Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

CommonFinder: A decentralized and privacy-preserving common-friend measurement method for the distributed online social networks

Yongquan Fu [a], Yijie Wang [a,*], Wei Peng [b]

[a] *Science and Technology on Parallel and Distributed Processing Laboratory, College of Computer, National University of Defense Technology, Hunan Province 410073, China*
[b] *College of Computer, National University of Defense Technology, Hunan Province 410073, China*

ARTICLE INFO

Article history:
Received 19 November 2012
Received in revised form 9 October 2013
Accepted 3 February 2014
Available online 7 March 2014

Keywords:
Distributed online social network
Privacy
Common friends
Friend recommendation
Coordinate
Bloom filter

ABSTRACT

Distributed social networks have been proposed as alternatives for offering scalable and privacy-preserving online social communication. Recommending friends in the distributed social networks is an important topic. We propose CommonFinder, a distributed common-friend estimation scheme that estimates the numbers of common-friends between any pairs of users without disclosing the friends' information. CommonFinder uses privacy-preserving Bloom filters to collect a small number of common-friend samples, and proposes low-dimensional coordinates to estimate the numbers of common friends from each user to any other users. Simulation results on real-world social networks confirm that CommonFinder scales well, converges quickly and is resilient to incomplete measurements and measurement noises.

1. Introduction

Online social networks such as Facebook, YouTube, Flickr have become quite popular these days. However, there are also hot debates over the privacy protection of these centralized social services. For example, the service providers can peek at users' profiles at will for targeted advertisements or even sell these sensitive information to third parties for profits. As a result, improving the privacy protection of social networks becomes increasingly important.

Fortunately, borrowing the success of the P2P systems, the distributed online social networks (DOSN for short), e.g., Safebook [1], LotusNet [2], Cuckoo [3], diaspora [4], Peerson [5], Vis-a-Vis [6] have been proposed to better protect users' privacy by hosting the DOSN infrastructures based on decentralized end hosts. The key idea is to store personal data on decentralized nodes and to enforce strict access rules on who can visit a user's profile.

To increase the popularity of distributed online social networks, an open question is how to find potential friends for users, which is known as the **friend recommendation** problem. The centralized online social networks are able to compute possible friends by using the complete knowledge of users' profiles. Unfortunately, for DOSNs, it is commonly believed that end hosts are unwilling to publish their profiles to unknown entities because of the privacy leakage. As a result, it is highly desirable to develop scalable and privacy-preserving methods to quantify the possibility of being friends for DOSN users.

One of the most popular metric for friend recommendation is based on the **Number of Common Friends** (NCF for short), which has been shown to be very effective to recommend new friends or find old friends [7]. Measuring

* Corresponding author. Tel.: +86 13308491230 (Y. Wang).
E-mail addresses: yongquanf@nudt.edu.cn (Y. Fu), wangyijie@nudt.edu.cn (Y. Wang), wpeng@nudt.edu.cn (W. Peng).

http://dx.doi.org/10.1016/j.comnet.2014.02.020
1389-1286/© 2014 Elsevier B.V. All rights reserved.

NCF values in the distributed online social networks is however, a difficult task. First, disclosing friend lists to non-friend users could severely leak users' privacy information [8,9]. Second, the NCF computation has to scale well since most end hosts have limited bandwidth capacity.

Talash [10] computes the common friends based on exchanging friend lists between pairs of users, which not only leaks personal information, but also does not scale well. The PSI approach [11–14] represents a list of friends with coefficients of a polynomial. It then exchanges Homomorphic encrypted coefficients via the communication links, and finally computes the sums of coefficients, which correspond to the set of common items in two friend lists. The PSI approach works well for semi-honest users, but increases the transmission bandwidth and the computation costs.

We propose a scalable and privacy-preserving distributed NCF estimation method called CommonFinder that estimates the NCF values between any pairs of users and is able to select top-k users that have the largest NCF values.

CommonFinder represents a user's friend list with the well-known Bloom filter. Our privacy analysis (Section 8) proves that the Bloom filter provides the differential privacy [15,16] for the friend list: Given a user i's Bloom filter, a curious entity is unable to determine who are user i's friends.

Unfortunately, exchanging the Bloom filters may need several KBytes bandwidth costs in order to control the false positives of the Bloom filter. To further increase the scalability of estimating the NCF values, we propose a distributed maximum margin matrix factorization method to estimate the NCF values by low-dimensional coordinates. As a result, *the transmission size is fixed to be the length of the coordinate that is independent of the length of the friend lists or Bloom filters.*

We model the NCF completion problem with the maximum margin matrix factorization method and provide a systematical design of a distributed NCF prediction method that significantly extends our prior work [17,18]. The MMMF method maps contiguous matrix factorization results to discrete NCF values with adaptive thresholds. These thresholds are learnt during the process of optimizing the matrix-factorization model. To adapt to the decentralization of users, we reformulate the MMMF method in separable objective functions that involve each user's coordinate and a small number of neighbors. For each user, we design a fully decentralized conjugate gradient optimization method to adjust the coordinates of each user that converges quickly.

Finally, we present extensive privacy analysis and simulation results over the real-world social network topologies. Our results show that CommonFinder not only protects users' friend lists, but also significantly improves the prediction accuracy and is more robust against incomplete or incorrect NCF measurements than previous methods.

The rest of the paper is organized as follows. Section 2 introduces the background information. Section 3 next defines the problem of predicting NCF values with preservation of users' privacy. Section 4 then analyzes typical characteristics of social-network data sets. Section 5 next presents an overview of our proposed NCF prediction methods. Section 6 presents the coordinate computation process. Section 7 next applies the decentralized coordinates to select top-k users for recommending friends. Section 8 then measures the degree of the privacy protection offered by CommonFinder. Section 9 next compares CommonFinder's performance with extensive simulation results. Section 10 next confirms CommonFinder is robust to Sybil accounts. Section 11 then summarizes related work. Section 12 concludes the paper. Table 1 shows key parameters.

2. Background

2.1. Distributed online social networks

We introduce the basic characteristics for existing DOSNs.

2.1.1. Social graph

Let a **user** be an online entity that participates in the DOSN. Each user A has a **friend list** S_A, where a **friend** B of a user A is a user B that establishes the **social link** or **friendship link** with user A on the DOSN. The **common friends** of two users A and B are represented by the subset of users that are both friends of user A and B at the same time.

Users and social links form a **social graph**. Each user and his/her friends are adjacent on the social graph, which correspond to one-hop **neighbors** to each other. The **degree** of each user amounts to the size of its neighbors on the social graph. The number of **hops** between two users amounts to the length of the shortest path for these two users on the social graph.

2.1.2. Privacy-preserving message routing

Users' data are usually stored into his/her own computer. For improving the availability of users' data when they are offline, some DOSNs like Safebook replicate each user's data on his/her friends' computers. These friends' replication will be used for offline users.

The social graph is maintained via some kind of Peer-to-Peer substrates. When a user joins the DOSN, each user's computer needs to be registered in the Peer-to-Peer substrate consisting of decentralized computers. Users' real names are decoupled with randomized keys called **identifiers** that are generated by cryptographic hash

Table 1
Notations and their meanings.

k_I	The number of hash functions in a Bloom filter
m_I	The length of a Bloom filter
N	The number of users on the DOSN
$\widehat{\mathbf{X}}$	The coordinate distance matrix
$\mathbf{Y}$	The pairwise NCF matrix between a set of users
L	The maximal NCF value
d	The coordinate dimension
θ	The NCF-mapping thresholds
k	The number of recommended users

functions like SHA-1. The identifier is originally used for routing messages among users, but for the DOSN context, these identifiers also hide persons real-world names, since each identifier can be assumed as a perfectly random variable.

The identifiers are robust against dictionary attacks by curious users. Since the space of the identifiers is huge enough to prohibit the enumeration because of the overwhelming computational overhead. For example, there are 2^{160} kind of possible strings for 160-bit identifiers.

Existing DOSNs usually route messages among users' along the social connections in a hop by hop manner, in order to protect the privacy of the end-to-end communication. Since each logical link corresponds to the real-lift friendship, otherwise, sending a request message from a user to another non-friend user may be eavesdropped by curious or malicious users.

2.2. Establishing friendship on the DOSNs

We introduce how users locate real-life friends on the DOSNs.

2.2.1. Searching friends

A user that logs into an OSN usually searches real-life friends with their names and filters out non-friend users by checking their profiles. The search procedure differs for centralized and distributed OSNs:

- When centralized OSNs receive users' requests, the back-end servers query the OSN's databases to select matched users' profiles.
- For most DOSNs, since users' profiles are stored on decentralized nodes, users' requests are routed among the Peer-to-Peer substrate. The response either locates exactly matched users or fails when timeout events occur.

Unfortunately, searching real-life friends is time-consuming for users, since users have to manually look up every friend he/she has remembered. Therefore, complement to the search process, most OSNs also recommend friends based on users' profiles and friend lists.

2.2.2. Recommending friends

To recommend friends, the OSNs have to locate the set of users that are likely to be future friends for a user. The Number of Common Friends (NCF) is frequently used to quantify the probabilities of being friends between users. Then recommending friends for each user A is realized by selecting the top-k users having the largest NCF values with user A. These top-k users will serve as the recommended future friends. Afterwards, users are prompted with these top-k users in the OSN's web pages. Each user later can send invitations to those people in the list. The NCF values may be appended as a proof of the friendship. Upon receiving the acknowledge of accepting the invitation of being friends, a new friendship link is then established on the DOSNs.

2.3. Maximum margin matrix factorization based collaborative filtering

We next introduce the Maximum Margin Matrix Factorization (MMMF) method [19] for the collaborative filtering problem, which seeks to predict rating scores from users to goods, given a partially available set of rating histories.

2.3.1. Predicting rating scores with matrix factorization

Given N_c customers and N_g kinds of items, we can organize the set of rating scores as a N_c-by-N_g rating matrix $\mathbf{Y}$, where $\mathbf{Y}_{ij}$ denotes the rating score from user i to the item corresponding to the jth column. The objective of the collaborative filtering problem is to complete the missing items in the matrix $\mathbf{Y}$. One of the most popular collaborative filtering approach is the matrix factorization, which trains a low rank matrix $\widehat{\mathbf{X}}$ to approximate the rating matrix $\mathbf{Y}$.

To obtain the matrix $\widehat{\mathbf{X}}$, the matrix factorization has to minimize a **loss function** that quantifies the differences between $\mathbf{Y}$ (training set) and $\widehat{\mathbf{X}}$. For a complete matrix $\mathbf{Y}$, the Singular Value Decomposition (SVD) method yields the optimal low-rank approximation [20]. But when $\mathbf{Y}$ has some missing items, SVD becomes less accurate with increasing missing items, since it treats all missing items as the same constant values.

Worse still, finding the optimal matrix $\widehat{\mathbf{X}}$ for an incomplete matrix $\mathbf{Y}$ is likely to fail due to the **overfitting** phenomenon: *although items in the training set are predicted perfectly, the items out of the training set however receive large prediction errors*. Fortunately, in order to mitigate the overfitting issue, it is well known that we can control the capacity of the loss function by combining the loss function with a regularization model.

Further, the matrix $\widehat{\mathbf{X}}$ are generally contiguous values, while the rating scores in $\mathbf{Y}$ are usually discrete integers, e.g., from one star to five stars. As a result, we have to map the contiguous outputs to discrete rating values, which is generally a **classification** problem. Unfortunately, SVD simply rounds the contiguous values to nearby integers, which may degrade classification errors.

2.3.2. Overview of MMMF

Suppose there are a total number of L rating scores, in order to optimize the classification of rating scores, MMMF introduces a N-by-$(L-1)$ threshold matrix θ. The threshold matrix θ is unknown a prior and must be learnt from the training process. The ith row vector $\vec{\theta}_i$ of the matrix θ is used for mapping the ith row vector of the matrix $\widehat{\mathbf{X}}$ to discrete rating scores.

MMMF classifies rating scores according to the Support Vector Machine (SVM). Suppose that we need to group $\widehat{\mathbf{X}}$ to a binary matrix $\mathbf{Y}$, the SVM groups $\widehat{\mathbf{X}}$ to two separate classes by locating a hyperplane that separates the items from one group with items in the other group. In MMMF, the threshold matrix is analogous to the hyperplane that separates items in $\widehat{\mathbf{X}}$ to discrete rating scores. The optimal hyperplane of a SVM is defined as the one with the largest margin between pairs of items from two groups, where the **margin** amounts to the maximum distance of the slab that is orthogonal with the hyperplane.

2.3.3. Mapping $\widehat{\mathbf{X}}$ to rating scores

MMMF maps the ith row vector in matrix $\widehat{\mathbf{X}}$ by the threshold vector $\vec{\theta}_i$, the ith row vector of the threshold matrix θ. Let $\overrightarrow{\theta_i} = (\theta_{i1}, \ldots, \theta_{i(L-1)})$ be a vector of thresholds sorted in the ascending order. We create N adjacent intervals:

$$(-\infty, \theta_{i1}], (\theta_{i1}, \theta_{i2}], \ldots, (\theta_{i(L-1)}, +\infty)$$

with $\vec{\theta}_i$ as $(L-1)$ separating points.

