Kou Y, Li D, Shen DR *et al.* Incremental user identification across social networks based on user-guider similarity index. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 37(5): 1086–1104 Sept. 2022. DOI 10.1007/s11390-022-2430-0

Incremental User Identification Across Social Networks Based on User-Guider Similarity Index

Yue Kou[1] (寇 月), *Member*, *CCF*, Dong Li[2,*] (李 冬), *Member*, *CCF*
De-Rong Shen[1] (申德荣), *Senior Member*, *CCF*, Tie-Zheng Nie[1] (聂铁铮), *Senior Member*, *CCF*, and
Ge Yu[1] (于 戈), *Senior Member*, *CCF*, *Member*, *ACM*, *IEEE*

[1] *School of Computer Science and Engineering, Northeastern University, Shenyang 110004, China*
[2] *School of Information, Liaoning University, Shenyang 110036, China*

E-mail: kouyue@cse.neu.edu.cn; dongli@lnu.edu.cn
{shenderong, nietiezheng, yuge}@cse.neu.edu.cn

Received April 15, 2022; accepted September 15, 2022.

Abstract Identifying accounts across different online social networks that belong to the same user has attracted extensive attentions. However, existing techniques rely on given user seeds and ignore the dynamic changes of online social networks, which fails to generate high quality identification results. In order to solve this problem, we propose an incremental user identification method based on user-guider similarity index (called CURIOUS), which efficiently identifies users and well captures the changes of user features over time. Specifically, we first construct a novel user-guider similarity index (called USI) to speed up the matching between users. Second we propose a two-phase user identification strategy consisting of USI-based bidirectional user matching and seed-based user matching, which is effective even for incomplete networks. Finally, we propose incremental maintenance for both USI and the identification results, which dynamically captures the instant states of social networks. We conduct experimental studies based on three real-world social networks. The experiments demonstrate the effectiveness and the efficiency of our proposed method in comparison with traditional methods. Compared with the traditional methods, our method improves precision, recall and rank score by an average of 0.19, 0.16 and 0.09 respectively, and reduces the time cost by an average of 81%.

Keywords user identification, social network, user-guider similarity index, incremental maintenance

1 Introduction

Online social networks have become more and more popular in recent years. It is very common that one real-world person gets involved in different social networks concurrently to satisfy his/her different requests. For instance, by using different public social networks (such as Twitter①, Facebook② and Foursquare③), a user can access the latest news, share photos and make a post. Besides public social networks, he/she can also share files by using multiple private social networks (such as enterprise networks and department networks). User identification, which is to identify accounts across different online social networks that belong to the same user, has been extensively studied. Integrating data across various social platforms down to the granularity

Regular Paper
Special Section on Scalable Data Science
This work was supported by the National Natural Science Foundation of China under Grant Nos. 62072084, 62172082 and 62072086, the Science Research Foundation of Liaoning Province of China under Grant No. LJKZ0094, the Natural Science Foundation of Liaoning Province of China under Grant No. 2022-MS-171, the Science and Technology Plan Major Project of Liaoning Province of China under Grant No. 2022JH1/10400009, and the Fundamental Research Funds for the Central Universities of China under Grant No. N2116008.

*Corresponding Author

①www.twitter.com, Sept. 2022.
②www.facebook.com, Sept. 2022.
③www.foursquare.com, Sept. 2022.

of individuals is an essential task in today's social-data-enabled business intelligence [1]. User-centric data fusion from various sources is very important for many applications such as social recommendation, community detection, public opinion analysis, and so on. For instance, by aggregating sufficient user-centric behaviour data from multiple networks, recommendation systems can obtain more complete profiles and richer behavioral data, which enables an all-sided analysis on user preferences.

