient latent factor model for link prediction that generates non-negative and sparse factorizations, and then we design an efficient method to search the top-$k$ possible links.

We assume that $G(N, A)$ is an undirected network (or graph) 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 top-$k$ 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 $i$th 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:

$$w_{ij} \approx \overline{f_i} \cdot \overline{f_j}, \quad \forall i, j \in \{1, \ldots, n\}. \tag{1}$$

This condition can be directly written in the matrix form. Let $F$ be an $n \times r$ matrix, in which the $i$th row is the row vector $\overline{f_i}$. Then, the above 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 (NMF) 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 [28], where $F$ can be determined by starting with random nonnegative entries in $(0, 1)$, and using the following multiplicative update rule:

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

in which $\beta$ is a constant in $(0, 1]$ [12].

**Discussions of computational complexity**. 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 [21]. 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 $\epsilon$-approximate top-$k$ predictions, denoted as top-$(\epsilon, k)$ predictions.

**Definition 2** (top-$(\epsilon, k)$ predictions)**.** *A set $L$ of predicted links in a network $G(N, A)$ is a top-$(\epsilon, k)$ 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 $\epsilon$ less than the $k$-th best value of $\overline{f_h} \cdot \overline{f_l}$ for any link $(h, l)$ not included in $A$.*

Intuitively, this definition allows a qualitative tolerance of $\epsilon$ 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 $S$, 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 $S$:

```
1.  for each i = 1 to f_p do
2.      for each j = i + 1 to f'_p do
3.          if S_ip · S_jp < ε/r then
4.              break inner loop;
5.          else increase the score of node-pair (R_ip, R_jp) by
                an amount of S_ip · S_jp
6.      endfor
7.  endfor
```

This nested loop is designed to track the relevant subset of node pairs from which one can determine the top-$(\epsilon, k)$ 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 $\epsilon$, 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 $\epsilon$ therefore provides the user a way to set the tradeoff 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-$(\epsilon, k)$ 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.

We next illustrate the above nested searching with an example.

**Example 1:** Given a $5 \times 3$ matrix $F$, we sort the three columns of $F$ in a descending order to generate the matrix $S$ and store the corresponding inverted indices in the matrix $R$, shown as follows:

$$\underbrace{\begin{bmatrix}0.7 & 0.3 & 0.7\\ 0.5 & 0.7 & 0.9\\ 0.4 & 1.1 & 0.7\\ 0.0 & 0.8 & 0.0\\ 0.5 & 0.0 & 0.1\end{bmatrix}}_{F} \Longrightarrow \underbrace{\begin{bmatrix}0.7 & 1.1 & 0.9\\ 0.5 & 0.8 & 0.7\\ 0.5 & 0.7 & 0.7\\ 0.4 & 0.3 & 0.1\\ 0.0 & 0.0 & 0.0\end{bmatrix}}_{S} \underbrace{\begin{bmatrix}1 & 3 & 2\\ 2 & 4 & 1\\ 5 & 2 & 3\\ 3 & 1 & 5\\ 4 & 5 & 4\end{bmatrix}}_{R}$$

We assume *w.l.o.g.* that $\epsilon = 1$, then $\sqrt{\epsilon/r} = 0.58$ and $\epsilon/r = 0.33$. For $p = 1$ (resp. $p = 2$ and $p = 3$), we have ($f_1 = 1, f'_1 = 4$) (resp. ($f_2 = 3, f'_2 = 4$) and ($f_3 = 3, f'_3 = 4$)), and only need to multiply 9 times: ($S_{11} \cdot S_{21}$, $S_{11} \cdot S_{31}$) (resp. ($S_{12} \cdot S_{22}$, $S_{12} \cdot S_{32}$, $S_{12} \cdot S_{42}$, $S_{22} \cdot S_{32}$) and ($S_{13} \cdot S_{23}$, $S_{13} \cdot S_{33}$, $S_{23} \cdot S_{33}$)), together with 6 multiplications for inner loop checking, instead of 75 times for calculating $FF^T$ directly, which reduces the search space. We then save the above values into the corresponding node pairs, *e.g.,* increasing the score of node pair $(R_{11}, R_{21})$ by $S_{11} \cdot S_{21}$, and finally return all node pairs encountered. □

It remains to show that this procedure does find a valid set of top-$(\epsilon, k)$ 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-$(\epsilon, k)$ predictions.*

*Proof:* The main part of the proof is to show that any dot product is underestimated by at most $\epsilon$. The aforementioned pseudo-code 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 $\epsilon/r$. 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 $\sqrt{\epsilon/r}$ as the matrix $S$ is sorted and $S_{ip} \geq S_{jp}$. 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 termination condition here is that the product is at most $\epsilon/r$.

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

### 2.2 Speeding up Top-$(\epsilon, k)$ Predictions

In real-life applications, a common scenario is to find a set of top-$(\epsilon, k)$ predictions with such that the prediction values are required to be greater than a given threshold $\theta$ [7], [37]. In this case, the top-$(\epsilon, k)$ prediction can be further speeded up by pruning the node pairs whose prediction values are equal to or less than $\theta$ (see details on the setting of parameter $\theta$ in Section 3.6).

The simple yet effective strategy is that the prediction value of a node pair $(i, j)$ satisfies:

$$\overline{f_i} \cdot \overline{f_j} = \|\overline{f_i}\| \cdot \|\overline{f_j}\| \cdot \cos(\overline{f_i}, \overline{f_j}) \leq \|\overline{f_i}\| \cdot \|\overline{f_j}\|, \quad (4)$$

in which $\|\cdot\|$ is the length of a vector.

By Equation (4), we obtain the following:

$$\|\overline{f_i}\| \cdot \|\overline{f_j}\| \leq \theta \Longrightarrow \overline{f_i} \cdot \overline{f_j} \leq \theta \quad (5)$$

The prediction value is no larger than $\theta$ if the product of the vector lengths is no larger than $\theta$. As we are to find the node pairs whose prediction values are greater than $\theta$, we can ignore the node pairs whose length products are no larger than $\theta$ in the search process.

After the matrixes $S$ and $R$ are obtained, an additional step is required to calculate the length of each row vector $\overline{f_i}$ and store them in an array $M$. Then the following if-clause is inserted into line 3 in the above nested loop to prune the node pairs whose prediction values are equal to or less than $\theta$.

**if** $M[R_{ip}] \cdot M[R_{jp}] \leq \theta$ **then** continue;

It is easy to verify that this procedure correctly finds a valid set of top-$(\epsilon, k)$ predictions whose values are greater than $\theta$.

**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 [21], [27]. For sparse networks, whose node degrees are 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 the network size. In order to handle this computational problem, we propose the method of *ensemble decomposition* that provides both efficiency and effectiveness advantages.

## 3 Structural Bagging Methods

Since the link prediction problem scales worse than linearly with the network size (the previous mentioned latent factor model based method in Section 2 requires $O(nr^2)$ time), 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 [9]. In the following, we introduce three different ways for the bagging decomposition. We consider a network $G(N, A)$.

### 3.1 Random Node Bagging

Random node bagging is the simplest form of structural bagging, and its basic idea is to iteratively apply the following three steps:

(1) Select a random set of nodes $N_r$ in the network $G$ 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 the matrix $W$.

(3) Apply the symmetric NMF method in Section 2 to the reduced matrix $W_s$, and use the pruning search process in Section 2.2 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 the matrix $W$.

(3) Apply the symmetric NMF method in Section 2 to the reduced matrix $W_s$, and use the pruning search process in Section 2.2 to determine the predictions of all pairs of nodes of $N_s$.

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 each node pair is considered with a high probability, many node pairs will not be considered at all. However, such node pairs are typically not 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 with the least number of selected 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 Incorporating Link Prediction Characteristics

Different from existing graph sampling methods [4], [23], we employ the characteristics of link prediction, as our bagging methods are designed in particular for link prediction. We develop a novel graph sampling method accompanied with two strategies: (1) node uptake, to choose those nodes with a high possibility of forming links and (2) edge filter, to eliminate edges without affecting the prediction accuracy.

**Node uptake**. 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 [22]. Inspired by this, we develop a node uptake strategy 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 three 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.