For each item $\widehat{\mathbf{X}}_{ij}$, its rating score is calculated as the **index** of the interval that contains the item $\widehat{\mathbf{X}}_{ij}$. For example, suppose there are two thresholds $(-0.2, 0.3)$ that separate the whole range of real values into three intervals:

$$(-\infty, -0.2], (-0.2, 0.3], (0.3, +\infty)$$

An item $\widehat{\mathbf{X}}_{ij} = 0.2$ is then mapped to 2, since $\widehat{\mathbf{X}}_{ij}$ is within the second interval $(-0.2, 0.3]$.

2.3.4. Soft margin loss function

An optimal classification of the estimated matrix $\widehat{\mathbf{X}}$ to the rating score matrix $\mathbf{Y}$ means that, each item $\widehat{\mathbf{X}}_{ij}$ should be contained in the interval $\left(\theta_{i(\mathbf{Y}_{ij}-1)}, \theta_{i\mathbf{Y}_{ij}}\right]$, otherwise, we have a misclassification.

The *hard margin matrix factorization* seeks a matrix $\widehat{\mathbf{X}}$ matching observed rating scores by classifying each item $\widehat{\mathbf{X}}_{ij}$ with immediate thresholds $\theta_{i(\mathbf{Y}_{ij}-1)}$ and $\theta_{i\mathbf{Y}_{ij}}$ of the $\mathbf{Y}_{ij}$th interval:

$$\theta_{i(\mathbf{Y}_{ij}-1)} + 1 < \widehat{\mathbf{X}}_{ij} < \theta_{i\mathbf{Y}_{ij}} - 1 \tag{1}$$

for all $ij \in S$. Unfortunately, the hard margin is sensitive to noises. Noises are common in $\widehat{\mathbf{X}}$, since $\widehat{\mathbf{X}}$ consists of inherent uncertainty. Further, Eq (1) is non-differentiable, causing the nontrivial difficulty for the optimization process.

For improving the robustness against noises, the *soft margin matrix factorization* relaxes the hard constraint of Eq. (1) by adding slack variables $\{\xi_{ij} \geqslant 0\}$ for all $ij \in S$:

$$\theta_{i(\mathbf{Y}_{ij}-1)} + 1 - \xi_{ij} < \widehat{\mathbf{X}}_{ij} < \theta_{i\mathbf{Y}_{ij}} - 1 + \xi_{ij} \tag{2}$$

As the optimal solution for Eq. (2) amounts to that by optimizing the **hinge loss function** $h(z) = \max(0, 1-z)$ [19] (the distance from the classification margin), we transform Eq. (2) to:

$$L\left(\mathbf{Y}_{ij}, \widehat{\mathbf{X}}_{ij}\right) = h\left(\widehat{\mathbf{X}}_{ij} - \theta_{i(\mathbf{Y}_{ij}-1)}\right) + h\left(\theta_{i\mathbf{Y}_{ij}} - \widehat{\mathbf{X}}_{ij}\right) \tag{3}$$

Enforcing the hinge loss function makes Eq. (3) become differentiable, therefore, Eq. (3) can be solved via the convex optimization.

Further, MMMF also penalizes the errors of classification with all thresholds in each threshold vector in order to further improve the robustness of Eq. (3):

$$\begin{aligned} L\left(\mathbf{Y}_{ij}, \widehat{\mathbf{X}}_{ij}\right) &= \sum_{r=1}^{L-1} (L(r, \widehat{\mathbf{X}}_{ij})) = \sum_{r=1}^{\mathbf{Y}_{ij}-1} h\left(\widehat{\mathbf{X}}_{ij} - \theta_{ir}\right) + \sum_{r=\mathbf{Y}_{ij}}^{L-1} h\left(\theta_{ir} - \widehat{\mathbf{X}}_{ij}\right) \\ &= \sum_{r=1}^{L-1} h\left(T_{ij}^{r}[r, \mathbf{Y}_{ij}] \cdot \left(\theta_{ir} - \widehat{\mathbf{X}}_{ij}\right)\right) \end{aligned} \tag{4}$$

where the indicator function T_{ij}^{r} determines whether a threshold item r in $\overrightarrow{\theta_i}$ is larger than the rating score $\mathbf{Y}_{ij}$:

$$T_{ij}^{r}[r, \mathbf{Y}_{ij}] = \begin{cases} +1 & r \geqslant \mathbf{Y}_{ij} \\ -1 & r < \mathbf{Y}_{ij} \end{cases} \tag{5}$$

2.3.5. Formulating objective function

MMMF seeks to optimize Eq. (4) with robustness against the overfitting phenomenon. To that end, MMMF finds a matrix $\widehat{\mathbf{X}}$ and a classification matrix θ by minimizing the loss function in Eq. (4) plus two regularized items $(\|U\|_F^2 + \|V\|_F^2)$, where $\|\cdot\|_F$ denotes the Frobenius norm that amounts to the squared root of the sum of squared items in the matrix U or V. The regularized item is used for the capacity control in order to mitigate the overfitting problem of Eq. (4).

The objective function of MMMF can be written as:

$$J(U, V, \theta) = \sum_{r=1}^{R-1} \sum_{i,j \in \Omega, \mathbf{Y}_{ij} > 0} h(T_{ij}^{r}(\theta_{ir} - U_{i*}V_{*j})) + \frac{\alpha}{2}(\|U\|_F^2 + \|V\|_F^2) \tag{6}$$

where α is a trade-off constant.

MMMF solves Eq. (6) based on the Polak–Ribière variant of the nonlinear conjugate gradient methods (PR-CG) in a centralized manner. PR-CG incurs a linear computation complexity, converges quickly and is robust to missing or erroneous inputs [21].

2.4. Differential privacy

The differential privacy [15,16] represents one of the state-of-art privacy-protection techniques. A **database** stores a set of items. Users interact with the database through statistical queries and the database responds query results on stored items. The differential-privacy method provides very strong protection of the privacy: For any two databases D_1 and D_2 that differ only one items x, i.e., $D_1 \oplus D_2 = x$. An adversary is unable to distinguish whether the entity x is in the database D_1 or D_2.

A random function κ provides ϵ-differential privacy if for any two databases D_1 and D_2 having one different item, i.e., where $|D_1 \oplus D_2| = 1$ and all $\phi \subseteq \text{Range}(\kappa)$

$$Pr(\kappa(D_1) \in \phi) \leqslant \exp(\epsilon) \times Pr(\kappa(D_2) \in \phi) \tag{7}$$

where Range denotes the output of the random function κ [15]. The random function κ adds some random noises to the query results. The parameter ϵ is public to all users. Setting ϵ is up to the users' decisions.

The degree of the privacy protection of the random function κ depends on the scale of the added noise and the **sensitivity** of the query result. The sensitivity states the maximum difference of the query results due to adding or removing an entity from the database:

Definition 1. For a function $f : D \to R^d$ over a certain domain D, the sensitivity of a function f is

$$\Delta(f) = \max_{A,B,A \oplus B = 1} \|f(A) - f(B)\|_1 \tag{8}$$

[15]

For real-valued functions, the most popular differential privacy approach [15] is to add independently identical distributed (i.i.d) noises from the Laplace distribution to the query result, which provides ϵ-differential privacy:

Theorem 1. *For a function $f : D \to R^d$ over a certain domain D, a random function*

$$M(x) = f(x) + \left(\text{Laplace}\left(\frac{\Delta(f)}{\epsilon}\right)\right)^d \tag{9}$$

provides ϵ-differential privacy [15].

3. Problem definition

We first introduce how we model the behaviors of users. We next present the requirements of recommending friends based on the numbers of common friends.

3.1. Adversary model

Users Trust Friends. We assume that users trust their friends and allow their friends to visit their profiles including their friend lists, since online friends correspond to the real-world friendships. In the social graph, it means that each user trusts his one-hop neighbors. This model allows for the anonymous communication process on the DOSN, since each user can recursively sends messages to his or her friends.

Users Are Semi-honest. We further assume that users are **semi-honest** [22]: users may be curious to learn the friend lists of non-friend, but they follow the common-friend measurement protocol and do not claim fake friends about his/her friend list. Particularly, each user is able to eavesdrop on the logical communication links connecting that user.

Although the semi-honest model is simple, designing algorithms for DOSNs turns out to be nontrivial due to the large scale and the decentralization of nodes. Further, we will extend our model to tolerate Sybil users that could inject fake friend-list information into the system (see Section 10).

3.2. Disclosing friend lists leaks privacy

To recommend each user a set of candidate friends based on the NCF metric, the OSN needs to know the sizes of the intersections of the friend lists among two-hop users.

Unfortunately, directly disclosing the friend lists to non-friend causes severe privacy leakage. This is because the friendship links on the social graph can serve as the fingerprints to locate users' activities [23]. For example, users' personal profiles could be accurately inferred from their friends even they hide their profiles [8]. Therefore, to better protect users' privacy, centralized OSNs allow for users to hide friend lists from non-friend.

Further, disclosing friend lists to non-friend is also vulnerable to the Identity Cloning Attack (ICA) [9]. Suppose that a user T obtains a large number of friend lists of users on the OSNs. To establish the friend link with each user A, user T creates an account with a friend list by copying from friend lists of user A and some other users for randomization. Then user T will have many common friends with user A, therefore, user T will be likely to be listed in the top-k recommended users of user A according to the NCF metric.

3.3. Problem requirements

We next summarize key requirements for computing pairwise NCFs in DOSN settings:

- **Privacy-preserving**. Friend lists should be treated as privacy information since disclosing them may leak users' privacy and suffer from the ICA attack in Section 3.2. However, hiding the friend lists from non-friend significantly increases the difficulty of computing the NCF values.
- **Decentralized**. As there exist no centralized entities that are able to collect the friend lists of all users on the DOSN, users have to compute the NCF values with their two-hop neighbors on the social graph in a distributed manner. Further, the NCF-computing process among two-hop online users should still work despite some users may join or leave the DOSN at will.
- **Scalable**. Computing the NCF values should require low bandwidth overhead and scale well with increasing friends, since each user only has limited computing and bandwidth resources.

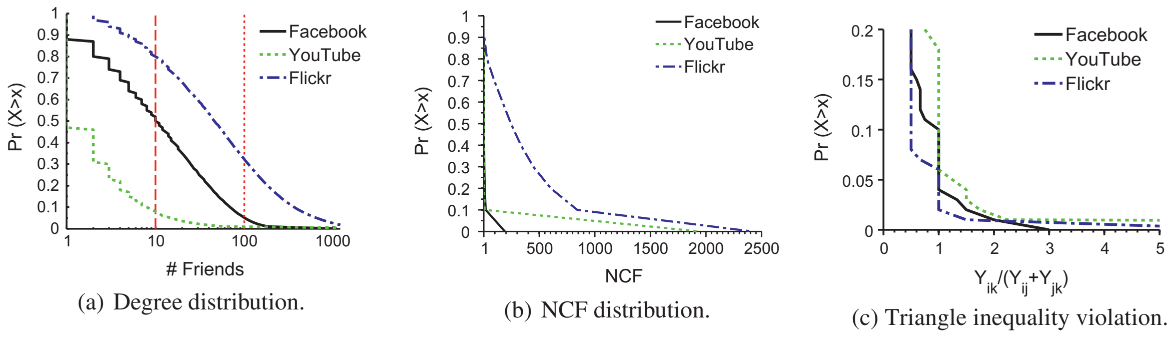
3.4. A naive privacy-preserving NCF-computation solution

One simple decentralized approach is to let each friend B of a user A be the broker of computing the NCF values for user A [?]. User B computes the NCF values between A and those in B's friend list. To that end, user B requests the friend lists from user A and one friend C, and then calculates the intersection of two friend lists of A and C. We can see that the friend lists are always kept at one-hop neighbors. Since each user trusts one-hop neighbors on the social graph, the privacy of the friend list is preserved.

For calculating the common friends between any pair of two-hop users, each broker j needs $O\left(\sum_{p \in List(j)} List(p)\right)$ communication overhead, where $List()$ denotes the list of friends of a user. The computation overhead is.
$O\left(\sum_{p \in List(j)} \sum_{q \in List(j), p \neq q} |List(p)||List(q)|\right)$ for querying the existence of each item over the friend lists. Due to the skewed distributions of friend lists shown in Fig. 1(a), the DOSN faces severely imbalanced computation and transmission costs.

4. Analyzing the NCF metric of the online social networks

To better understand the characteristics of pairwise NCF values, we next analyze the NCF metric with OSN data sets.



(a) Degree distribution. (b) NCF distribution. (c) Triangle inequality violation.

Fig. 1. The CCDF plots of key statistical metrics.

4.1. Data sets

We choose three representative social graphs that are all collected by the online social networks research group in the Max Planck Institute for Software Systems [24–26]. Table 2 summarizes the basic information for each of the networks.

4.2. Distributions of numbers of friends

We first plot the Complementary Cumulative Distribution Function (CCDF) of the number of friends for all users. Fig. 1(a) shows that the sizes of friend lists are quite non-uniform. Most friend lists have modest sizes, but a small number of friend lists could have thousands of friends. For Facebook and YouTube data sets, only 5% of users have more than 100 friends, while for the Flickr data set, over 30% of users have over 100 friends.

4.3. Distributions of numbers of common friends

We then plot the *NCF values* between online users in Fig. 1(b). The NCF values are non-uniformly distributed. For the Facebook and YouTube data sets, most one-hop or two-hop users have only one common friends, but there exists 10% of pairs having a large number of common friends. For the Flickr data set, there are much more common friends than the other two data sets, where around 30% of pairs have more than 500 common friends.