We study the problem of user identification across social networks. Given multiple social networks, we aim to efficiently identify accounts that belong to the same user. For user identification across social networks, three key issues need to be addressed. First, we need to design an efficient index scheme for searching the similar users with respect to an incoming user. According to the statistics from Hootsuite [2], there are now 4.62 billion social media users, with 424 million new users added in the past year. The number of social media users is now equal to more than 58% of the world's population. Obviously, all-pairwise-comparison for this huge number of users is infeasible. Second, we need to propose a robust user identification strategy that effectively integrates data across various sources, even for incomplete social networks. The incompleteness mainly lies in the lack of user attributes and seeds. In practice, it is very difficult to obtain users' private information in many real-world applications because of privacy concerns, resulting in the lack of attributes. Also it is laborious to label enough prior alignment manually, leading to the lack of user seeds. In scenarios where social data is sparse, incomplete, or hard to obtain, the process of user identification needs to be still effective. Third, we need to propose an incremental maintenance strategy for both the constructed index and the current identification results, which can effectively capture the instant states of social networks. As users' attributes and their interactions with others may change over time, failing to capture these dynamic states may lead to low quality identification.

According to the features used in the process of identification, existing user identification approaches can be classified into three categories, attributes-based approaches [1, 3–10], structure-based approaches [11–26] and hybrid approaches [27–35]. Most approaches only rely on one type of features (either user attributes or topological structures), which does not reflect the complete features of users. To improve the accuracy of identification, hybrid approaches have been proposed to identify users by combining user attributes with topological structures. However, existing hybrid approaches ignore the dynamic changes of social networks (including the changes of users themselves and the changes of interaction among users), which fails to generate high quality identification. Meanwhile, these methods are not effective for incomplete networks.

In this paper, we propose an incremental user identification method based on user-guider similarity index (called CURIOUS), which identifies users more efficiently and accurately. Specifically, a novel user-guider similarity index (called USI) is built to speed up searching the similar users with respect to an incoming user. Then, a robust two-phase user identification strategy (consisting of USI-based bidirectional user matching and seed-based user matching) is proposed, which is effective even for incomplete networks. Finally, we propose some incremental maintenance strategies, which effectively capture the instant states of users and the interactions among users. Our contributions are as follows.

- We construct USI, an efficient index scheme, to improve the identification efficiency, which is guaranteed by a novel guider-based candidate pruning. It has better generality, which does not depend on particular patterns or specific hash functions.
- We propose a two-phase user identification strategy. In the first phase, the similar users with respect to an incoming user are searched by bidirectionally matching in USI. In the second phase, seeds are exploited to further expand the matching results generated in the previous phase, which better solves the problems caused by the incompleteness of networks.
- We propose incremental maintenance for USI and the identification results respectively. Some incremental maintenance strategies are proposed for the changes including the insertion, deletion or updating of user nodes, which enables the identification results to well reflect the networks' dynamic states.
- Extensive experimental studies based upon three real-world networks are conducted. The experiments demonstrate the effectiveness and efficiency of our proposed method.

The rest of this paper is organized as follows. Section 2 reviews related work. Section 3 formulates the problem and gives an overview of our CURIOUS method. Section 4 and Section 5 propose the USI construction algorithm and the two-phase user identification strategy respectively. Section 6 proposes the incremental maintenance strategies. Section 7 shows the

experimental results and Section 8 makes conclusions.

2 Related Work

Various approaches for user identification across social networks have been studied over the years. First, we briefly review the techniques for them, and then analyze how our work differs from them.

2.1 Attribute-Based User Identification

Attributes-based user identification methods collect user attributes about user profiles, user-generated content, behaviors or friend networks, and then represent them in vectors, of which each dimension corresponds to an attribute field. These methods can be divided into profile-based methods and content-based methods.

The basic idea of profile-based methods is to use the user profile attributes to measure the similarity of user accounts, and then to identify users based on the similarity results. For example, in [3, 4], user profiles such as user-name or description, are collected as attributes for user identification. In [5] username and display name are used to solve the problem of user identification across social networks. In [6] a reinforcement learning based framework is designed to augment the attributes following an exploration-exploitation strategy. In [1] the user identities of multiple networks are projected into a common latent user space and the linked users are identified by comparing their profiles in the latent user space.