**Edge filter**. Inspired by graph sparsification [10], [33] which has been successfully applied to clustering without sacrificing the quality, we develop an edge filter strategy to choose a portion of edges only, rather than all the edges of the subgraph induced on $N_s$ of $G$, which significantly improves the efficiency of the bagging methods. The challenge here is how to remove edges as many as possible while maintaining the high prediction accuracy. To do this, we make use of the following link prediction characteristic.

Preferential attachment (PA) [5], [22] is one of the well-known link prediction characteristics, which says that *the probability that a new link is connected to a node $i$ is proportional to its degree*. In other words, nodes with small degree tend to have fewer links in the future. Therefore, we remove those edges that have at least one endpoint with a smaller degree. Let $d_i$ and $d_j$ be the degrees of the endpoints $i$ and $j$ of edge $(i, j)$, respectively. Given a sampling ratio $\rho$, we sort edges in the induced subgraph by $\min(d_i, d_j)$ in a descending order and only select the top $m\rho$ edges, where $m$ is the total number of edges. To achieve this, we further revise the previous three bagging methods as follows:

> At the second step of the three bagging methods, construct a reduced adjacency matrix $W_s$ from the reduced subgraph obtained by applying the edge filter on the subgraph induced on the node set $N_s$ of the network $G$.

### 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 $\mu$ 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 $\mu/f^2$ ensemble components is at least $\mu$.*

*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 while 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 performs better than NMF, 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.

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

To ensure that a node pair appears in the ensemble components at least $\mu$ expected times, $\mu/f^2$ 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 on the matrix $W_s$ obtained by filtering edges on the reduced matrix, and the aforementioned pruning search process in Section 2.2 is used to determine the predictions of all pairs of nodes. If a node pair appears in multiple ensemble components and has multiple prediction values, the maximum prediction value is considered, which means that we can use the minimum value in the previous predicted links to speed up the top-$k$ predictions, *i.e.,* the minimum value is assigned to the parameter $\theta$ in Section 2.2. 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 $\mu/f^2$ ensemble components are returned.

**Remarks**. It is worth mentioning that our ensemble-enabled approach is a general framework, not limited to NMF, and may be applied to other link prediction methods, *e.g.,* AA [1] and RA [44].

## 4 Experimental Study

In this section, we present an extensive experimental study of our ensemble-enabled approach for link prediction. Using real-life datasets, we conducted four sets of experiments to evaluate: (1) the effectiveness and efficiency of our bagging methods with the top-$(\epsilon, k)$ speeding up technique, node uptake and edge filter techniques (referred to as bagging+ methods) vs. their counterparts bagging methods with only node uptake technique developed in [14], (2) the effectiveness and efficiency of our approach vs. conventional methods AA [1], RA [44] and BIGCLAM [40], (3) the impacts of various parameters of our approach, and (4) the limitations of our approach.

### 4.1 Experimental settings

We first present our experimental settings.

**Real-life datasets**. We used the real-life network datasets from the Koblenz Network Collection (http://konect.uni-koblenz.de/).

(1) Digg is a 5 year friendship graph of Digg users with $279,630$ nodes and $1,731,653$ directed edges.
(2) YouTube is a 7 month friendship graph of YouTube users with $3,223,589$ nodes and $9,375,374$ undirected edges.
(3) Wikipedia is a 6 year English Wikipedia hyperlink graph with $1,870,709$ nodes and $39,953,145$ directed edges.
(4) Flickr is a 6 month friendship connections of Flickr users with $2,302,925$ nodes and $33,140,017$ directed edges.
(5) Twitter is the follower network from Twitter with $41,652,230$ nodes and $1,468,365,182$ directed edges.

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.

| Datasets | Date | Nodes | Edges |
|---|---|---|---|
| Digg | 2005-08-06 — 2009-02-08 | 207,570 | 1,049,611 |
| | 2009-02-09 — 2009-07-08 | 207,570 | 467,816 |
| 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 | 5,856,596 |

Table 3
Parameters used in the experiments. Note that we set $k = 10^5$ for Digg and YouTube and $k = 10^6$ for other datasets, respectively.

| Parameters | Descriptions | Default |
|---|---|---|
| $\beta$ | Coefficient in NMF update rule | 0.5 |
| $iter$ | Number of iterations for NMF | 50 |
| $r$ | Number of latent factors | 10 / 30 |
| $\epsilon$ | Tolerance of top-($\epsilon$, $k$) prediction | 1 |
| $k$ | Number of links returned by top-($\epsilon$, $k$) prediction | $10^5$ / $10^6$ |
| $\mu$ | 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 |
| $\rho$ | Fraction of the number of edges to be selected for an ensemble component | 0.75 |
| $n$ | Number of nodes | see Table 2 |

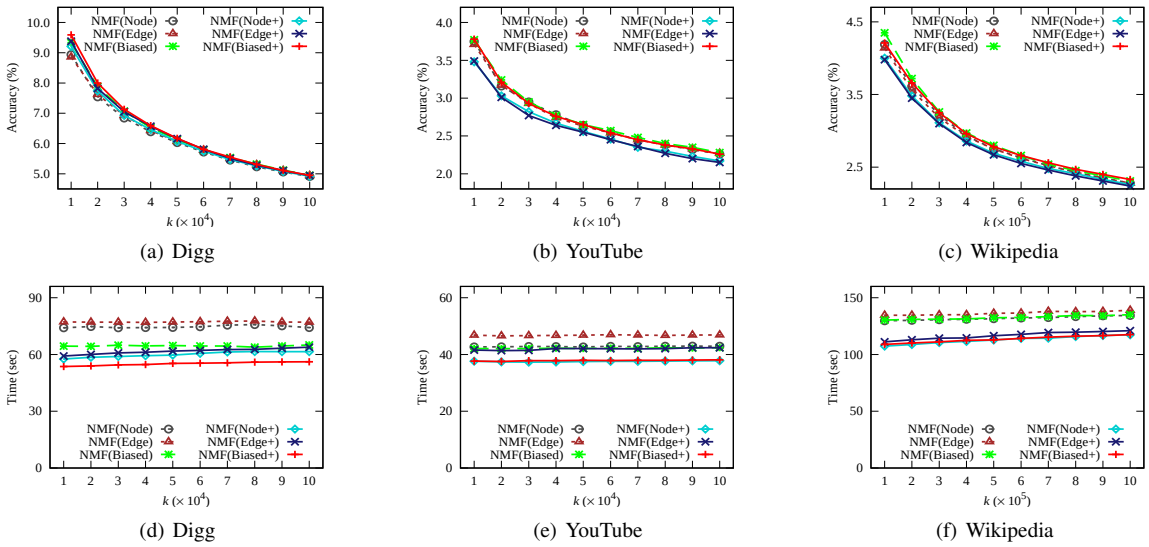
(6) Friendster is the friendship graph of the Friendster with $68,349,466$ nodes and $2,586,147,869$ directed edges.

Here (1) Digg, YouTube, Wikipedia and Flickr 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]: Algorithm AA is a popular neighborhood based method that produces a score for each link $(u, v)$