4.4. Triangle inequality violations of numbers of common friends

We next test whether the pairwise NCF values meet the triangle-inequality assumption required by the metric space model. The triangle inequality states that the sum of two NCF values is always not smaller than the third NCF value for a triple (i,j,q): $\mathbf{Y}_{ij} + \mathbf{Y}_{jq} \geqslant \mathbf{Y}_{iq}$. We define the triangle inequality violation (TIV) as $TIV_{ijq} = \frac{\mathbf{Y}_{iq}}{\mathbf{Y}_{ij}+\mathbf{Y}_{jq}}$. When $TIV_{ijq} > 1$, this triple (i,j,q) has a TIV.

From Fig. 1(c), most triples follow the triangle inequality, but there exist about 5% of triples violating the triangle inequalities. Further, for the YouTube and Flickr data sets, some triples may have large degrees of the triangle inequality violation. Since the pairwise NCF values do not fulfill the metric-space requirements, there exist inherent distortions for predicting NCF values with the metric space based methods such as LandmarkMDS.

5. Designing a privacy-preserving NCF computation method for DOSNs

We next present a scalable and privacy-preserving NCF prediction method that measures NCF values with constant bandwidth overhead.

5.1. Intuitions

Since our major objective is to recommend friends based on top-k NCF values, we do not necessarily know the exact list of common friends. Further, the recommendation also tolerates a low degree of prediction errors, as different people have varying subjective views on friends. We can obtain a more efficient NCF computation method with sketches of friend lists that have compact storage structures. For obtaining the pairwise NCF values scalably with the protection of users' privacy, our work applies the well-studied Bloom filters and the collaborative filtering methods to measure NCF values the decentralized OSN settings.

5.2. Architecture

We present the main components in the CommonFinder method. For ease of presentation, assume that a user Bob

Table 2
Dataset summary.

Network	Date	# Users	# Links	Authors
Facebook	12/29/2008, 1/3/2009	60,290	1,545,686	Viswanath et al. [24]
Flickr	11/2/2006, 104 days	2,570,535	33,140,018	Cha et al. [25]
YouTube	1/15/2007	1,157,827	4,945,382	Mislove et al. [26]

logs into the DOSN. As plotted in Fig. 2, at the top level, CommonFinder provides two interfaces for the upper-layer friend recommendation application:

- *Top-k ranking:* Selects at most k two-hop users having the largest numbers of common friends with Bob. These users will serve as the recommended friends for Bob. The number of common friends for a pair of users is computed via the NCF-prediction component.
- *NCF prediction:* Computes the NCF value between Bob and any other DOSN user based on their decentralized coordinates. Each user's coordinates is freely exchanged among DOSN links.

The coordinate-maintenance component manages a user's coordinate in a fully distributed manner. Each user is assigned a random coordinate when he/she logs into the system; afterwards, a daemon incrementally maintains each user's coordinate.

For maintaining the coordinates, the neighbor-management component selects neighbors from online users having non-zero NCF values. This neighbor-management component sends heart-beat messages to these users and pulls back online users' coordinates via the piggyback messages.

The Bloom-filter component represents each user's friend list with a Bloom filter. It uses a bit array to represent a list of items. Each item is hashed to a number of bits according to a set of hash functions. The storage and transmission overhead are reduced by a constant factor compared to the friend lists. To test whether an element is in the friend list, we compute the bits in the bit array using the same set of hash functions, and test whether these bits are all set to ones. If yes, then the Bloom filter returns that the element is in the friend list.

5.3. Exchanging bloom filters for privacy protection

We select the Bloom filter as the sketch structure. Each Bloom filter has a certain probability of **false positives**, i.e.,

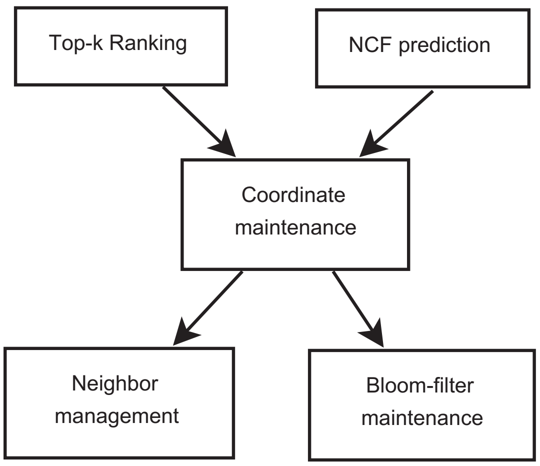


Fig. 2. The main components of the NCF computation method.

it may claim a user that is not in the friend list to be in the list. The false positives are often treated as one drawback of the Bloom filter. However, each user can state that he/she is not represented by the Bloom filter because of the uncertainty caused by false positives. Therefore, we can use the false positives to protect the privacy of users' friend lists, since a curious user cannot exactly infer who is in the friend list by querying the sketch.

Suppose that we represent each friend list with a Bloom filter, two users can exchange their Bloom filters to each other without worrying about the loss of privacy of the friend lists. In Section 8.2, we prove that the Bloom filter provides differential privacy for the friend lists.

5.4. Scale the NCF computation with decentralized prediction

The transmission and computation overhead of the Bloom filter are still linear with the sizes of friend lists. For controlling these overhead, we predict NCF values with decentralized coordinates.

5.4.1. Overview

We compute the NCF values with the Bloom filters for a small fraction of users, then we predict the NCF values for the rest of pairs of two-hop users with low-dimensional coordinates. Accordingly, each user has a low-dimensional coordinate, the NCF value between a pair of users amounts to the coordinate distance between these two users. Estimating NCF values with decentralized coordinates has at least two desirable advantages:

- **Scalability:** The coordinates incur fixed computation and communication costs, which are independent of the sizes of the friend lists.
- **Privacy-protection:** Since coordinates are low-dimensional real-valued vectors, exchanging coordinates only reveals the NCF values without the information of the friend lists.

Let $\mathbf{Y}$ denote a N-by-N matrix that denotes the pairwise NCF values for pairs of users. For correctly predicting NCF values, the coordinates must be optimized with respect to an **objective function**. Selecting an appropriate objective functions is nontrivial:

- **Sparse:** The NCF matrix $\mathbf{Y}$ may be quite sparse, because $\mathbf{Y}_{ij}$ amounts to zero for any pair (i,j) of users that are more than two hops away on the social graph.
- **Distributed:** Since users are decentralized in DOSN, optimizing the coordinates has to be performed in a distributed manner, where each user adjusts his/her own coordinate adaptively with respect to a number of neighbors.

Predicting NCF values is analogous to completing the rating scores by users in the collaborative filtering field, since both are discrete integers. We thus propose to predict NCF values for decentralized pairs of users by borrowing the success of the collaborative filtering field [27]. The MMMF method (see Section 2.3) is robust to missing items, but we have to transform the centralized MMMF method to a decentralized MMMF method since users are decentralized in the DOSN context. We solve the distributed MMMF method with a novel decentralized conjugate optimization method that converges quickly to stable positions.

5.4.2. Coordinate structure

We next introduce the structure of the coordinates. Let S denote a set of DOSN users. Let N be the size of set S. Let d be a constant parameter ($d \ll N$). Recall that the MMMF method computes a N-by-N matrix $\widehat{\mathbf{X}}$ and a N-by-$(L-1)$ threshold matrix θ for predicting discrete rating scores. The ith row vector of matrix $\boldsymbol{\theta}$ is used to separate real values of the ith row vector in matrix $\widehat{\mathbf{X}}$ to discrete values.

- The matrix $\widehat{\mathbf{X}}$ is represented by a linear combination of two low-rank matrices: $\widehat{\mathbf{X}} = U \times V$, where U is a $N \times d$ matrix, V is a $d \times N$ matrix, and $d \ll N$. Each row vector of the matrix $\widehat{\mathbf{X}}$ is represented by the inner product of vectors from U and V, i.e., $\widehat{\mathbf{X}}_{ij} = \sum_{m=1}^{d} u_{im} v_{mj}$.
- The threshold matrix $\boldsymbol{\theta} = \{\theta_{il}\}$, where $i \in S, l \in [1, L-1]$, is learnt from MMMF's optimization process.

For predicting NCF values with the MMMF method, we assign row vectors of the matrices $\widehat{\mathbf{X}}$ and $\boldsymbol{\theta}$ to decentralized users. For each user i, we set its coordinate as two low-dimensional vectors $(\vec{u}_i, \vec{v}_i)$ and a vector of threshold values $(\vec{\theta}_i)$, where $\vec{u}_i$ denotes ith row vector of $\mathbf{U}$, $\vec{v}_i$ denotes the ith column vector of $\mathbf{V}$, and $\vec{\theta}_i$ represents the ith row vector of $\boldsymbol{\theta}$. Accordingly, the coordinate distance $\widehat{\mathbf{X}}_{ij}$ between two users i and j amounts to the dot product $\widehat{\mathbf{X}}_{ij} = u_i v_j$.

5.4.3. NCF prediction

To predict NCF values to another user j, each user i maps the coordinate distance $\widehat{\mathbf{X}}_{ij}$ to a discrete NCF value with his/her own threshold vector $(\vec{\theta}_i)$. Algorithm 1 summarizes key steps to compute the NCF values with coordinates. Step 2 computes the distance $\widehat{\mathbf{X}}_{ij}$. Steps 3–7 determine the index of the interval that contains $\widehat{\mathbf{X}}_{ij}$. We can see that user i does not need user j's threshold vector. Fig. 3 shows an example of computing NCF values by users.

Algorithm 1. Mapping the coordinate distance to the NCF value.

```
1 MapXtoY ($\vec{u}_i$, $\vec{v}_j$, $\vec{\theta}_i$)
  **input**: $\vec{u}_i$: user $i$'s coordinate component, $\vec{v}_j$: user $j$'s
  coordinate component, $\vec{\theta}_i$: User $i$'s threshold vector.
  **output**: $y$
2 $\widehat{\mathbf{X}}_{ij} = \vec{u}_i * \vec{v}_j$;
3 $y = 1$;
4 **for** $l = 1 \rightarrow (L-1)$ **do**
5     $y += \widehat{\mathbf{X}}_{ij} \geqslant \theta_i(l) ? 1 : 0$;
6 **end**
7 **return** $y$;
```

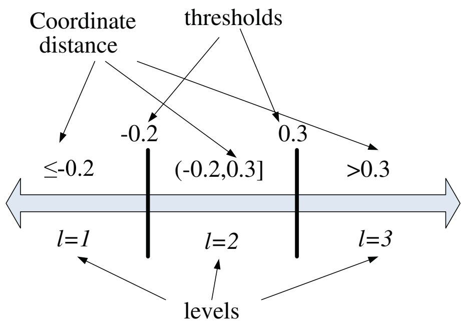


Fig. 3. Mapping the coordinate distance to NCF values.

5.4.4. Coordinate dissemination

In order to predict pairwise NCF values, users need to exchange the low-dimensional vectors corresponding to the matrix factorization model, but hides each user's threshold vector from other users. For example, each user i can request any other user j's coordinate components $(\vec{u}_j, \vec{v}_j)$. However, we do not allow each user to request other user's threshold vector. As a result, each user is only able to predict NCF values from himself/herself to other users, but cannot determine the NCF values of an arbitrary pair of users.

Sending the threshold vectors could leak users' friendship information. Since knowing the nonzero NCF values of a user pair indicates that two users are either friends or friends of friends, thus the social contacts of users can be easily derived. Consequently, users' privacy setting could be violated and even worse, users' otherwise secret information could be inferred as discussed in Section 3.2.

5.5. Benefits of decentralized NCF prediction

We next summarize the benefits of the decentralized NCF prediction process.

First, CommonFinder protects users' privacy via two complement data structures. Bloom filters anonymize users' identifiers and hide the existence of any specific users with a degree of false positives. Coordinates do not contain any information about who is in a friend list, therefore exchanging coordinates among non-friend users does not disclose users' friend lists.

Second, CommonFinder fully utilizes the low-dimensional coordinates for calculating pairwise NCF values that have constant transmission bandwidth costs and computation overhead. Therefore, CommonFinder scales well with decentralized users on the DOSNs.

Finally, as NCF has been a very popular metric for recommending friends on centralized OSNs, CommonFinder can serve as a replacement for existing NCF computation in centralized OSNs, which can improve the scalability and increase the responsiveness to users' requests.

6. NCF Prediction with decentralized coordinates

We next present how to estimate the NCF between any pair of users combining the Bloom filters and the decentralized coordinates. We first measure a small number of NCF values with the Bloom filters as inputs to maintain the coordinates. We next present the neighborhood management process with respect to which the coordinates are optimized. We then present a distributed algorithm to adjust each user's coordinate adaptively and scalably.

6.1. Measuring NCF values based on bloom filters

We represent each friend with an identifier created by cryptographic hash functions like SHA-1 for data routing in the DOSN. Then each friend list is represented by a Bloom filter. We use the Dynamic Bloom filter (DBF) [28] to represent a dynamic set of friends because of its simplicity and low computational complexity. The DBF [28] includes multiple standard Bloom filters where each standard Bloom filter stores only c items.

Each user then obtains other nodes' Bloom filters and queries them to estimate the set of common friends with his/her own friend list:

- Alice and Bob exchange their Bloom filters.
- Alice queries Bob's Bloom filter with her friend list, and vice versa. The subset of friends represented in the Bloom filter is returned as the estimated common friends.