Content-based methods attempt to identify users based on the time and locations that users post content, as well as the writing style of the content. For example, in [7] personal identifiable information from user-generated content is collected and used for identification. In [8] users' behavior patterns, such as user language and writing styles, are analyzed for building user identification models. In [9, 10] the representative features from users' geo-locations or trajectories are extracted, compared to decide whether to link two trajectories as the same user.

2.2 Structure-Based User Identification

Structure-based user identification methods utilize the structures of social networks, because for most users their online friends usually constitute a similar group of people on different social graphs. These structure-based methods leverage user arrange structures or seeds (i.e., priori distinguished users) to identify more users, and can be divided into three categories: propagation-based methods, network alignment methods, and representation-based methods.

The basic idea of propagation-based methods is to start from seeds and extend to their unidentified neighbors to identify more nodes. For example, in [11] a divide-and-conquer approach is proposed, which partitions the networks into communities and performs a two-stage mapping (first at the community level, and then for the entire network). In [12] an algorithm called FRUI is proposed, which employs a unified friend relationship from a heterogeneous network and chooses candidate matching pairs from currently known identical users.

Network alignment [13–16] is to fuse multiple graph data sources by finding the corresponding nodes across them. For example, in [13] an algorithm called IsoRank is proposed to compute network alignments, which simultaneously uses the network data and the sequence similarity data. In [14] a message passing algorithm called NetAlignMP is proposed to compute approximate solutions to the sparse network alignment problems. In [15] a method for multimodal network alignment is proposed, which computes approximate maximum weight matchings of low-rank matrices to produce an alignment. In [16] a solution for the maximum weight bipartite matching problem on the low-rank matrix is proposed to produce the matching between the graphs.

Via node representation learning, we can get the node representation vector, which can reflect both the local and the global structural features of user nodes. Representation-based methods (e.g., [17–26]) first represent users as vectors and then compute the similarity between user vectors to determine whether two users are the same person. In [17] an unsupervised scheme FRUI-P is proposed. First it extracts the friend relationships into feature vectors, and then calculates the similarities of all the candidate identical users between two networks. In [18] a dual attention matching network is proposed, which models both intra-graph and cross-graph information smartly. In [19] an adversarial-enhanced hybrid graph network is proposed to learn hybrid user representation. In [20] a reinforcement learning model is proposed to optimize the linkage strategy, which utilizes both the social network structure and the history matched identities. In [26] an unsupervised embedding method is proposed by utilizing the network structural information.

2.3 Hybrid User Identification

Hybrid user identification methods comprehensively consider multiple types of features including user attributes and structural features to improve the accuracy of user identification.

First, some weighted combination methods for features are proposed. For example, in [27] not only the structural features such as out-degree, in-degree and n-hop neighbors, but also the attributes attached with user nodes are considered to compute the structural similarity and the attributes similarity respectively. And then the weights are assigned to them to calculate the final similarity between users. In [28] a framework called REGAL is proposed, which quantifies the overall user similarity via the weighted combination. In [29] an iterative graph aligner called gsaNA is proposed, which synthetically considers the similarities of user attributes, relationships, node out-degree and node in-degree to analyze the global structure of the graphs. In [30] three levels of attribute features are combined with the structural features by a concatenate operation to obtain the final representation of the social network.

Second, some methods combine different features based on collaborative training. For example, in [31] two models based on attribute features and structural features are trained respectively, and then the top-k user pairs of each model are added to the training set iteratively. In [32] a collaborative training framework is proposed to train the representation of various features separately and to enhance each other with their results until the model is convergent. In [33] a joint attribute-preserving embedding model for cross-lingual entity alignment is proposed, which jointly embeds the structures and further refines the embeddings by leveraging attribute correlations. In [34] an ontology-guided entity alignment method is proposed, which jointly embeds both the networks and their ontologies.