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

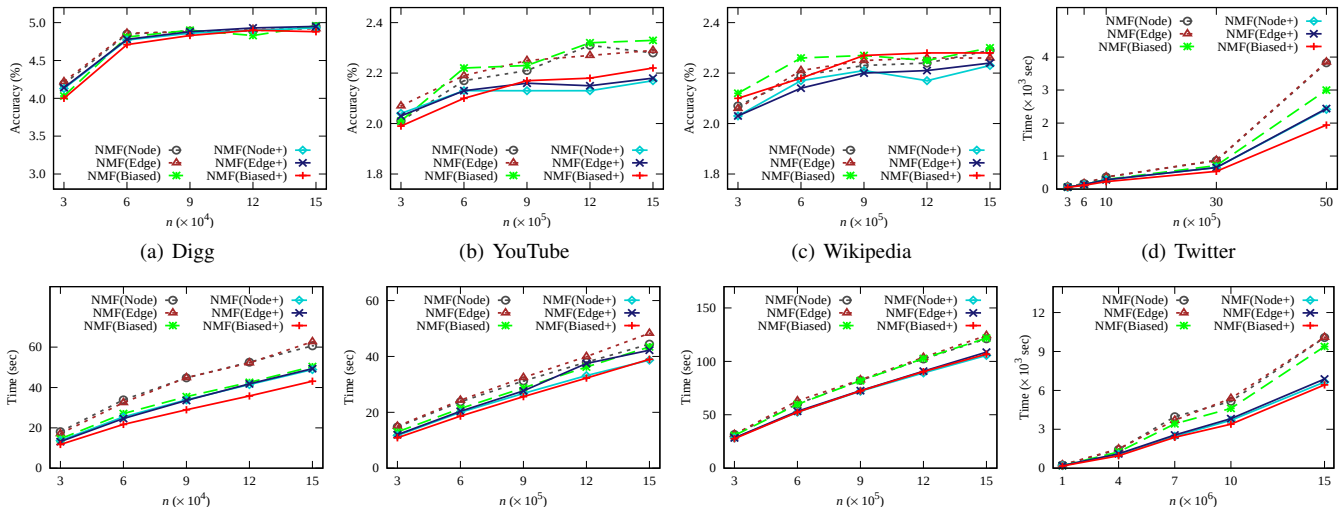
where $N(u)$ is the set of neighbors of node $u$. Lü *et al.* [31] showed that AA performs well on a range of networks because it only concerns 2-hop neighbors. We implemented a top-$k$ version of AA by searching the $k$ largest AA scoring links. The complexity of this method is $O(nd^2 \log(k))$, where $d$ is the average degree of networks. Indeed, there is also another popular link prediction method Katz based on the ensemble of all paths [18]. However, its complexity is $O(n^3)$, and does not work on large networks with millions of nodes. Therefore, we did not choose Katz for comparison in the experiments.




(a) Digg (b) YouTube (c) Wikipedia

(d) Digg (e) YouTube (f) Wikipedia

Figure 1. Bagging+ vs. Bagging on accuracy and efficiency: with respect to the number $k$ of predicted links.




(a) Digg (b) YouTube (c) Wikipedia (d) Twitter

(e) Digg (f) YouTube (g) Wikipedia (h) Friendster

Figure 2. Bagging+ vs. Bagging on accuracy and efficiency: with respect to the network sizes.

(2) Resource Allocation (RA) [44]: Algorithm RA is another neighborhood based method that assumes the node $u$ can send some resource to $v$ with their common neighbors playing the role of transmitters. The RA score for a link $(u, v)$ is

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

We also implemented a top-$k$ version of RA by searching its $k$ largest scoring links, which has the same time complexity as AA. Note that RA is very similar to AA when $|N(z)|$ is small, but greatly different when $|N(z)|$ is large. Zhou *et al.* [44] show that RA performs better than AA on the networks with the high degree heterogeneity since RA punishes high degree common neighbors more. Hence we chose it for comparison in the experiments.

(3) Cluster Affiliation Model for Big Networks (BIGCLAM) [40]: 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 the 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 $c$. Given $F$, the BIGCLAM generates a network $G(N, A)$ by creating edge $(u, v)$ between a pair of nodes $u, v \in N$ with the probability

$$p(u, v) = 1 - \exp(-\overline{f_u} \cdot \overline{f_v}),$$

where $\overline{f_u}$ 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-$(\epsilon, k)$ method to predict links.

**Implementation**. We implemented all algorithms including (a) AA, RA, BIGCLAM, link prediction method in Section 2 (NMF), (b) NMF with random node bagging (NMF(Node)), NMF with edge bagging (NMF(Edge)), NMF with biased edge bagging (NMF(Biased)), developed in [14], and (c) their counterparts NMF(Node+), NMF(Edge+) and NMF(Biased+) with the top-

$(\epsilon, k)$ speeding up technique in Section 2.2 and the edge filter technique in Section 3.4, 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 $(50, 50, 50)$ (resp. $(10, 40, 50)$) for NMF (resp. BIGCLAM) on Digg, YouTube, and Wikipedia, respectively, by default. For the bagging+ and bagging methods, we also fixed $r = 30$ on Digg and $r = 10$ on other datasets by default (See Exp-3.4 for more details about the setting of $r$). Here Flickr is used to illustrate the limitation of our bagging methods only. The other parameters with their descriptions and default values are presented in Table 3.

### 4.2.1 Bagging+ vs. Bagging

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

$$accuracy(x) = \frac{\#\text{ of correctly predicted links}}{\text{the number } k \text{ of predicted links}}. \quad (6)$$

We first evaluate the improvements of our bagging+ methods and then evaluate the impacts of $k$ and network sizes. Compared with their counterparts bagging methods [14], the bagging+ methods have two modifications: the top-$(\epsilon, k)$ speeding up technique (denoted as top-$(\epsilon, k)$+) and the edge filter technique. These techniques have no side effects on accuracy. Therefore, we focus on the efficiency improvements of these techniques.

**Exp-1.1: Performance improvements of bagging+ methods**. To evaluate the efficiency improvements of these techniques using in our bagging+ methods, we generated different combinations of these techniques and fixed the parameters to their default values. The results of running time are reported in Table 4.

The results tell us that (a) Top-$(\epsilon, k)$+ indeed improves the efficiency of Top-$k$ on all datasets, (b) the edge filter technique also speeds up the bagging methods, and (c) the maximum speedup is obtained by taking the two techniques together. Therefore, we adopt these two techniques in our bagging+ methods to achieve the maximum speedup. Note that the running time of NMF(Biased+) and NMF(Node+) on YouTube and Wikipedia is very close because NMF(Biased+) spends more time on sampling the nodes from the input graph than NMF(Node+) while NMF(Node+) spends more time on processing its ensembles. Furthermore, the improvement of Top-$(\epsilon, k)$+ on Wikipedia is not distinct since the $\epsilon = 1$ is very suitable for this dataset.

**Exp-1.2: Impacts of** $k$. To evaluate the impacts of the number $k$ of predicted links, we varied $k$ from $1 \times 10^4$ to $1 \times 10^5$ on Digg and YouTube (resp. from $1 \times 10^5$ to $1 \times 10^6$ on Wikipedia) and fixed other parameters to their default values. The results of accuracy and running time are reported in Figures 1(a)–1(c) and Figures 1(d)–1(f), respectively.

The accuracy results tell us that (a) NMF(Biased) and NMF(Biased+) are the best methods on all datasets, (b) the accuracy of the bagging+ methods is very close to that of their counterparts bagging methods, and (c) the accuracy of all methods decreases with the increase of $k$. It is means that the bagging+ methods maintain the accuracy while some of the edges have been removed compared with their counterparts bagging methods. This verifies the effectiveness of the bagging+ methods.

Table 4
Running time (sec.) of different combinations of the modifications. Node+ (resp. Edge+ and Biased+) is the Random Node Bagging (resp. Edge Bagging and Biased Edge Bagging) with edge filter.

| Bagging | Top-$k$ Prediction | Digg | YouTube | Wikipedia |
|---|---|---|---|---|
| Node | Top-$(\epsilon, k)$ | 74.15 | 43.43 | 134.06 |
| Node | Top-$(\epsilon, k)$+ | 65.01 | 41.29 | 133.80 |
| Node+ | Top-$(\epsilon, k)$ | 67.22 | 40.02 | 118.13 |
| Node+ | Top-$(\epsilon, k)$+ | **60.38** | **37.87** | **117.27** |
| Edge | Top-$(\epsilon, k)$ | 77.33 | 47.23 | 137.72 |
| Edge | Top-$(\epsilon, k)$+ | 67.26 | 45.04 | 137.57 |
| Edge+ | Top-$(\epsilon, k)$ | 70.29 | 43.69 | 121.42 |
| Edge+ | Top-$(\epsilon, k)$+ | **62.31** | **41.29** | **121.16** |
| Biased | Top-$(\epsilon, k)$ | 63.93 | 42.39 | 135.59 |
| Biased | Top-$(\epsilon, k)$+ | 58.56 | 41.07 | 134.02 |
| Biased+ | Top-$(\epsilon, k)$ | 58.97 | 39.61 | 119.44 |
| Biased+ | Top-$(\epsilon, k)$+ | **55.32** | **38.15** | **117.25** |

The running time results tell us that (a) the NMF(Biased+) outperforms other methods on all datasets, (b) the three bagging+ methods are faster than their counterparts bagging methods, and (c) the running time of all methods increase slightly with the increase of $k$. For instance, NMF(Biased+) is $(1.2, 1.1, 1.2)$ times faster than NMF(Biased) on Digg, YouTube and Wikipedia, respectively. This verifies the efficiency of the bagging+ methods.