Assume that an attacker maintains a dictionary of identifiers crawled over the DOSN. The attackers cannot exactly know whether an identifier is indeed in the set, since the Bloom filter is a probabilistic data structure with a false positive probability for a query. The false positives are statistical valid for any query, therefore a user is always able to deny that he/she is a friend of another user due to the false positives.

6.2. Neighbor management

Each user i maintains a set S_i of neighbors with a maximal capacity S_m ($S_m = 20$ by default). Each neighbor is represented with three fields: **the identifier, the Bloom filter and the coordinate**.

The neighbor set S_i covers user i's friends. If a user has too few friends, we also include a randomized set of two-hop users as neighbors. But we deliberately set users' friends to have a higher priority than two-hop users since friends are trusted. The two-hop users help if there are too few friends. Since we probe NCF values with Bloom filters, users' privacy is still preserved.

When a user i joins the DOSN system, he/she first finds real-life friends that have already registered in the DOSN. After establishing several links with friends on the social graph, user i puts these friends into his/her neighbor set S_i, otherwise, this user will restart the search process later.[1]

[1] For bootstrapping users' online friends, the DOSN may also advertise popular users' profiles to each users.

Each online user has a daemon process that triggers a periodical procedure to sample two-hop users on the social graph for user i. The daemon process randomly selects one of user i's friends, say user j, as the counterpart for communication. Then user i's daemon requests the daemon process at user j to send the contact address of a friend of user j. The daemon process at user j then selects a user q uniformly at random from his own friend list and piggybacks user q's data to the daemon process at user i. Finally, the daemon process at user i puts user q's data into set S_i. If user j does not respond within 20 s, the daemon process at user i removes user j from his/her neighbor set.

6.3. Decentralized MMMF optimization

We next formulate the objective function of optimizing users' coordinates for accurate NCF prediction. We formulate the problem of the NCF prediction with the MMMF method. Readers are referred to Section 2.3 for the background of the MMMF method.

We formulate the objective of predicting NCFs with MMMF's optimization function in Eq. (6) as:

$$J(u,v,\theta) = \sum_{r=1}^{L-1}\sum_{(i,j)\in\Omega} h\left(T_{ij}^{r}[r,\mathbf{Y}_{ij}]\cdot(\theta_{ir}-\vec{u}_i\vec{v}_j)\right) + \frac{\lambda}{2}\left(\sum_{i=1}^{N}\left(\|\vec{u}_i\|_F^2+\|\vec{v}_i\|_F^2\right)\right) \tag{10}$$

where Ω denotes the set of pairs of users with observed NCF values and λ is a regularized constant. We can see that Eq. (10) requires all users' information, as a result, solving Eq. (10) belongs to the centralized optimization problem where an entity collects the NCF matrix $\mathbf{Y}$ and computes the coordinates for all users. Unfortunately, such a centralized formulation not only leaks the friendship information of all users, but also does not adapt to the decentralization of DOSN users where users may leave the system dynamically.

As a result, we need a decentralized optimization process that is able to adapt the dynamics of DOSN users. To that end, we decompose Eq. (10) to a set of distributed optimization problems that are solved by decentralized users. The key idea is to let each user iteratively optimize his/her own coordinate with a set of neighbors' coordinates in a fully decentralized manner. We decompose (10) into N sub-objective consisting of coordinates and NCF measurements among each user i and his/her neighbors S_i:

$$J_{\mathbf{Y}}\left(\vec{u}_i,\vec{v}_i,\vec{\theta}_i\right) = \sum_{r=1}^{L-1}\sum_{j\in S_i} h\left(T_{ij}^{r}[r,\mathbf{Y}_{ij}]\cdot(\theta_{ir}-\vec{u}_i\vec{v}_j)\right) + \frac{\lambda}{2}\left(\|\vec{u}_i\|_F^2+\|\vec{v}_i\|_F^2\right) \tag{11}$$

The objective $J_{\mathbf{Y}}\left(\vec{u}_i,\vec{v}_i,\vec{\theta}_i\right)$ seeks to optimize user i's coordinates. Since Eq. (11) is defined for each user i separately, each user i is able to optimize Eq. (11) in a distributed manner based on the local information of its neighbors.

6.4. Distributed coordinate maintenance

We adjust each user's coordinate in a fully distributed manner via a novel implementation of the PR-CG method. We first present how we transform the centralized PR-CG method to a distributed algorithm for adjusting users' coordinates. We next adapt the coordinate maintenance process to the dynamics of decentralized coordinates.

6.4.1. Decentralized conjugate gradient optimization

PR-CG is an iterative optimization algorithm. To minimize a nonlinear function $f(x)$, it iteratively updates the vector $\vec{x}$ according to the conjugate direction of $\vec{x}$ until reaching a local minima.

Transforming the iterative PR-CG process to a distributed PR-CG method (D-PR-CG) algorithm is straightforward. Each user i optimizes Eq. (11) in separate rounds. For each round, user i adjusts the vector $\vec{x}_i$ once by one round of the PR-CG method. Further, every two adjacent rounds is separated by τ seconds.

(i) Setting Parameters.

We define the vector $\vec{x}_i$ for user i with the concatenation of coordinate components of i, i.e.,

$$x_i = \left[\vec{u}_i; \vec{v}_i; \vec{\theta}_i\right]$$

We next derive the gradient of the vector $\vec{x}_i$ as

$$\nabla_x J_{\mathbf{Y}}(x_i) \to \left[\frac{\partial J_{\mathbf{Y}}}{\partial u_i}; \frac{\partial J_{\mathbf{Y}}}{\partial v_i}; \frac{\partial J_{\mathbf{Y}}}{\partial \theta_i}\right] \tag{12}$$

where $\frac{\partial J_{\mathbf{Y}}}{\partial u_i}$, $\frac{\partial J_{\mathbf{Y}}}{\partial v_i}$ and $\frac{\partial J_{\mathbf{Y}}}{\partial \theta_i}$ are partial derivatives with respect to the coordinate components $\vec{u}_i$, $\vec{v}_i$ and $\vec{\theta}_i$:

$$\frac{\partial J_{\mathbf{Y}}}{\partial u_{iz}} \leftarrow \lambda u_{iz} - \sum_{j \in S_i} \sum_{r=1}^{L-1} T_{ij}^r[r, \mathbf{Y}_{ij}] \cdot h\prime\left(T_{ij}^r[r, \mathbf{Y}_{ij}] \cdot \left(\theta_{ir} - \vec{u_i}\vec{v_j}\right)\right) v_{jz}$$

and

$$\frac{\partial J_{\mathbf{Y}}}{\partial v_{iz}} \leftarrow \lambda v_{iz} - \sum_{j \in S_i} \sum_{r=1}^{L-1} T_{ji}^r[r, \mathbf{Y}_{ji}] \cdot h\prime\left(T_{ji}^r[r, \mathbf{Y}_{ji}] \cdot \left(\theta_{jr} - \vec{u_j}\vec{v_i}\right)\right) u_{jz}$$

for $z \in [1, d]$, and

$$\frac{\partial J_{\mathbf{Y}}}{\partial \theta_{ir}} \leftarrow \sum_{j \in S_i} T_{ij}^r[r, \mathbf{Y}_{ij}] \cdot h\prime\left(T_{ij}^r[r, \mathbf{Y}_{ij}] \cdot \left(\theta_{ir} - \vec{u}_i \vec{v}_j\right)\right)$$

for $r \in [1, L-1]$.

(ii) Bootstrapping.

First, we initialize necessary parameters for bootstrapping the optimization process. Each user i's coordinate is initialized as a randomized vector, where each dimension is selected from the interval $[-10, 10]$. Then, user i adjusts his coordinate periodically, separated by τ second.

We set the steepest direction and the conjugate direction for each user i:

- Compute the **steepest direction** Δx as the reverse direction of the gradient $\nabla_x J_{\mathbf{Y}}(x_i)$:

$$\Delta x = -\nabla_x J_{\mathbf{Y}}(x_i)$$

since the reverse of $\nabla_x J_{\mathbf{Y}}(x_i)$ indicates the direction of the maximum decrease for $f(x)$.

- Set the **conjugate direction** Λx_1 with the steepest direction Δx in the first round.

(iii) Incremental Adjustment.

Using the steepest direction to adjust the coordinate is sensitive to input noises such as the coordinate oscillation or erroneous NCF values. We therefore use the conjugate direction to adjust the coordinates. At each round $l > 1$, we updates the **conjugate direction** Λx of vector $\vec{x}$ as the weighed sum of the steepest direction and the conjugate direction of the last round:

$$\Lambda x_l = \Delta x_l + \beta_l \times \Lambda x_{l-1} \tag{13}$$

where $\beta_l > 0$ controls the capacity of the conjugate direction in the current round. We use a revised Polak–Ribière scalar:

$$\beta_i \to \max\left\{0, \frac{\Delta x_i(l)^T(\Delta x_i(l) - \Delta x_{i-1}(l-1))}{\Delta x_i(l-1)^T \Delta x_i(l-1)}\right\} \tag{14}$$

that automatically resets the conjugate direction. The parameter β_l trades off the robustness and the convergence speed: *Decreasing β_l favors the steepest direction, but becomes more sensitive to noises, while increasing β_l reduces the effect of the steepest direction, but slows down the convergence speed of the optimization process.*

We determines the step α_l of moving the vector $\vec{x}$. We use the well-studied line-search method [21] to be aware of the convergence status of the vector $\vec{x}$. the line search method minimizes the objective function (11). The more accurate the vector $\vec{x}$ is, the smaller the movement step becomes. Therefore, the coordinates become more stable with increasing rounds of adjustments.

We then adjust the vector $\vec{x}$ at a small distance with the conjugate direction $\Lambda_l(x)$ and the movement step α_l:

$$\vec{x} = \vec{x} + \alpha_l \times \Lambda_l(x) \tag{15}$$

Algorithm 2 summarizes the above process for adjusting the coordinates. Step 2 calculates the concatenation $\vec{x}_i$ of the coordinate components of user i. Then step 3 computes the new steepest direction Δ as the reverse direction of the current gradient of $\vec{x}_i$. Step 4 then computes the Polak-Ribière scalar β that trades off well between the robustness of the optimization process and the convergence speed. Step 5 next updates the conjugate direction with the steepest direction Δ and the conjugate gradient of the last round. Then step 6 calculates the moving step. Step 7 next adds the vector $\vec{x}_i$ with α_i times of conjugate gradient direction. Finally, steps 8–12 cache the steepest direction and the conjugate direction and then reconstruct user i's coordinate components based on the updated vector $\vec{x}_i$.

Algorithm 2. Distributed Polak–Ribière nonlinear conjugate gradient optimization algorithm.

```
1  Update $\vec{u}_i, \vec{v}_i, \vec{\theta}_i, \Delta x_i, \Lambda x_i, S_i, \mathbf{Y}$
   input: a user $i$'s current coordinate $\vec{u}_i$, $\vec{v}_i$, $\vec{\theta}_i$, a user $i$'s
     steepest direction $\Delta x_i$, a user $i$'s conjugate direction
     $\Lambda x_i$, the set of neighbors $S_i$, the NCF values from a
     user $i$ to its neighbors in $S_i$, the coordinates of
     neighbors in $S_i$.
2  $x_i \leftarrow \left[\vec{u}_i; \vec{v}_i; \vec{\theta}_i\right]$;
3  $\Delta \leftarrow -\nabla_x J_{\mathbf{Y}}(x_i)$;
4  $\beta \leftarrow \frac{\Delta^T(\Delta - \Delta x_i)}{\Delta x_i^T \Delta x_i}$;
5  $\Lambda \leftarrow \Delta + \beta \Lambda x_i$;
6  $\alpha_i \leftarrow \arg\min_{\alpha_i} J_{\mathbf{Y}}(x_i + \alpha_i \Lambda)$;
7  $x_i \leftarrow x_i + \alpha_i \Lambda$;
8  $\Delta x_i \leftarrow \Delta$;
9  $\Lambda x_i \leftarrow \Lambda$;
10 $\vec{u}_i \leftarrow x_i[1 : d]$;
11 $\vec{v}_i \leftarrow x_i[(d+1) : 2d]$;
12 $\vec{\theta}_i \leftarrow x_i[(2d+1) : (2d+L)]$;
```

From Algorithm 2, the lengths of steepest direction and the conjugate direction amount to the twice size of the coordinate. The overall space overhead for Algorithm 2 is $O(|S_i| \times (2d + L))$. The communication complexity is linear with the number of neighbors. In practice, we limit the number $|S_i|$ of friends, the length d of coordinate vector to be around 20, the communication complexity and the storage overhead of Algorithm 2 is quite modest.

6.4.2. Minibatch based coordinate update

As users' coordinates are modified at each round by Algorithm 2, we have to adapt to the dynamics of users' coordinates. One simple approach is to let each user i probe the NCF values to neighbors in set S_i and to request these users' coordinates at each round. However, frequently measuring NCF values or requesting coordinates to all neighbors incurs heavy traffics at nodes with large friend lists. Therefore, we have to adapt to the coordinate modifications of neighbors and control the measurement traffics.

We follow the well-studied *minibatch* approach [29–31] proposed in the network-coordinate field. The minibatch approach reuses historical positions of most neighbors' coordinates, since historical coordinates become closer to the current positions as coordinates converge to stable positions.