In addition, some work also focuses on how to speed up identification. The common way is to reduce the number of candidate users to be identified as much as possible, that is, to avoid the similarity calculation of a user with a large number of dissimilar users. Some work organizes similar users together using buckets or space partitions, in order to limit the candidates within a smaller space. For example, in [29] user nodes are partitioned into buckets on a 2D plane by using their distances to the vantage anchor pairs. Only the pairwise similarity among the nodes of the two graphs that fall into the same bucket needs to be computed. In [35], an efficient hash-based framework for the network alignment called HashAlign is proposed, which leverages structural properties and node attributes simultaneously.

2.4 Differences from Existing Work

The differences between our work and existing work are as follows.

On the one hand, attributes-based methods and structure-based methods assume that they can collect rich user attributes and sufficient seeds respectively. However, in practice we can only obtain the limited information, leading to limited or incomplete features. Unfortunately, as most methods rely on particular patterns or seeds, they face serious challenges in identifying users across social networks if required data patterns or seeds are not available. In this paper, we construct a more general index scheme, which does not depend on particular patterns or hash functions. In addition, we propose a two-phase user identification strategy, which takes into account both user attributes and structure features to supply more evidence for user identification.

On the other hand, although hybrid user identification methods improve the accuracy of identification by considering various user features, existing methods are proposed for static networks, which ignores the dynamic interactions among users, resulting in generating some outdated results. In this paper, we propose some incremental maintenance strategies for both the constructed index and the identification results, which can well reflect the networks' dynamic states.

3 Solution Overview

We propose an incremental user identification method based on USI (called CURIOUS). The framework of CURIOUS is shown in Fig.1.

Definition 1 (User Identification Across Social Networks). *Given two social networks (G_A and G_B) and a few prematched user pairs $\mathcal{M} = \{(u^A, u^B) \mid u^A \in G_A \wedge u^B \in G_B\}$, the problem of user identification across social networks involves locating the other hidden matched identity pairs $\chi = \{(u^A, u^B) \mid u^A \in G_A \wedge u^B \in G_B \wedge (u^A, u^B) \notin \mathcal{M}\}$, where u^A and u^B belong to the same natural person.*

CURIOUS consists of three major components, USI construction, two-phase user identification, and incremental maintenance. First, USI is constructed according to the extracted node features from one network