**Exp-1.3: Impacts of network sizes**. To evaluate the impacts of network sizes, we varied the number of nodes $n$ from $3 \times 10^4$ to $1.5 \times 10^5$ on Digg (resp. from $3 \times 10^5$ to $1.5 \times 10^6$ on YouTube and Wikipedia, from $3 \times 10^5$ to $5 \times 10^6$ on Twitter and from $1 \times 10^6$ to $1.5 \times 10^7$ 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 $r = 50$ for NMF (resp. $r = 33$ for BIGCLAM and $r = 17$ for the bagging methods). The results of accuracy and running time are reported in Figures 2(a)–2(c) and Figures 2(d)–2(h), respectively.

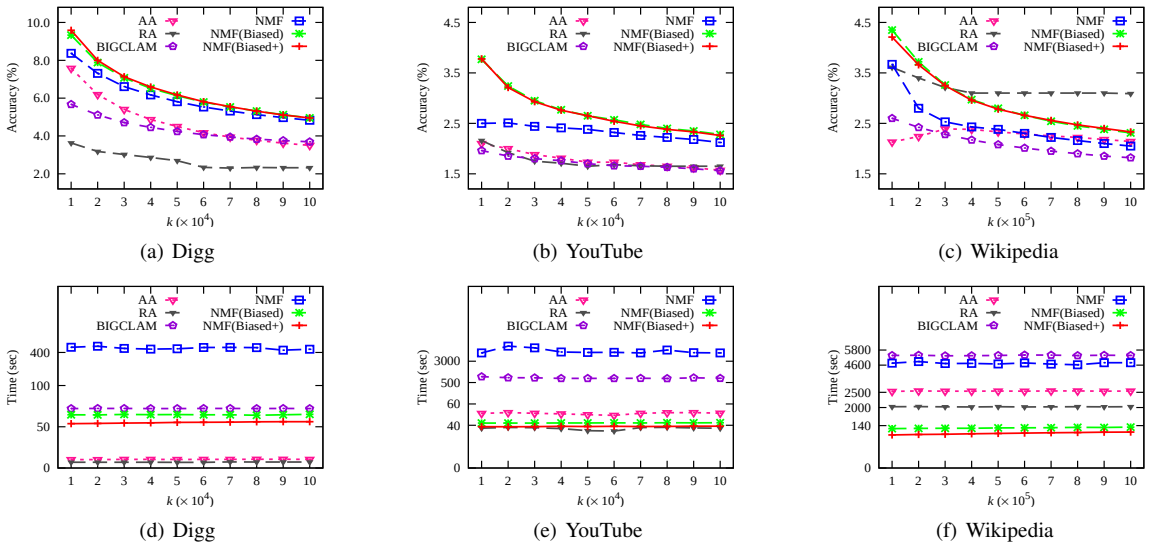
The accuracy results tell us that (a) NMF(Biased) has the highest accuracy on all datasets among the three bagging methods, (b) the three bagging+ methods perform as well as their counterparts bagging methods. This means that the bagging+ methods are also accurate and robust with the increase of network sizes.

The running time results tell us that (a) NMF(Biased+) is the fastest method on all datasets, (b) the bagging+ methods are faster than their counterparts bagging methods, and (c) the running time of all methods increase nearly linearly with the increase of $n$. For instance, NMF(Biased+) speeds up NMF(Biased) for around 1.3 (resp. 1.4) times on Twitter (resp. Friendster) and is thus essential for making our bagging+ methods scale better than their counterparts to large networks.

### 4.2.2 Comparison with AA, RA and BIGCLAM

In the second set of tests, we evaluated the effectiveness and efficiency of our methods compared with AA, RA and BIGCLAM. From the previous tests, we find that the NMF(Biased+) and its counterpart NMF(Biased) are the best bagging methods. Therefore, we chose them for the comparison in this set of tests.

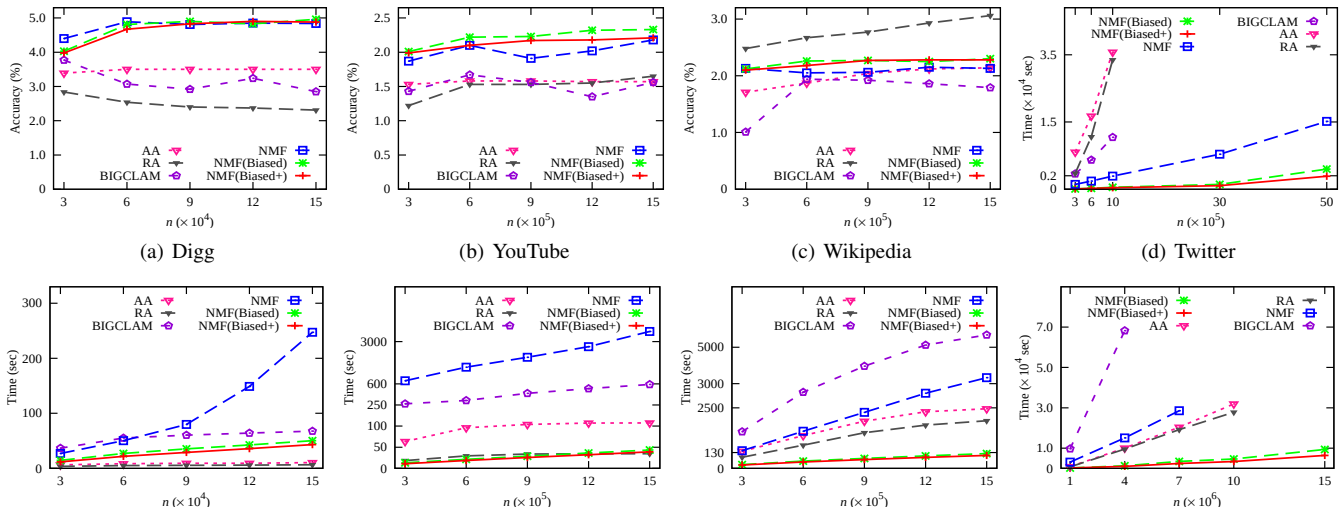
**Exp-2.1: Impacts of** $k$. Using the same setting as Exp-1.2, we evaluated the impacts of the number $k$ of predicted links. The results of accuracy and running time are reported in Figures 3(a)–3(c) and Figures 3(d)–3(f), respectively.




(a) Digg (b) YouTube (c) Wikipedia

(d) Digg (e) YouTube (f) Wikipedia

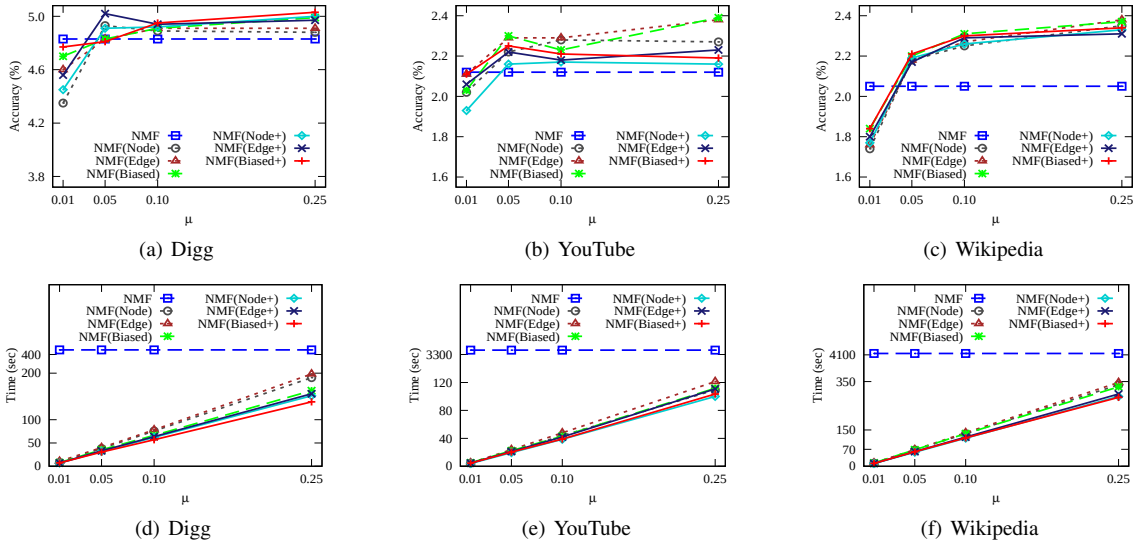
Figure 3. Accuracy and efficiency comparison with AA, RA and BIGCLAM: with respect to the number $k$ of predicted links.