Each user caches the NCF values and coordinates of all neighbors in the memory; at each round, each user refreshes this cache by probing the NCF value to one neighbor and requests this neighbor's coordinate, and then updates his coordinate with respect to all cached neighbors according to recorded historical records. The minibatch approach works quite well in practice. For instance, CommonFinder's coordinates converge within twenty rounds.

7. Top-*k* ranking

Having presented the decentralized coordinates, we next show how to use coordinates to select top-k users having the highest NCF values as recommended friends on the DOSN.

A naive approach is to aggregate all coordinates of two-hop users and to compute the descending order of coordinate distances. Unfortunately, since we only care about the top-k users, most of the communication is wasteful.

We present a *divide-and-conquer* based method to reduce the transmission bandwidth costs.

Spreading: A user (called the **initiator**) spreads the ranking task to all of his/her friends. Upon receiving the ranking task, each user j filters out those who have already been the initiator's friends and selects k users from the rest of friends having the largest NCF values with the initiator. The NCF values are computed with the decentralized coordinates. Finally, user j piggybacks the initiator with the top-k users.

Merging: The initiator then aggressively merges the lists of top-k users from his/her friends. The top k users in the merged list will be returned as the top-k recommended friends. The initiator initializes an empty list $\bar{List}$ and then merges each newly received list $\widehat{List}_j^k$ sent from his/her friend j. Both $\bar{List}_i$ and $\widehat{List}_j^k$ are represented with the linked-list data structure for flexible expansion.

Theorem 2 confirms that merging the local top-k users yields the correct results. Extending to cases where some NCF values are identical is straightforward, since users having the same NCF values with the initiator are exchangeable with each other.

Theorem 2. *For ease of presentation, we assume that pairwise NCF values are different. Suppose that an initiator i seeks the top-k ranking list of users. By merging local ranking lists from friends of the initiator i, user i has the correct top-k ranking list.*

Proof. Let (i,j) and (j,q) be two pairs of friends in the social graph. Assume that user q is among the top-k ranking list for i, but is pruned from the local merging step by user j. We will prove by contradiction.

Since user q is pruned by user j, user j's top-k ranking list for i must exclude user q. In other words, user j has at least k friends having larger NCF values with i than user q. As a result, user q must not be among the top-k ranking list for user i. Thus, a contradiction happens. Therefore, user q must be included in the top-k ranking list of the initiator i, the correct top-k list is preserved. □

8. Privacy analysis

8.1. Coordinates hide users' friend lists

Recall that in CommonFinder, each user i can send his coordinate component $(\vec{u}_i, \vec{v}_i)$ to any other user. The $(\vec{u}_i, \vec{v}_i)$ components help other users compute the number of common friends with user i (see Algorithm 1).

Intuitively, since $(\vec{u}_i, \vec{v}_i)$ are two low-dimensional real-valued vectors, which do not encode any information about user i's friend list. Therefore, disclosing the $(\vec{u}_i, \vec{v}_i)$ components of user i to his non-friend does not leak user i's friend list.

CommonFinder requires to hide each user's threshold vector $\vec{\theta}$. This is because for each user i, his/her threshold vector $\vec{\theta}$ is only useful for computing the numbers of common friends between i and other users. Further, disclosing $\vec{\theta}$ leaks users' friend links. This is because for each user i, a malicious user can compute the set of users having some common friends with user i, indicating that these users are at most two-hop away from user i on the social graph. Therefore, these users are either user i's friends or friends of user i's friends. Consequently, the friendship links are disclosed to malicious users.

Suppose a malicious user collects user i's coordinate components $(\vec{u}_i, \vec{v}_i)$, but does not known user i's threshold vector. We can see that this malicious user cannot determine the number of common friends from i to other users, since only the threshold vector uniquely determines the NCF values.

8.2. Bloom filters yield differential privacy

We next show that the Bloom filters also protect the privacy of users' friend lists. Intuitively, since Bloom filter may introduce false positives, i.e., the estimation may not coincide with the ground-truth common friends. For around 99% of all cases, the Bloom filter is able to predict correct common friends. Moreover, we can turn the false positives into a way of protecting the privacy: *a user can deny that a user is his (or her) friend, since the Bloom filter could fail due to the false positives.* Increasing the size of the Bloom filter may increase the accuracy of estimation, but still fail to distinguish whether a user is in the Bloom filter with 100% certainty.

8.2.1. Sensitivity of a Bloom filter

Let m_I and k_I be the length of the bit array I and the number of hash functions for a Bloom filter. Let $f : S \to BF(S)$ be a function that maps a set of identifers S to a Bloom filter $BF(S)$. Given two sets $S_1 = \{x_1, \ldots, x_n\}$, $S_2 = \{x_1, \ldots, x_n, x_{n+1}\}$ that differ in only one identifier x_{n+1}.

We compute the L_1 difference of the bit arrays of two Bloom filters $BF(S_1)$ and $BF(S_2)$:

$$g(S_1, S_2) = \sum_{i=1}^{m_I} |I_{BF(S_1)}(i) - I_{BF(S_2)}(i)| \tag{16}$$

in order to reveal the difference of the bit arrays by hashing S_1 and S_2 into two Bloom filters. Theorem 3 states the expected number of different bits for two Bloom filters representing two sets S_1 and S_2 that differ in one item.

Theorem 3. *Given two sets S_1 and S_2 satisfying that $S_2 = S_1 \cup \{x_{n+1}\}$. The expected number of different bits between the Bloom filter $BF(S_1)$ and $BF(S_2)$ amounts to $E[g(S_1, S_2)] = k_I e^{-\frac{nk_I}{m_I}}$.*

Proof. Let I_1 and I_2 denote the bit arrays for the Bloom filters $BF(S_1)$ and $BF(S_2)$, respectively.

We first compute the probability $P(i)$ that two ith bits in the bit arrays I_1 and I_2 are different, i.e., $I_1[i] = 0$ and $I_2[i] = 1$. In other words, $P(i)$ is the probability that the ith bit in $BF(S_2)$ does not change after inserting the item $\{x_{n+1}\}$, which amounts to the probability that $I_1[i]$ is still zero after inserting n keys:

$$p(i) = e^{-\frac{nk_I}{m_I}} \tag{17}$$

Since after inserting n keys into a sized-m_I Bloom filter with k_I hash functions, the probability that a bit is still zero amounts to $\left(1 - \frac{1}{m_I}\right)^{nk_I} \approx e^{-\frac{nk_I}{m_I}}$.

We next show the expected L_1 difference between the Bloom filters $BF(S_1)$ and $BF(S_2)$. Let k_I hashed positions for the item x_{n+1} be $\{a_1, \ldots, a_{k_I}\}$.

Let Z_l be the indicator function denoting whether two a_lth bits in $BF(S_1)$ and $BF(S_2)$ are identical:

$$Z_l = \begin{cases} 1 & I_1(a_l) = I_2(a_l) \\ 0 & \text{else} \end{cases},$$

where $a_l \in [1, m_I]$. Let Z be the sum of k_I indicator functions:

$$Z = \sum_{l=1}^{k_I} Z_l \tag{18}$$

We compute the expected number $E[Z]$ of different bits for $BF(S_1)$ and $BF(S_2)$ as:

$$E[Z] = E\left[\sum_{l=1}^{k_I} Z_{a_l}\right] = \sum_{l=1}^{k_I} E[Z_{a_l}] = \sum_{l=1}^{k_I} (1 \times \mathrm{P}(a_l)) = k_I e^{-\frac{nk_I}{m_I}} \quad \square \tag{19}$$

We next plot the expected number $E[Z]$ of different bits as the number of items n increases. From Fig. 4, the number of different bits is mostly lower than four. Further, the expected number of different bits drops quickly as the number of item increases.

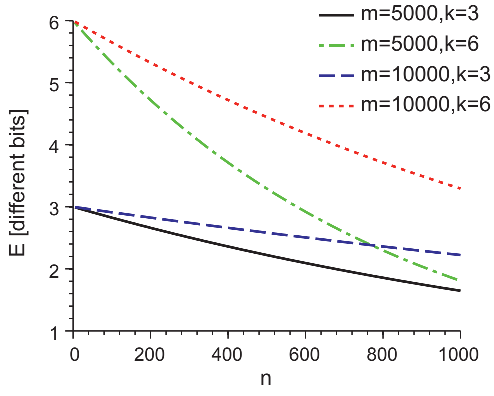


Fig. 4. Expected number of different bits with increasing items n.

8.2.2. Differential privacy of bloom filters

Having shown that the Bloom filters are slightly modified by adding or removing an item, we next derive the capability of the differential privacy for the Bloom filter (see Section 2.4 for brief introduction).

Theorem 4. *Given a sized-m_l Bloom filter with k_i hash functions. Let n be the number of items inserted into this Bloom filter. This Bloom filter provides $\left(k_l \ln\left(1-\exp\left(-\frac{nk_l}{m_l}\right)\right)\right)$-differential privacy.*

Proof. For two sets S_2 and $S\prime_2$ that differ in one item, we compute the ratio of the probabilities that two Bloom filters representing sets S_2 and $S\prime_2$ having the same bit array:

$$\begin{aligned}\frac{\Pr(I_{S_2}=I)}{\Pr\left(I_{S'_2}=I|S'_2=S_2\cup\{y\}\right)} &= \Pr(y \text{ does not change the bit array of } I_{S_2})\\ &= \prod_{i=1}^{k_l}(\Pr(I_{S_2}[h_i(y)]=1))\\ &= \left(\left(1-\exp\left(-\frac{nk_l}{m_l}\right)\right)\right)^{k_l}\\ &= \exp\left(k_l\ln\left(1-\exp\left(-\frac{nk_l}{m_l}\right)\right)\right)\end{aligned}$$

We see that

$$\begin{aligned}\Pr(I_{S_2}=I) \leqslant{}& \Pr\left(I_{S'_2}=I|S\prime_2=S_2\cup\{y\}\right)\\ &\times\exp\left(k_l\ln\left(1-\exp\left(-\frac{nk_l}{m_l}\right)\right)\right)\end{aligned}$$

Therefore, representing two sets that differ in one item with the Bloom filters provides $\left(k_l \ln\left(1-\exp\left(-\frac{nk_l}{m_l}\right)\right)\right)$-differential privacy. □

Theorem 4 confirms that using the Bloom filter representing a friend list provides differential privacy for users in this friend list. As a result, a malicious user cannot be certain about who is a user's friends.

9. Evaluation

We next test CommonFinder's performance with extensive simulations.

9.1. Experimental setup

9.1.1. Evaluation process

Our evaluation tries to ask whether CommonFinder can estimate the number of common friends accurately and scalably: (1) How does the Bloom filter based NCF estimation method scale for real-world social networks? (2) Is the decentralized coordinate more accurate than existing NCF estimation methods? (3) How does CommonFinder scale for real-world social networks.

The simulation seeks to be general enough for any existing DOSN proposals. Therefore, rather than integrating with specific DOSNs, the simulation focuses on predicting NCF values for pairs of users. The simulation takes the social graph among users as the input and then predicts NCF values among two-hop users with CommonFinder and related methods.

Unfortunately, since most DOSNs are still under developments, no DOSN data sets are publicly available yet. Since DOSNs have consistent social functionalities with the centralized OSNs, we use the wide-adopted data sets of centralized OSNs introduced in Section 4. These data sets correspond to static social graphs, which do not reveal when each user adds friends. As a result, our simulation assumes that each user has all friends that are available on the social graph.

9.1.2. Comparison methods

We compare CommonFinder with two representative NCF prediction methods, LandmarkMDS, a geometric coordinate based method [32], and ProximityEmbed, a matrix-factorization based method [33]. We configure the parameters of LandmarkMDS and ProximityEmbed according to the recommended parameters.

We represent each user with a 160-bit randomized string generated with a SHA-1 cryptographic hash function. We set CommonFinder's default parameters in Table 3. We choose the coordinate dimension to be five, since further increasing the dimension does not improve the accuracy. We set the default number of neighbors for adjusting the coordinate to 20, which is modest compared to the sizes of most users' friend lists. We configure the regularized parameter λ to 0.3, which yields stable accuracy for optimizing the decentralized coordinates. We select neighbors randomly for each user from the whole set of candidate users due to its robustness. Finally, we evaluate CommonFinder's accuracy after each user's coordinate is updated in 60 rounds. In fact, all users' coordinates have converged to stable positions within 20 rounds.

9.1.3. Performance metrics

To quantify the accuracy of estimations, we use the popular metric **Normalized Mean Average Error** (NMAE), which is defined as

$$\frac{\sum_{(i,j):\mathbf{Y}_{ij}>0}|\mathbf{Y}_{ij}-\widehat{\mathbf{Y}}_{ij}|}{\sum_{(i,j):\mathbf{Y}_{ij}>0}\mathbf{Y}_{ij}} \qquad (20)$$

where $\mathbf{Y}_{ij}$ is the ground-truth value, $\widehat{\mathbf{Y}}_{ij}$ is the estimated value. Smaller NMAE values correspond to higher prediction accuracy.

All experiments are performed on a laptop with 2.13 GHz CPU and a 2 GB RAM. We have implemented all related methods with Matlab 7.0. The experiments are

Table 3
CommonFinder parameters.

Parameter	Value
Coordinate dimension d	5
Number of neighbors for adjusting coordinates	20
Regularized parameter λ	0.3
Neighbor-selection method	Random
Rounds of coordinate updates	60

repeated in ten times and we report the average results and the corresponding standard deviations.

9.2. Baseline performance of measuring pairwise NCF values

To tune the coordinate positions, CommonFinder uses the DBF to probe the NCF values from a user to his friends. Moreover, DBF also lists the set of common friends that are not used in the coordinate computation process. The obtained NCF values to a set of neighbors are then treated as the inputs to adjust each user's coordinates. Suppose the NCF measurements by the DBF is inaccurate, the coordinate adjustment will be impaired accordingly. In other words, DBF sets up the upper bound for the prediction accuracy of the NCF values.

We therefore first characterize the baseline performance of measuring the NCF values for the CommonFinder. We focus on DBF's performance with $n \leqslant 1000$, since most of the number of online users's friends is below 1000 as shown in Section 4.1. To scale well with dynamic items, Dynamic Bloom Filter (DBF) [28] includes multiple CBF where each CBF stores only c items. After having c inserted items, a new CBF is added to store subsequent insertions. The false positive probability of a DBF is:

$$f_{m,k,c,n} = 1 - \left(1 - \left(1 - e^{-kc/m}\right)^k\right)^{\lfloor n/c \rfloor} \left(1 - \left(1 - e^{(-k(n - c\lfloor n/c \rfloor))/m}\right)^k\right) \tag{21}$$

9.2.1. Sensitivity to Bloom-filter parameters

To investigate DBF's sensitivity to parameter settings, we compute DBF's false positive probability and calculate its storage size with increasing SBF size m and the number c of items inserted into each SBF.

First, from Fig. 5(a), increasing the SBF size m from 0 to 1000 bits significantly decreases the false positive probability of DBF, compared to DBFs with SBF size $m > 1000$. As a result, we need to select a SBF size m bigger than 1000 bits in order to control the false positive probability. Besides, decreasing the upper bound c of the number of items per SBF also decreases the degradation of the false positive probability of DBF.

Second, from Fig. 5(b), the storage size of the DBF increases with decreasing number c of items inserted into a SBF or the SBF size m. For $c \leqslant 50$, the storage size increases quickly with increasing SBF size m. As a result, we need to choose the number c of items to be bigger than 50 in order to control the increment of the storage size.

We next show the detailed variation of performance by fixing c or m. From Fig. 6(a) and (b), we see that increasing the upper bound c exponentially increases the false positive probability of DBF, but decreases the storage size significantly. As a result, there is a natural trade off between the false positive probability and the storage size. From Fig. 6(c) and (d), increasing the SBF size m exponentially decreases the false positive probability, but also increases the storage size. As a result, there are no optimal choices for DBF's parameters, we need to select c and m balancing the false positive probability and the storage size. For the rest of the paper, we choose $c = 100$, $m = 2000$ as the default parameters of the DBF.

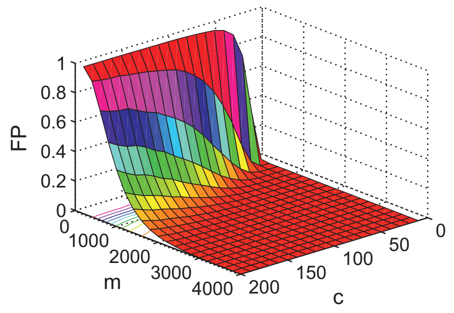
9.2.2. DBF's accuracy

Having determined the default parameters for the DBF, we next test DBF's accuracy in measuring the pairwise NCF values for two-hop users on the social graph. We compute the **Percentage of incorrect intersection** (**PII** for short) as the percentage of correct NCF results by DBF.

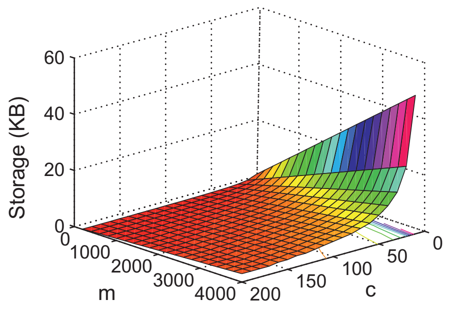
Table 4 shows the results on three social network topologies. We see that more than 99% of all tests are correct, which implies that the Bloom filter based NCF measurements are quite accurate.

9.2.3. DBF's bandwidth requirements

We next compute the bandwidth requirements of exchanging DBFs between pairs of two-hop users. From Fig. 7, we see that the storage sizes of all social network topologies are quite modest, implying that the Bloom filters are quite practical for measuring the common friends.

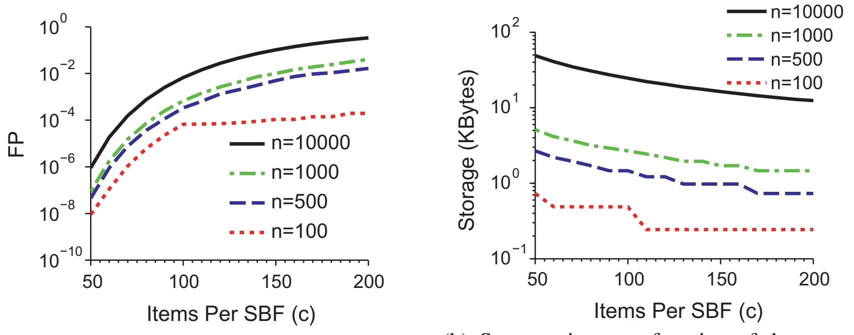


(a) The expected false positive probabilities as a function of the Bloom filter size m and upper bound c.



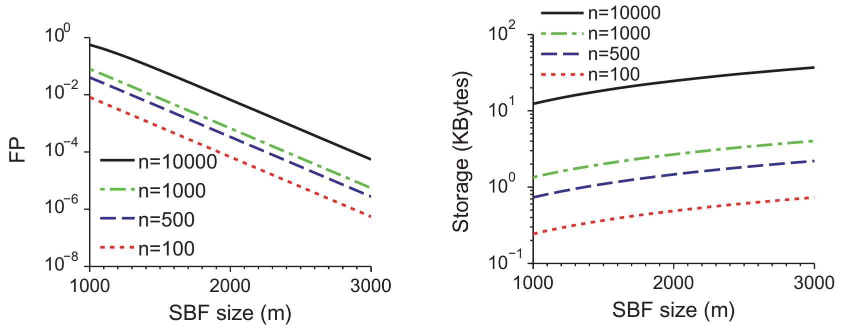
(b) Storage sizes as a function of the Bloom filter size m and upper bound c.

Fig. 5. Variation of the performance of DBF as we vary the upper bound c and the SBF size m. The number of items inserted into the DBF $n = 1000$. The number k of hash functions is selected to optimize the false positive probability of each SBF.



(a) The expected false positive probability as a function of the upper bound c.

(b) Storage size as a function of the upper bound c.



(c) The false positive probabilities as a function of the SBF size m.

(d) Storage size as a function of the SBF size m.

Fig. 6. The expected false positive probability and the storage size of DBF as we vary the upper bound c and the SBF size m, respectively. The default parameters for the experiments are as follows: $n = 1000$, $c = 100$, $m = 2000$. The number k of hash functions is selected to optimize the false positive probability of each SBF.

Table 4
Percentages of incorrect intersection results on three data sets. DBF is configured with the default parameters $m = 2000$, $c = 100$.

Data sets	PII
Facebook	0.000015
YouTube	0.000012
Flickr	0.00842

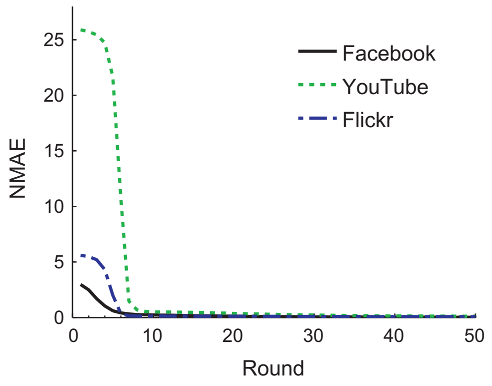


Fig. 8. Convergence of CommonFinder on three data sets.

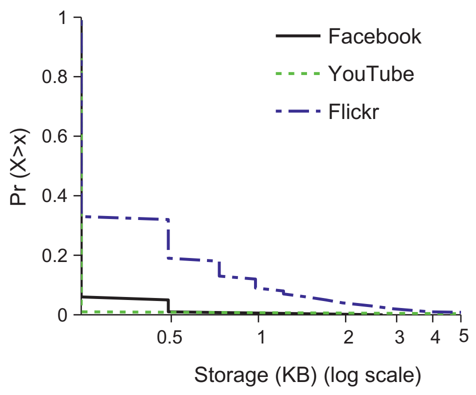


Fig. 7. The storage sizes of DBF for three social network topologies.

9.3. Convergence of the coordinates

We next test the convergence of the NCF estimations in rounds of coordinate updates. Each user selects at most twenty friends for the coordinate update in Algorithm 2 in each round.

Fig. 8 shows the dynamics of estimation accuracy with increasing rounds of coordinate updates. The estimation accuracy is quickly improved as the coordinate update rounds increase. The coordinates have converged to stable positions after ten rounds of coordinate updates. Further coordinate updates do not significantly improve the estimation accuracy.

Furthermore, we also measure the computation time of CommonFinder for each user. The averaged computation time of each user on Facebook, YouTube and Flickr is less than 0.1 s, which indicates that the coordinate-update process is quite efficient.

9.4. Comparison of coordinates based NCF prediction methods

Having confirmed the convergence of the coordinate computation, we next compare CommonFinder's coordinate based NCF prediction method with existing methods. From Fig. 9, we can see that CommonFinder outperforms LandmarkMDS and ProximityEmbed in several times. CommonFinder is also consistently accurate on different social networks. However, we also see that the performance on different data sets varies. This is because different social networks have varying distributions of common friends.

9.4.1. Robustness to incomplete measurements

We next test the robustness of the NCF estimations to incomplete measurements because of offline users. We randomly set a fraction of pairwise NCF values to be zero. Then we compare the NCF estimation results of different methods with the available NCF observations. We set CommonFinder's parameters to be the default values in Table 3.

Fig. 10 plots the dynamics of the accuracy with increasing missing measurements. CommonFinder is stably accurate when the missing fraction is below 0.8, and gracefully degrades the accuracy as the missing fraction increases to 0.9. Finally, CommonFinder is significantly inaccurate after 90% of NCF probing results are missing. On the other hand, the proximity embedding approach degrades the accuracy significantly as the missing fraction increases. We can see that CommonFinder is quite robust to a wide range of missing measurements.

9.4.2. Robustness to incorrect measurements

Since some NCF-probing results could be incorrect due to the false positives of the Bloom filters, we next test

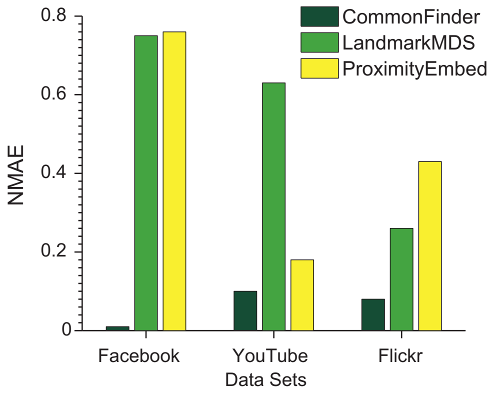


Fig. 9. Comparing the estimation accuracy of different methods on three data sets.

the NCF-estimation accuracy with increasing percentage of incorrect NCF measurements. We select a fraction of NCF probing results uniformly at random and add an error constant e ($e > 0$) to the ground-truth NCF values. We vary the percentage of erroneous NCF results and the error constant e to verify the robustness against erroneous NCF observations. We set CommonFinder's parameters to be the default values in Table 3.

Fig. 11 shows the results. CommonFinder gracefully degrades the accuracy as the error constant e increases. On the other hand, ProximityEmbed significantly degrades the accuracy. Furthermore, CommonFinder and ProximityEmbed are less robust on the Facebook data set than the rest two data sets. This is because the NCF values on the Facebook data set are much lower than those on the other two data sets, which makes the NCF estimation results on the Facebook data set be more sensitive to erroneous NCF probing results on the other two data sets.

9.4.3. Accuracy of top-k prediction

We next compare the accuracy of estimating top-k highest NCF values. For each user i and a constant parameter k, we calculate the multiplicative ratio $\frac{\sum_{y\in\widehat{S}_k}\mathbf{Y}_{iy}}{\sum_{x\in S_k}\mathbf{Y}_{ix}}$, where S_k denotes the set of ground-truth top-k users with the highest NCF values, $\widehat{S}_k$ denotes the set of estimate top-k users with the NCF prediction methods. We can see that the ratio values are at most 1 and larger ratios lead to better prediction results.

From Fig. 12, we see that CommonFinder significantly outperforms both ProximityEmbed and LandmarkMDS methods. The multiplicative ratios for CommonFinder are mostly larger than 0.9 and become closer to one as we increase the number k of recommended users. This is because CommonFinder predicts pairwise NCF values much more accurate than ProximityEmbed and LandmarkMDS. Further, the ProximityEmbed is much more accurate than the LandmarkMDS. This is because ProximityEmbed uses the matrix factorization to adapt the triangle inequality violations of the NCF values, while LandmarkMDS assumes the triangle inequality to hold for all triples, which however does not hold for social data sets.