(a) Digg (b) YouTube (c) Wikipedia (d) Twitter

(e) Digg (f) YouTube (g) Wikipedia (h) Friendster

Figure 4. Accuracy and efficiency comparison with AA, RA and BIGCLAM: with respect to the network sizes.




(a) Digg (b) YouTube (c) Wikipedia

(d) Digg (e) YouTube (f) Wikipedia

Figure 5. Accuracy and efficiency comparison: with respect to the expected appearing times $\mu$.

The accuracy results tell us that (a) NMF(Biased) outperforms other methods on all datasets except Wikipedia, (b) the accuracy of NMF(Biased+) is very close to that of NMF(Biased), (c) both NMF(Biased) and NMF(Biased+) have a higher accuracy than AA, RA and BIGCLAM, except RA on Wikipedia, (d) NMF is more accurate than AA, RA and BIGCLAM, except RA on Wikipedia, and (e) the accuracy of all methods decreases with the increase of $k$. Indeed, when $k$ is the default value, NMF(Biased) improves the accuracy by $(2.5\%, 41.4\%, 113.4\%, 34.1\%)$, $(7.5\%, 45.2\%, 38.2\%, 46.2\%)$ and $(12.7\%, 7.9\%, -25.2\%, 26.9\%)$ over (NMF, AA, RA, BIGCLAM) on Digg, YouTube and Wikipedia, respectively. Here, minus $(-)$ means a worse accuracy. Moreover, NMF(Biased) performs consistently well on all networks (*i.e.,* more robust), unlike RA which works well on Wikipedia but poorly on other datasets. This verifies the effectiveness of our bagging methods.

The running time results tell us that (a) NMF(Biased+) is the fastest method on all datasets except that AA and RA are the two fastest methods on Digg since its size is really small, (b) the two bagging methods are much faster than NMF, AA, RA and BIGCLAM, (c) the running time of all methods is insensitive to the increase of $k$, except AA and RA whose time complexities are both $O(nd^2 \log(k))$. Indeed, the bagging+ and bagging methods finished the prediction in 140 seconds on the three datasets. Furthermore, NMF(Biased+) is $(7.3, 0.2, 0.1, 1.2)$, $(90, 1.3, 1.0, 15.5)$ and $(37, 22, 17, 49)$ times faster than (NMF, AA, RA, BIGCLAM) on Digg, YouTube and Wikipedia, respectively. This verifies the efficiency of our bagging methods.

Note that the accuracy of all methods is not very high, even the best accuracy (NMF(Biased+) on Digg when $k = 1 \times 10^4$) is less than $10\%$. The reason is that there are more than $1 \times 10^{10}$ possible links in the search space of each dataset, but less than $1 \times 10^7$ links in the ground truth. Since RA is suitable for high degree heterogeneity networks [44], it performs the best on Wikipedia which contains a number of extreme high degree nodes. In addition, NMF is slower than AA and RA on three datasets because $r$ is fixed to 50, which is consistent to the $O(nr^2)$ complexity of NMF. The running time of AA (resp. RA) is about 50 (resp. 37) seconds on YouTube, while more than 2,500 (resp. 1,900) seconds on Wikipedia. This is because that the average degrees of these datasets are 5 and 33, and AA and RA take more time with the increase of the network degree. 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, but takes more time on Wikipedia when the degree is increased.

**Exp-2.2: Impacts of network sizes**. Using the same setting as Exp-1.3, we evaluated the impacts of network sizes. The results of accuracy and running time are reported in Figures 4(a)–4(c) and Figures 4(d)–4(h), respectively. Note that there are *some missing plots* for NMF, AA, RA and BIGCLAM in the Figures 4(d) and 4(h) as they could not finished in 24 hours.

The accuracy results tell us that (a) NMF(Biased) has the highest accuracy on all datasets except Wikipedia, (b) NMF(Biased+) performs as well as its counterpart NMF(Biased), (c) the bagging+ and bagging methods perform better than NMF, AA, RA and BIGCLAM, except RA on Wikipedia, and (d) NMF has a higher accuracy than AA, RA and BIGCLAM, except RA on Wikipedia. That is, our methods are more accurate and robust with the increase of network sizes.

The running time results tell us that (a) our bagging+ and bagging methods are much faster than the other methods, (b) the running time of all methods increase with the increase of $n$. For instance, NMF(Biased+) speeds up (NMF, AA, RA, BIGCLAM) for around $(12, 135, 66, 53)$ (resp. $(15, 8, 8, 65)$) 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 Digg and YouTube, but faster on other datasets because BIGCLAM needs more time with the increase of the network degree, which is consistent with the complexity analysis.

#### 4.2.3 Impacts of Parameters

In the third set of tests, we evaluated the impacts of parameters on the accuracy and running time of bagging+, bagging, NMF and BIGCLAM. We first tested the impacts of $\mu$ and $f$ in bagging+ and bagging methods. We then tested the impacts of $\rho$ in bagging+ methods. We finally tested the impacts of $r$ and $\epsilon$.

**Exp-3.1: Impacts of** $\mu$. To evaluate the impacts of $\mu$, we varied $\mu$ from $0.01$ to $0.25$ and fixed other parameters to their default values. The accuracy and running time results are reported in Figures 5(a)–5(c) and Figures 5(d)–5(f), respectively. We also plotted the accuracy and running time of NMF for comparison.