9.5. Resilience to decentralized nodes

We next test whether CommonFinder is sensitive to erroneous coordinates due to the decentralization of users. We divide the overall set of users in the data sets into two equal halves. Only half users in the data sets are added to the simulator at the first round, some users' friends are in the second half of users and thus may not be added yet. The other half users join the system after 40 rounds. After that cycle, all links between friends in the data sets are present in the simulator. As a result, the erroneous coordinates are injected into the system after 40 rounds.

To quantify the resilience of the coordinate, we calculate the **Coordinate Drift** with the L_1 norm defined as

$$\sum_{l=1}^{2d+L}|x_i(l)-\tilde{x}_i(l)| \qquad (22)$$

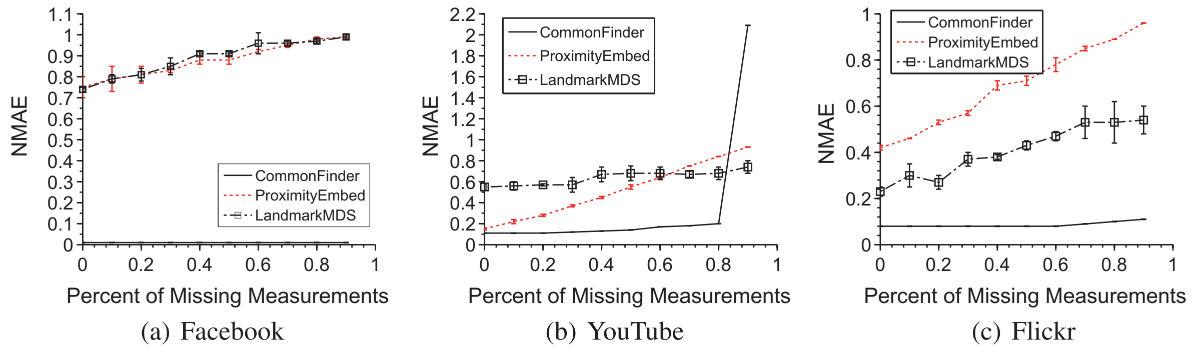


Fig. 10. Accuracy comparison by varying the percentage of missing measurements to neighbors.

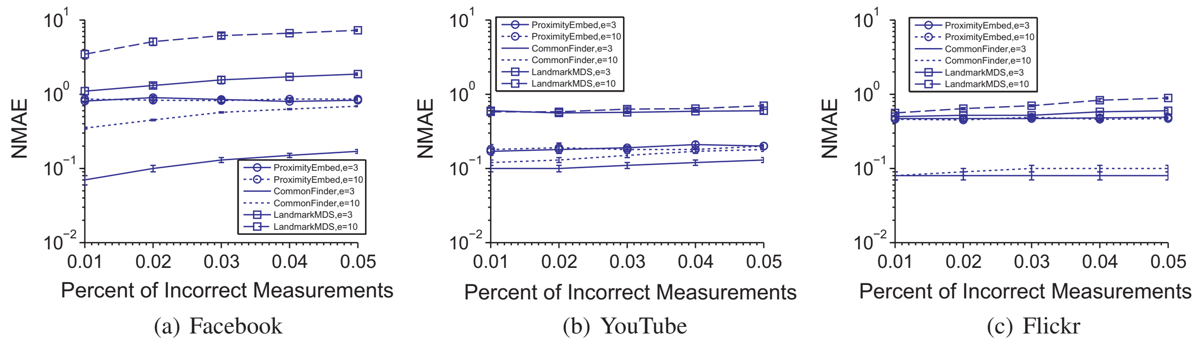


Fig. 11. Accuracy comparison by varying the percent of incorrect measurements.

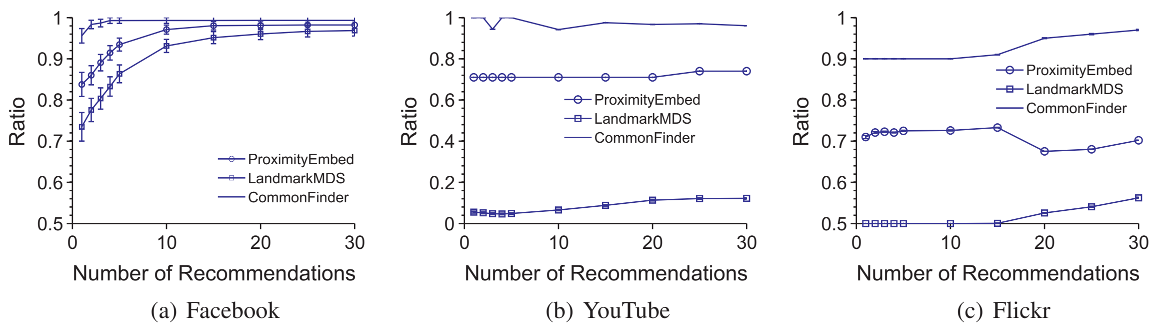


Fig. 12. Accuracy of selecting two-hop users with the top-*k* NCF values.

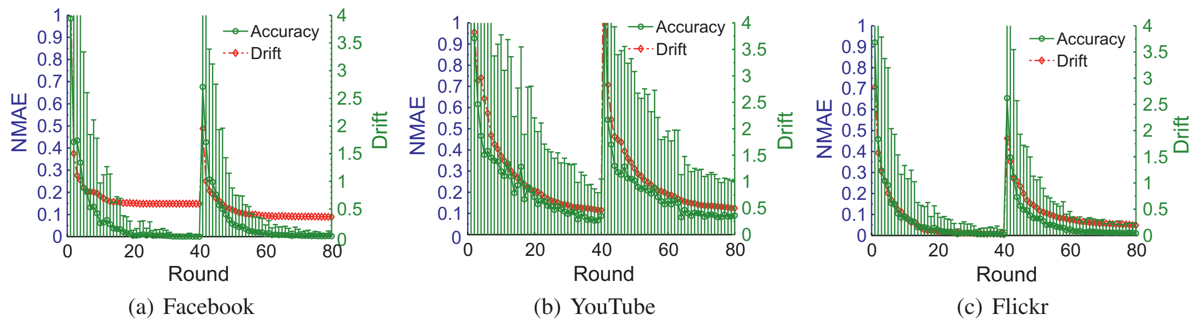


Fig. 13. The effect of high-error coordinates on the rate of convergence.

where x_i and $\tilde{x}_i$ denote the updated coordinate and the last-round coordinate, respectively. We plot the mean coordinate drifts and the mean/standard deviations of the estimation errors in Fig. 13.

The first half users converge to stable coordinates within 20 rounds and keep steady until 40 rounds. Furthermore, the coordinate drifts are reduced to be below 0.1 after the coordinates are stabilized after 20 rounds.

When the other half users join the system after 40 rounds, the overall coordinate errors increase sharply, since the coordinates of newly-joined users are randomly initialized and thus may incur high errors. Furthermore, the coordinate drifts also increase due to the similar reasons. However, the whole set of coordinates converge within the next 20 rounds to stable positions. The newly stabilized coordinates have the similar accuracy as those before 40 rounds. Furthermore, the coordinate drifts also decrease to similar degrees as those before 40 rounds. As a result, the coordinate-update process is quite resilient to erroneous coordinates.

9.6. Parameter sensitivity of optimizing decentralized coordinate

We then test whether the coordinate adjustment process is sensitive to parameter settings. We vary one parameter and set the rest of CommonFinder's parameters to be the default values in Table 3. We report the mean NMAE values and the standard deviations.

9.6.1. Coordinate dimension

Fig. 14(a) plots the accuracy with increasing coordinate dimensions. CommonFinder accurately predicts NCFs even with low coordinate dimensions. The standard deviations are also quite low, which imply that the NCF estimation is quite stable.

9.6.2. Number of neighbors

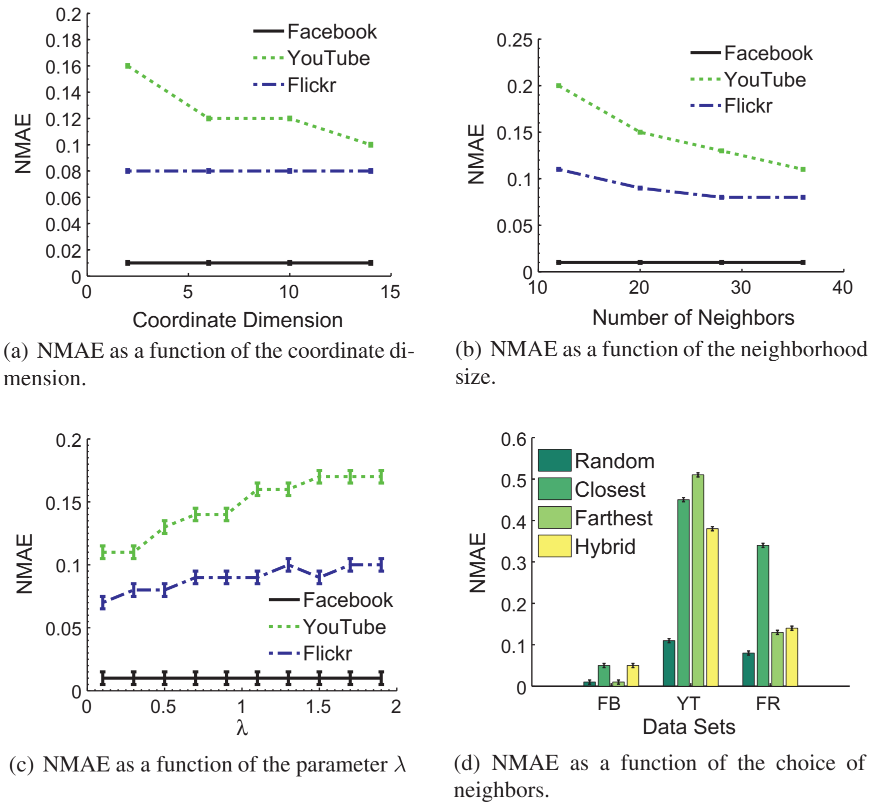
Fig. 14(b) shows the dynamics of accuracy with increasing neighbors. CommonFinder becomes more accurate with increasing neighbors, but keeps stably accurate when the number of neighbors exceeds 20. Therefore, CommonFinder is able to predict accurate NCFs with a modest number of neighbors.

9.6.3. Regularized constant λ

Fig. 14(c) plots the accuracy of CommonFinder by increasing regularized constant λ. The accuracy decreases with increasing λ for the YouTube and Flickr data sets, but keeps stably accurate when we vary λ for the Facebook data set. Therefore, we should choose a smaller λ for updating coordinates.

9.6.4. Choice of neighbors

We next evaluate the choice of neighbors on the accuracy of CommonFinder. We test four popular neighbor-selection rules: (i) **Random**, we choose neighbors uniformly at random from the whole set of friends. (ii) **Closest**, we choose neighbors as friends that have the



(a) NMAE as a function of the coordinate dimension.

(b) NMAE as a function of the neighborhood size.

(c) NMAE as a function of the parameter λ

(d) NMAE as a function of the choice of neighbors.

Fig. 14. NMAE of CommonFinder as we vary its parameters.

lowest NCFs. (iii) **Farthest**, we choose neighbors as friends that have the highest NCFs. (iv) **Hybrid**, we select half neighbors using the Closest based selection and the other half neighbors using the Farthest based selection. Fig. 14(d) shows the results. FB, YT and FR denote the Facebook, YouTube and Flickr data sets, respectively. The Random policy provides the highest accuracy, while other three policies incur much higher errors than the Random policy, since the randomness in neighbors brings much higher diversity in the distribution of NCFs, which makes the optimization process to be robust to bad local minima.

10. Extending the NCF computation to Sybil accounts

10.1. Sybil accounts

Prior work [34,35] broadly group those fake accounts for spamming or performing privacy attacks as Sybil accounts. To launch the attacks, the Sybil accounts generally seek to integrate with the social graph by establishing friend links with normal users. Therefore, they have strong motivations to use fake friend lists to obtain information about other users. Further, these Sybil accounts usually collude together to establish friend links between themselves.

If Sybil accounts do not establish friend links with normal users, we can see that the CommonFinder protocol will not be affected, since each user adjusts its coordinate with friends. Therefore, our analysis assumes that Sybil accounts *have established several friend links with normal users*. We call these normal users as infected ones.

10.2. Learn information about other users

We argue that the CommonFinder protocol does not leak sensitive privacy information to Sybil accounts. First, since we enforce the one-hop trustiness, non-infected users' friend lists will not be disclosed to Sybil accounts. Second, during the coordinate adjustment process, disclosing the Bloom filter and the coordinate of a user do not leak this user's privacy information due to the randomization brought by these two data structures.

10.3. Destabilize coordinate system

The coordinate system, however may be sensitive to Sybil accounts, when they inject fake information into the coordinate computation process. In order to destabilize the coordinates of normal users, the Sybil accounts need to have friend links with some normal users and then create fake Bloom filters and/or generate fake coordinate positions.

10.3.1. Fake Bloom filters

Fake Bloom filters bring incorrect NCF measurements. However, as discussed in the simulation section, CommonFinder is robust to the incorrect NCF measurements because of the inaccuracy brought by the Bloom filters. This is because the decentralized MMMF method used by CommonFinder mitigates the problem of overfitting by regularizing the coordinates with nuclear norms that is robust to noises [36].