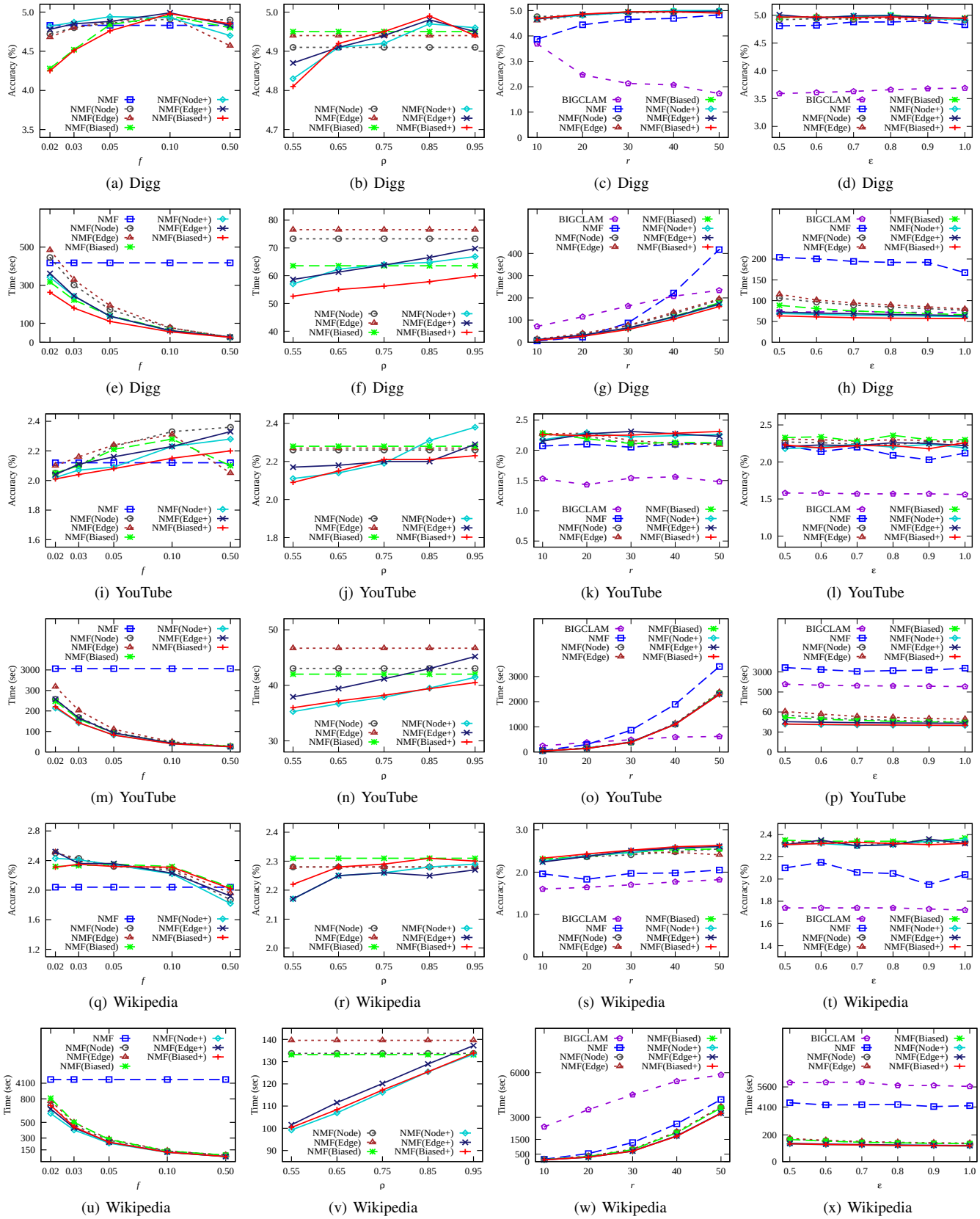
The results tell us that (a) bagging+ and bagging methods are more accurate than NMF when $\mu$ is large enough, (b) the accuracy of bagging+ and bagging methods increases with the increase of $\mu$ and becomes stable when $\mu$ is greater than $0.1$. This means that the accuracy of bagging+ and bagging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

**Exp-3.2: 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 Figures 6(a), 6(i) and 6(q) and Figures 6(e), 6(m) and 6(u), respectively.

The results tell us that (a) bagging+ and bagging methods have a higher accuracy than NMF when $f = 0.1$, (b) the accuracy of bagging methods decreases with the increase of $f$ when $f$ is greater than 0.1 on Digg and Wikipedia, and (c) the running time of bagging methods increases nearly linearly with the decrease of $f$. Note that the complexity of bagging methods is $O((n_1 r^2 + n_1 d_1 r) * \mu/f^2)$, 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 $O(nd_1 r\mu/f)$. Thus, the running time of bagging methods increases with the decrease of $f$. We fixed $f = 0.1$ to keep the accuracy of bagging+ and bagging methods better than that of NMF, by default to achieve a better efficiency.

**Exp-3.3: Impacts of** $\rho$. To evaluate the impacts of $\rho$ for the bagging+ methods, we varied $\rho$ from $0.55$ to $0.95$ and fixed other parameters to their default values. The accuracy and running time results are reported in Figures 6(b), 6(j) and 6(r) and Figures 6(f), 6(n) and 6(v), respectively. We also plotted the accuracy and running time of their counterparts bagging methods for comparison.

The results tell us that (a) the three bagging+ methods perform as well as their counterparts bagging methods in accuracy when $\rho$ is greater than 0.75, (b) the accuracy of the bagging+ methods increases with the increase of $\rho$, (c) the bagging+ methods are




(a) Digg (b) Digg (c) Digg (d) Digg

(e) Digg (f) Digg (g) Digg (h) Digg

(i) YouTube (j) YouTube (k) YouTube (l) YouTube

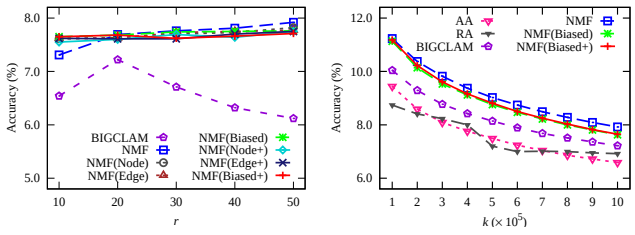
(m) YouTube (n) YouTube (o) YouTube (p) YouTube

(q) Wikipedia (r) Wikipedia (s) Wikipedia (t) Wikipedia

(u) Wikipedia (v) Wikipedia (w) Wikipedia (x) Wikipedia

Figure 6. Accuracy and efficiency comparison: with respect to the fraction $f$, the fraction $\rho$, the number $r$ of latent factors and the tolerant $\epsilon$ of top-$(\epsilon, k)$ prediction.

faster than their counterparts bagging methods when $\rho$ is less than 0.95, and (d) the running time of the bagging+ methods increases linearly with the increase of $\rho$, which is consistent with the complexity that is proportional to the number of edges in the ensemble component. Keeping the accuracy of the bagging+ methods close to that of the bagging methods, we fixed $\rho = 0.75$




(a) (b)

Figure 7. Accuracy comparison on Flickr: (a) with respect to the number $r$ of latent factors, (b) with respect to the number $k$ of predicted links.

by default to achieve a better efficiency.

**Exp-3.4: 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 Figures 6(c), 6(k) and 6(s) and Figures 6(g), 6(o) and 6(w), respectively.

The results tell us that (a) NMF(Biased+) and NMF(Biased) obtain the best accuracy compared with other methods, (b) bagging+ and bagging methods have a higher accuracy than 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+ and bagging methods. To obtain the highest accuracy, we fixed $r$ to $(50, 50, 50)$ (resp. $(10, 40, 50)$) for NMF (resp. BIGCLAM) on Digg, YouTube and Wikipedia, respectively. Note that, when $r = 30$ on Digg and $r = 10$ on other datasets, the accuracy of bagging+ and bagging methods is greater than the best accuracy of NMF. Hence, we fixed $r = 30$ on Digg and $r = 10$ on other datasets by default for bagging+ and bagging methods.

**Exp-3.5: Impacts of** $\epsilon$. To evaluate the impacts of $\epsilon$, we varied $\epsilon$ from 0.5 to 1.0 and fixed other parameters to their default values. The accuracy and running time results are reported in Figures 6(d), 6(l) and 6(t) and Figures 6(h), 6(p) and 6(x), respectively.

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

### 4.2.4 Analysis of Limitations

In the last set of tests, we discuss the limitations of our methods: (1) the bagging methods might not as good as NMF in certain situations, and (2) both the bagging and NMF methods cannot predict valid links with large hop distances as well as those with short hop distances. We first report the accuracy comparison between our bagging methods and NMF on Flickr. We then report the accuracy of the bagging methods and NMF with varied hop distances. These could be useful when users are to apply our bagging methods in practice.

(1) We first report the results on the impact of parameters $r$ and $k$ shown in Figures 7(a) and 7(b). We varied $r$ from 10 to 50 (resp. $k$ from $10^5$ to $10^6$) and fixed other parameters to their default values. The results tell us that the accuracy of our bagging methods is slightly lower than that of NMF. Indeed, when $k$ is the default value, NMF(Biased) decreases the accuracy by 4% on Flickr compared with NMF.

Table 5
Pairwise overlapping of predicted links and nodes of ensemble components.

| Overlap | Methods | Digg | YouTube | Wikipedia | Flickr |
|---|---|---|---|---|---|
| Links | Node | 0.57 | 0.54 | 0.39 | **0.69** |
| | Edge | 0.58 | 0.55 | 0.40 | **0.72** |
| | Biased | 0.46 | 0.48 | 0.42 | **0.67** |
| $N_s$ | Node | 0.31 | 0.22 | 0.29 | **0.33** |
| | Edge | 0.35 | 0.27 | 0.30 | **0.41** |
| | Biased | 0.21 | 0.11 | **0.29** | 0.21 |

We next explain the reasons. It is well known that the principle in ensemble methods is to make each ensemble as unique as possible and this is also called as diversity [45]. Diversity can be achieved by using different ensemble components or different settings of parameters for the ensemble base algorithms. Since the base algorithm and its parameters were fixed in our experiments, we examined the diversity of ensembles generated by our bagging methods. We adopted the pairwise overlapping of predicted links and nodes (*i.e.,* $N_s$ in each ensemble component) among ensemble components to measure the diversity. The value of pairwise overlapping is between zero and one and the higher the value means the lower the diversity. The results are shown in Table 5.

The results tell us that (a) the pairwise overlapping of predicted links on Flickr is obviously higher than that on other datasets, which means that the diversity on Flickr is the worst one, (b) the pairwise overlapping of nodes for NMF(Node) and NMF(Edge) methods on Flickr is also higher than that on other datasets, which may be one of reasons decreasing the diversity of these methods. These impair the performance of ensemble methods, which leads to a lower accuracy for our bagging methods on Flickr than NMF.

Moreover, observe that NMF(Edge) has the highest pairwise overlapping of predicted links and nodes since this method inclines to select high-degree nodes repeatedly. Moreover, NMF(Biased) overcomes the drawback of NMF(Edge) and obtains the lowest pairwise overlapping, which guarantees its higher accuracy. That is, to gain a high accuracy with the bagging methods, it is necessary to make each ensemble as unique as possible.

(2) We also study the distribution of hod distances induced by the links in the ground truth data, and then compared the accuracy of BIGCLAM, NMF and NMF(Biased) with varied hop distances, shown in Table 6. Although a large portion of ground truth links are with 2 or 3 hops, there remain many true links with hops greater than 3. However, BIGCLAM, NMF and NMF(Biased) are more accurate for predicting links with 2 or 3 hop, but are not good at predicting links with large hop distances. To improve the prediction accuracy, a good predictor should successfully predict links with high hop distances as well as with short hop distances.

To address these limitations of ensemble-enabled methods, it deserves a full treatment in the future.

**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-($\epsilon$, $k$) process to explore the $O(n^2)$ search space. NMF is more effective and robust than AA, RA and BIGCLAM. Moreover, NMF runs faster than BIGCLAM on dense networks, *e.g.,* Wikipedia, Twitter and Friendster, which is consistent with the complexity analysis that BIGCLAM is sensitive to the degree of networks.

(2) The running time of NMF, however, is increased quadratically with the increase of $r$. This might be a bottleneck for large networks. By decomposing the link prediction problem into smaller

Table 6
The distribution of hops between ground truth links and the accuracy comparison of BIGCLAM, NMF and NMF(Biased) in each hop distance.

| The Distribution of Hop Distances | | | | Accuracy of BIGCLAM | | |
|---|---|---|---|---|---|---|
| Hop | Digg | YouTube | Wikipedia | Digg | YouTube | Wikipedia |
| 2 | 32% | 26% | 63% | 3.8% | 1.7% | 1.9% |
| 3 | 39% | 37% | 32% | 0.4% | 0.1% | 0 |
| $\geq 4$ | 29% | 37% | 5% | 0 | 0.2% | 0 |
| Accuracy of NMF | | | | Accuracy of NMF(Biased) | | |
| Hop | Digg | YouTube | Wikipedia | Digg | YouTube | Wikipedia |
| 2 | 4.8% | 2.1% | 2.1% | 4.9% | 2.1% | 2.4% |
| 3 | 0.8% | 0.3% | 2.3% | 0 | 1.1% | 0.1% |
| $\geq 4$ | 0 | 0 | 0 | 0 | 0 | 0 |

pieces and solving them independently, our bagging methods are more efficient and scale well with the size and density of large networks, especially for the bagging+ methods with the top-$(\epsilon, k)$ speeding up and edge filter techniques, *e.g.,* NMF(Biased+) finished in two hours on Friendster with $1.5 \times 10^7$ nodes and $1.0 \times 10^9$ edges, while NMF, AA, RA and BIGCLAM could not finish in a day. Further, NMF(Biased+) speeds up (NMF(Biased), NMF, AA, RA, BIGCLAM) for around $(1.3, 12, 135, 66, 53)$ (resp. $(1.4, 15, 8, 8, 65)$) times on Twitter (resp. Friendster).

(3) Incorporated into link prediction characteristics, our bagging methods also provide the effectiveness advantages for link prediction, *e.g.,* NMF(Biased) improves the accuracy by $(2.5\%, 41.4\%, 113.4\%, 34.1\%)$, $(7.5\%, 45.2\%, 38.2\%, 46.2\%)$ and $(12.7\%, 7.9\%, -25.2\%, 26.9\%)$ over (NMF, AA, RA, BIGCLAM) on Digg, YouTube and Wikipedia, respectively. Here, minus $(-)$ means a worse accuracy.

## 5 Related work

This study extends our earlier work [14] by adding (a) an optimizing method for speeding up top-$k$ predictions *w.r.t.* a threshold (Section 2.2), (b) an edge filter technique to reduce the sizes of ensembles while keeping the high prediction accuracy (Section 3.4), and (c) a more detailed experimental study, including the limitation discussion of our approach (Section 4).

The link prediction problem has been studied extensively in the data mining and machine learning community [24], [31], and has various applications [2], [6], [8], [19], [24], [29], [36]. The link prediction algorithms can be classified into unsupervised and supervised methods [25]. 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 [18] 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 [31]; And (c) [19], [24] investigate the low rank approximation methods by generating a small rank matrix to approximate the initial adjacency matrix. Supervised methods [10], [25] typically treat link prediction as a classification problem, *e.g.,* supervised matrix factorization and random walk based approaches [6], [32]. Moreover, the link predictability has also been studied in [30].

Recently, several models for link prediction have been proposed, such as community affiliation models [40], stochastic topic models [8], negative link prediction models [36] and statistical relational models [16]. Moreover, link prediction has also been studied for mining missing hyperlinks [39]. While some recent work has focused on the heterogeneous [41], temporal [38] scenarios, dynamic networks [46], coupled networks [13], graph streams [43] and signed networks [34], some optimizing methods for link recommendation have been proposed to improve the precision at the top of the recommending list [35]. 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 [21], which has been successfully used for collaborative filtering [26]. Since the adjacency matrix in our approach is symmetric, we adopt the symmetric NMF method [12]. Methods for retrieving large entries in the product of two matrices have been studied in [7], [37], which motivate us to speed up the top-$(\epsilon, k)$ prediction. Our work is also related to bagging predictor [9] that generates an aggregated predictor based on multiple bootstrap samples. Different from the bootstrap sampling methods, we focus on sampling subgraphs from large networks. To our knowledge, although a variety of graph sampling techniques have been introduced in [4], [11], our approach is the first work that combines link prediction characteristics [22] with graph sampling methods to achieve the 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. We have also developed an optimization to speed up the top-$k$ predictions when a threshold is available. 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 have then developed three bagging methods that are designed in particular for link prediction, which also provide a better accuracy and scalability. Furthermore, we have proposed effective techniques to reduce the network sizes of ensembles in the bagging without sacrificing the prediction accuracy. Finally, we have experimentally verified that our ensemble-enabled approach is much more effective and scalable than existing methods, *e.g.,* direct NMF, AA [1], RA [44] and BIGCLAM [40].

Several topics need further investigation. First, we are to develop distributed approaches scalable on networks with billions of nodes, in a way similar to [27], [42]. Second, we are to develop new principles and methods to address the limitations observed in Section 4.2.4. Third, we are to study personalized recommendations using our ensemble-enabled link prediction approach.

## Acknowledgments

This work is supported in part by NSFC (U1636210), 973 program (2014CB340300), NSFC (61322207&61421003), Special Funds of Beijing Municipal Science & Technology Commission, Beijing Advanced Innovation Center for Big Data and Brain Computing, and MSRA Collaborative Research Program.

## References

[1] L. Adamic and E. Adar. Friends and neighbors on the web. *Social Networks*, 25:211–230, 2001.
[2] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *TKDE*, 17(6):734–749, 2005.
[3] C. Aggarwal and S. Parthasarathy. Mining massively incomplete data sets by conceptual reconstruction. In *KDD*, 2001.
[4] N. K. Ahmed, J. Neville, and R. Kompella. Network sampling: From static to streaming graphs. *TKDD*, 8(2):7:1–7:56, 2014.