10.3.2. Fake coordinates

Fake coordinates generally refer to those that are created by not following the CommonFinder protocol. Suppose that a Sybil user sends a fake coordinate to a friend *B* that is a normal user. As a normal user always has some other normal users as friends, user *B* will move its coordinates with both Sybil and normal neighbors. When there is only a small fraction of Sybil accounts for a normal user, this normal user's coordinate will keep to converge to stable positions since the conjugate gradient optimization tries to balance the coordinates from Sybil and normal users.

As it is difficult to model the Sybil nodes' behaviors. In this paper, we use a simple model to represent uncertain coordinates: To simulate the randomness of coordinate errors in decentralized settings, we randomize the coordinates of a set of online nodes. For each round of the coordinate update process of a node, we randomly choose a fraction p of neighbors to have random coordinates. By varying the percent p of random coordinates, we are able to study impact of the degree of node dynamics on the coordinate's accuracy.

Fig. 15 plots the dynamics of CommonFinder's accuracy as we increase the percent of randomized coordinates. Increasing randomized coordinates only smoothly degrades the accuracy.

11. Related work

Talash [10] calculates the common friends by exchanging two friend lists, which however, does not scale well

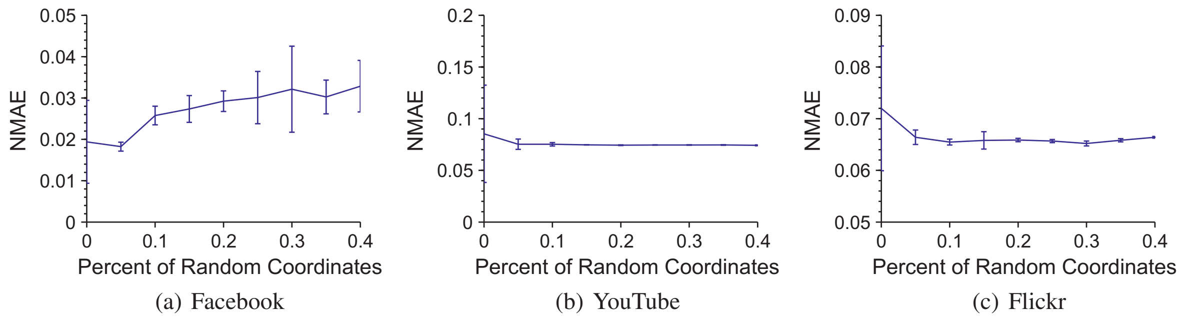


Fig. 15. The accuracy of the NCF prediction process as we increase the percent of random coordinates for each user.

with increasing friends and also discloses sensitive personal information to unknown entities.

The Private Set Intersection (PSI) or private matching method [11–14] computes the intersection of two sets with privacy protection. Generally, each set is represented by a list of coefficients of a polynomial. Estimating the intersection of two sets is implemented as the arithmetic operations on the coefficients of two polynomials. To improve the privacy protection, the homomorphic encryption is enforced over the arithmetic operations in order to hide the coefficients to the other user. Unfortunately, the PSI methods need a large amount of bandwidth cost and computation cost, which renders them to be less practical for bandwidth-limited end hosts.

Song et al. [33] estimate diverse proximity metrics between users through a matrix factorization process that is computed in a centralized manner, which is impractical for distributed online social networks. Dong et al. [37] propose a secure and privacy-preserving dot product protocol for static coordinates computed by [33], in order to estimate the social proximity in mobile social networks.

Finally, network coordinate methods such as Vivaldi [29], LandmarkMDS [32] can be used to predict NCF values. Network coordinates embed hosts into a geometric space and estimate the pairwise distance of two nodes based on the corresponding coordinate distance. However, these decentralized coordinate approaches select neighbors that are used for updating coordinates uniformly at random from the whole set of nodes, which could leak the coordinate positions to non-friend users. On the contrary, CommonFinder selects neighbors from the friend list of each user, which avoids the disclosure of coordinates to unknown users.

12. Conclusion

To estimate common friends in distributed social networks scalably with privacy protection, we propose a distributed common-friends prediction method that combines the privacy-preserving common-friend measurement by using Bloom filters and the distributed common-friend estimation with decentralized coordinates. Simulation results show that our coordinate based method estimates the NCF values scalably and accurately with modest transmission costs. Besides, CommonFinder is also resilient to incomplete NCF measurements and noisy NCF measurements. We plan to implement our method as a plugin for distributed social networks.

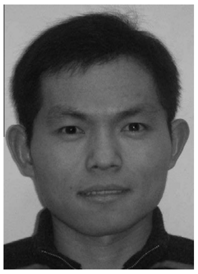
Acknowledgements

This work was supported by the National Grand Fundamental Research 973 Program of China (Grant No. 2011CB302601), the National Natural Science Foundation of China (Grant No. 61379052), the National High Technology Research and Development 863 Program of China (Grant No. 2013AA01A213), the Natural Science Foundation for Distinguished Young Scholars of Hunan Province (Grant No. S2010J5050), Specialized Research Fund for the Doctoral Program of Higher Education (Grant No. 20124307110015).

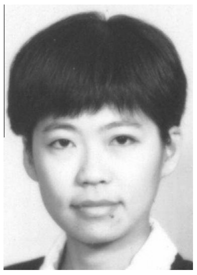
References

[1] L.A. Cutillo, R. Molva, T. Strufe, Safebook: Feasibility of Transitive Cooperation for Privacy on a Decentralized Social Network, in: Proc. of IEEE WOWMOM'09, pp. 1–6.
[2] L.M. Aiello, G. Ruffo, LotusNet: tunable privacy for distributed online social network services, Comput. Commun. 35 (2012) 75–88.
[3] T. Xu, Y. Chen, J. Zhao, X. Fu, Cuckoo: Towards Decentralized, Socio-aware Online Microblogging Services and Data Measurements, in: Proc. of HotPlanet '10, pp. 4:1–4:6.
[4] Diaspora, Diaspora Alpha, 2011. <https://joindiaspora.com/>.
[5] S. Buchegger, D. Schiöberg, L.-H. Vu, A. Datta, PeerSoN: P2P Social Networking: Early Experiences and Insights, in: Proc. of SNS '09, pp. 46–52.
[6] A. Shakimov, H. Lim, R. Cáceres, L.P. Cox, K.A. Li, D. Liu, A. Varshavsky, Vis-à-Vis: Privacy-preserving Online Social Networking via Virtual Individual Servers, in: Proc. of COMSNETS'11, pp. 1–10.
[7] D. Liben-Nowell, J. Kleinberg, The Link Prediction Problem for Social Networks, in: Proc. of CIKM '03, pp. 556–559.
[8] A. Mislove, B. Viswanath, K.P. Gummadi, P. Druschel, You are who you know: inferring user profiles in online social networks, in: Proceedings of the third ACM international conference on Web search and data mining, WSDM '10, pp. 251–260.
[9] L. Bilge, T. Strufe, D. Balzarotti, E. Kirda, All your contacts are belong to us: automated identity theft attacks on social networks, in: Proceedings of the 18th international conference on World wide web, WWW '09, pp. 551–560.
[10] R. Dhekane, B. Vibber, Talash: Friend Finding In Federated Social Networks, in: Proc. of LDOW2011.
[11] M.J. Freedman, K. Nissim, B. Pinkas, Efficient private matching and set intersection, in: Proc. of EUROCRYPT 2004, pp. 1–19.
[12] E.D. Cristofaro, P. Gasti, G. Tsudik, Fast and private computation of set intersection cardinality, Cryptology ePrint Archive, Report 2011/141, 2011. <http://eprint.iacr.org/>.
[13] L. Kissner, D. Song, Privacy-Preserving Set Operations, in: Proc. of CRYPTO 2005, pp. 241–257.
[14] D. Dachman-Soled, T. Malkin, M. Raykova, M. Yung, Efficient robust private set intersection, in: Applied Cryptography and Network Security, vol. 5536, 2009, pp. 125–142.
[15] C. Dwork, Differential privacy: a survey of results, in: TAMC, pp. 1–19.
[16] F. McSherry, K. Talwar, Mechanism design via differential privacy, in: Proc. of FOCS '07, pp. 94–103.
[17] Y. Fu, Y. Wang, On the feasibility of common-friend measurements for the distributed online social networks, in: The 7th International ICST Conference on Communications and networking in China (CHINACOM), 2012, pp. 24–29.
[18] Y. Fu, Y.Wang, BCE: A privacy-preserving common-friend estimation method for distributed online social networks without cryptography, in: The 7th International ICST conference on Communications and Networking in China (CHINACOM), 2012, pp. 212–217.
[19] J.D.M. Rennie, N. Srebro, Fast maximum margin matrix factorization for collaborative prediction, in: Proc. of ICML'05, pp. 713–719.
[20] G.H. Golub, C.F. Van Loan, Matrix Computations, third ed., Johns Hopkins University Press, Baltimore, MD, USA, 1996.
[21] J.R. Shewchuk, An Introduction to the Conjugate Gradient Method Without the Agonizing Pain, Technical Report, Pittsburgh, PA, USA, 1994. <http://www.ncstrl.org:8900/ncstrl/servlet/search?formname=detail&id=oai3Ancstrlh3Acmucs3ACMU2F2FCS-94-125>.
[22] O. Goldreich, Foundations of Cryptography, Basic Applications, 2, Cambridge University Press, 2004.
[23] J. Becker, H. Chen, Measuring privacy risk in online social networks, in: Proc. of W2SP 2009: Web 2.0 Security and Privacy.
[24] B. Viswanath, A. Mislove, M. Cha, K.P. Gummadi, On the evolution of user interaction in Facebook, in: Proc. of WOSN '09, pp. 37–42.
[25] M. Cha, A. Mislove, K.P. Gummadi, A Measurement-driven analysis of information propagation in the Flickr Social Network, in: Proc. of WWW '09, pp. 721–730.
[26] A. Mislove, M. Marcon, K.P. Gummadi, P. Druschel, B. Bhattacharjee, Measurement and analysis of online social networks, in: Proc. of IMC'07.
[27] X. Su, T.M. Khoshgoftaar, A survey of collaborative filtering techniques, Adv. Artif. Intell. (2009).

[28] D. Guo, J. Wu, H. Chen, Y. Yuan, X. Luo, The dynamic bloom filters, IEEE Trans. Knowl. Data Eng. 22 (2010) 120–133.
[29] F. Dabek, R. Cox, F. Kaashoek, R. Morris, Vivaldi: a Decentralized Network Coordinate System, in: Proc. of ACM SIGCOMM 2004, pp. 15–26.
[30] J. Ledlie, P. Gardner, M.I. Seltzer, Network Coordinates in the Wild, in: Proc. of 4th USENIX Conference on Networked Systems Design and Implementation, USENIX Association, Cambridge, MA, 2007. pp. 22–22.
[31] Y. Liao, W. Du, P. Geurts, G. Leduc, DMFSGD: A Decentralized Matrix Factorization Algorithm for Network Distance Prediction, CoRR abs/1201.1174, 2012.
[32] B. Eriksson, P. Barford, R.D. Nowak, Estimating hop distance between arbitrary host pairs, in: Proc. of IEEE INFOCOM'09, pp. 801–809.
[33] H.H. Song, T.W. Cho, V. Dave, Y. Zhang, L. Qiu, Scalable proximity estimation and link prediction in online social networks, in: Proc. of IMC '09, pp. 322–335.
[34] G. Danezis, P. Mittal, Sybilinfer: Detecting sybil nodes using social networks, in: NDSS.
[35] Z. Yang, C. Wilson, X. Wang, T. Gao, B.Y. Zhao, Y. Dai, Uncovering social network sybils in the wild, in: Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference, IMC '11, pp. 259–268.
[36] E.J. Candès, Y. Plan, Matrix completion with noise, Proc. IEEE 98 (2010) 925–936.
[37] W. Dong, V. Dave, L. Qiu, Y. Zhang, Secure friend discovery in mobile social networks, in: Proc. of IEEE INFOCOM'11.



Yongquan Fu received the B.S. degree in computer science and technology from the School of Computer of Shandong University, China, in 2005, and received the M.S. in Computer Science and technology in the School of Com- puter Science of National University of Defense Technology, China, in 2008. He is currently an assistant professor in the School of Computer Science of National University of Defense Technology. He is a member of CCF and ACM. His current research interests lie in the areas of cloud management, distributed online social networks and distributed algorithms.



Yijie Wang received the PhD degree from the National University of Defense Technology, China in 1998. She was a recipient of the National Excellent Doctoral Dissertation (2001), a recipient of Fok Ying Tong Education Foundation Award for Young Teachers (2006) and a recipient of the Natural Science Foundation for Distinguished Young Scholars of Hunan Province (2010). Now she is a Professor in the National Key Laboratory for Parallel and Distributed Processing, National University of Defense Technology. Her research interests include network computing, massive data processing, parallel and distributed processing.

Wei Peng received the M.Sc. and the Ph.D degrees in Computer Science from the National University of Defense Technology, China, in 1997 and 2000 respectively. From 2001 to 2002, he is a research assistant at the School of Computer from the National University of Defense Technology, China, then a research fellow from 2003. He was a visiting scholar of University of Essex, UK, from May 2007 to April 2008. His major interests are internet routing, network science, and mobile wireless networks.