[5] R. Albert and A. L. Barabasi. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.
[6] L. Backstrom and J. Leskovec. Supervised random walks: predicting and recommending links in social networks. In *WSDM*, 2011.
[7] G. Ballard, T. Kolda, A. Pinar, and C. Seshadhri. Diamond sampling for approximate maximum all-pairs dot-product (mad) search. In *ICDM*, 2015.
[8] N. Barbieri, F. Bonchi, and G. Manco. Who to follow and why: Link prediction with explanations. In *KDD*, 2014.
[9] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.
[10] Y. Chen, M. Chen, and P. S. Yu. Ensemble of diverse sparsifications for link prediction in large-scale networks. In *ICDM*, 2015.
[11] F. Chierichetti, A. Dasgupta, R. Kumar, S. Lattanzi, and T. Sarlos. On sampling nodes in a network. In *WWW*, 2016.
[12] C. Ding, X. He, and H. D. Simon. On the equivalence of nonnegative matrix factorization and spectral clustering. In *SDM*, 2005.
[13] Y. Dong, J. Zhang, J. Tang, N. V. Chawla, and B. Wang. Coupledlp: Link prediction in coupled networks. In *KDD*, 2015.
[14] L. Duan, C. Aggarwal, S. Ma, R. Hu, and J. Huai. Scaling up link prediction with ensembles. In *WSDM*, 2016.
[15] L. Getoor and C. Diehl. Link mining: A survey. *SIGKDD Exploration*, pages 3–12, 2005.
[16] L. Getoor, N. Friedman, D. Koller, and B. Taskar. Learning probabilistic models of link structure. *JMLR*, 3:679–707, 2002.
[17] M. A. Hasan and M. J. Zaki. A survey of link prediction in social networks. In *Social Network Data Analytics*, 2011.
[18] L. Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, 1953.
[19] J. Kunegis and A. Lommatzsch. Learning spectral graph transformations for link prediction. In *ICML*, 2009.
[20] 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. In *WWW*, 2014.
[21] D. D. Lee and H. S. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401:788–791, 1999.
[22] J. Leskovec, L. Backstrom, R. Kumar, and A. Tomkins. Microscopic evolution of social networks. In *KDD*, 2008.
[23] J. Leskovec and C. Faloutsos. Sampling from large graphs. In *KDD*, 2006.
[24] D. Liben-Nowell and J. Kleinberg. The link prediction problem for social networks. *JASIST*, 58(7):1019–1031, 2007.
[25] R. Lichtenwalter, J. Lussier, and N. Chawla. New perspectives and methods in link prediction. In *KDD*, 2010.
[26] B. Liu. Web data mining. In *Springer*, 2010.
[27] C. Liu, H. chih Yang, J. Fan, L.-W. He, and Y.-M. Wang. Distributed nonnegative matrix factorization for web-scale dyadic data analysis on mapreduce. In *WWW*, 2010.
[28] B. Long, Z. Zhang, and P. Yu. Co-clustering by block value decomposition. In *KDD*, 2005.
[29] L. Lü, M. Medo, C. H. Yeung, Y. Zhang, and Z. Zhang. Recommender systems. *Physics Reports*, 519:1–49, 2012.
[30] L. Lü, L. Pan, T. Zhou, Y. Zhang, and H. Stanley. Toward link predictablility of complex networks. *PNAS*, 118(8):2325–2330, 2015.
[31] L. Lü and T. Zhou. Link prediction in complex networks: A survey. *Physica A*, pages 1150–1170, 2011.
[32] A. K. Menon and C. Elkan. Link prediction via matrix factorization. In *ECML/PKDD*, 2011.
[33] V. Satuluri, S. Parthasarathy, and Y. Ruan. Local graph sparsification for scalable clustering. In *SIGMOD*, 2011.
[34] D. Song, D. Meyer, and D. Tao. Efficient latent link recommendation in signed networks. In *KDD*, 2015.
[35] D. Song, D. Meyer, and D. Tao. Top-k link recommendation in social networks. In *ICDM*, 2015.
[36] J. Tang, S. Chang, C. Aggarwal, and H. Liu. Negative link prediction in social media. In *WSDM*, 2015.
[37] C. Teflioudi, R. Gemulla, and O. Mykytiuk. Lemp: Fast retrieval of large entries in a matrix product. In *SIGMOD*, 2015.
[38] D. Wang, D. Pedreschi, C. Song, F. Giannotti, and A.-L. Barabasi. Human mobility, social ties, and link prediction. In *KDD*, 2011.
[39] R. West, A. Paranjape, and J. Leskovec. Mining missing hyperlinks from human navigation traces: A case study of wikipedia. In *WWW*, 2015.
[40] J. Yang and J. Leskovec. Overlapping community detection at scale: A nonnegative matrix factorization approach. In *WSDM*, 2013.
[41] Y. Yang, N. Chawla, Y. Sun, and J. Han. Predicting links in multi-relational and heterogeneous networks. In *ICDM*, 2012.
[42] L. Yu, Y. Shao, and B. Cui. Exploiting matrix dependency for efficient distributed matrix computation. In *SIGMOD*, 2015.
[43] P. Zhao, C. Aggarwal, and G. He. Link prediction in graph streams. In *ICDE*, 2016.
[44] T. Zhou, L. Lü, and Y. Zhang. Predicting missing links via local information. *The European Physical Journal B*, 71:623–630, 2009.
[45] Z. H. Zhou. *Ensemble methods: Foundations and Algorithms*. Chapman and Hall/CRC Press, 2012.
[46] L. Zhu, D. Guo, J. Yin, G. V. Steeg, and A. Galstyan. Scalable temporal latent space inference for link prediction in dynamic social networks. *TKDE*, 28(10):2765–2777, 2016.


**Liang Duan** is a PhD student in the School of Computer Science and Engineering, Beihang University, supervised by Prof. Shuai Ma. He received his MS degree in computer software and theory from Yunnan University in 2014, and BS degree in computer science and technology from Beihang University in 2009. His current research interests include databases and social data analysis.

**Shuai Ma** is a professor at the School of Computer Science and Engineering, Beihang University, China. He obtained his PhD degrees from University of Edinburgh in 2010, and from Peking University in 2004, respectively. He was a post-doctoral research fellow in the database group, University of Edinburgh, a summer intern at Bell labs, Murray Hill, USA and a visiting researcher of MRSA. He is a recipient of the best paper award for VLDB 2010 and the best challenge paper award for WISE 2013. His current research interests include database theory and systems, social data and graph analysis, and data intensive computing.

**Charu Aggarwal** received the BS degree from IIT Kanpur in 1993 and the PhD degree from Massachusetts Institute of Technology in 1996. He is a research scientist at the IBM T.J. Watson Research Center in Yorktown Heights, New York. He has since worked in the field of performance analysis, databases, and data mining. He has served on the program committees of most major database/ data mining conferences, and served as program vice-chairs of SDM 2007, ICDM 2007, WWW 2009, and ICDM 2009. He served as an associate editor of TKDE from 2004 to 2008. He is an associate editor of TKDD, an action editor of DMKD, an associate editor of SIGKDD Explorations, and an associate editor of KAIS. He is a fellow of the IEEE and a fellow of the ACM.

**Tiejun Ma** is an associate Professor within Centre for Risk Research, Department of Decision Analytics and Risk at the University of Southampton. He received his PhD from University of Edinburgh. Before he joined the University of Southampton, he worked at the Department of Computing, Imperial College London and the Oxford e-Research Centre, University of Oxford. His research focuses on risk analysis and decision-making using quantitative modelling and real-time big data analysis techniques applies to FinTech, Cyber-Risk, and Resilience of distributed systems.

**Jinpeng Huai** is a professor at the School of Computer Science and Engineering, Beihang University, China. He received his Ph.D. degree in computer science from Beihang University, China, in 1993. Prof. Huai is an academician of Chinese Academy of Sciences and the vice honorary chairman of China Computer Federation (CCF). His research interests include big data computing, distributed systems, virtual computing, service-oriented computing, trustworthiness and security.