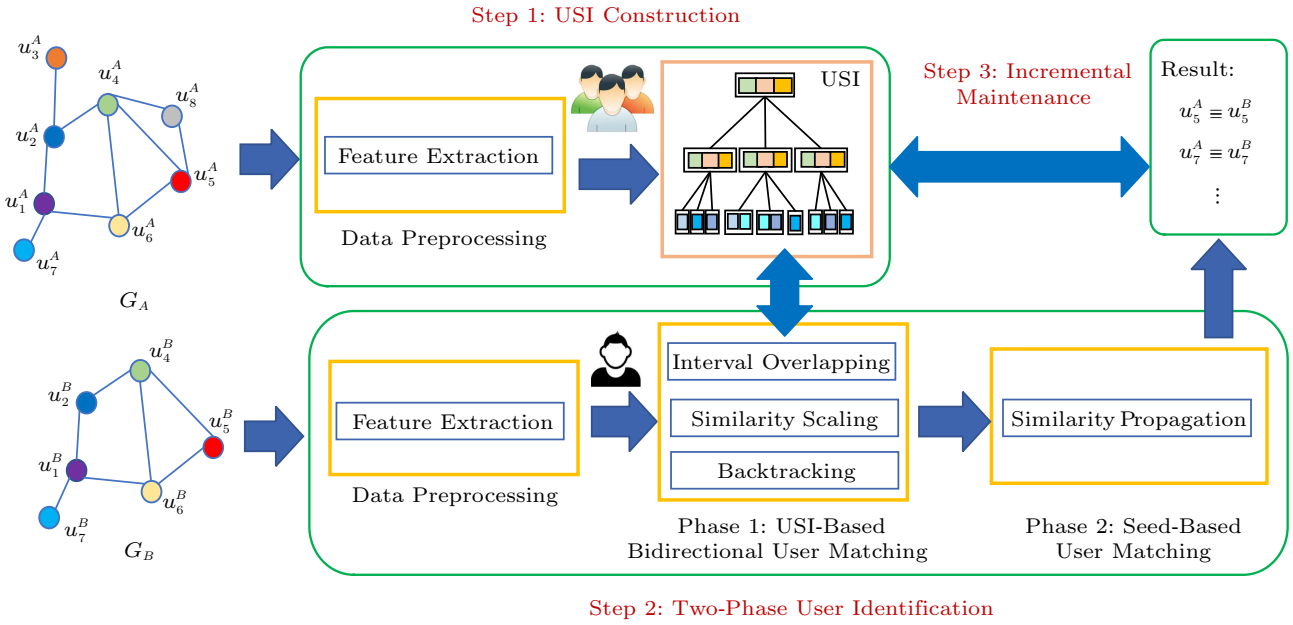


Fig.1. Framework of our solution.

(e.g., G_A), which is made up of bucket-level index nodes and guider-level index nodes. Second, two-phase user identification is performed. In the first phase, for each user to be identified in G_B, users in G_A which are similar to it, are searched by bidirectionally matching in USI. In the second phase, seeds are used to further expand the matching results generated in the previous phase. Finally, incremental maintenance is performed for the changes of both USI and the current identification results, including the insertion, deletion or updating of user nodes. We will detail these components in Sections 4–6. The notations used in this paper are listed in Table 1.

Table 1. Notations

Symbol	Definition and Description
G_A, G_B	Two social networks
x	An index node in USI
q^u	Social circle feature of user u
Q^u	Social chain feature of user u
f_i^u	The i-th feature of user u
b	Number of branches of each index node
ϵ	Threshold for splitting
p_{skew}	Similarity skew ratio
u^A, u^B	A user from G_A, and a user from G_B
$S_f(u^A, u^B)$	Feature-based similarity between u^A and u^B
$S_p(u^A, u^B)$	Propagation-based similarity between u^A and u^B
ΔS_f	Similarity scaling rate
θ_f, θ_p	Threshold for S_f, and threshold for S_p
N_i^u	User u's neighbor set with i-radius
d^u	User u's degree

4 User-Guider Similarity Index

Starting from a user node in one social network (e.g., G_A), we will construct a user-guider similarity index (i.e., USI). It is used to quickly locate the similar users in G_A with respect to an incoming user.

4.1 USI Data Structure

Given a social network G_A, USI is a tree that describes the similarities between users in G_A. It is made up of guider-level index nodes and bucket-level index nodes. The data structure of USI is as follows. 1) Each guider-level index node in USI consists of a set of guiders. Each guider corresponds to a user node in G_A, which is used as a reference for searching. 2) Each bucket-level index node includes a group of users in G_A. For two users (suppose one is from an index node x, and the other is from x's child in USI), the similarity between them should meet a certain threshold. 3) The relationships between two adjacent index nodes in USI are one-to-many. An edge between two index nodes is built if and only if their similarity meets a certain interval. For each index node, we use a node set NS and an edge set ES to store its guiders/users and the similarity relationships with its child node, respectively.

An example of USI is shown in Fig. 2. There are two guider-level index nodes and two bucket-level index nodes, each of which has a guider/user node set and an edge set. For bucket-level index nodes, they

make up all the leaves of USI and their ESs are empty. Let us suppose u_1 and u_2 are selected as guiders to form a guider-level index node (denoted as x_1) in USI. Each unassigned user in G_A (denoted as u) will be assigned to USI according to its similarity with u_1 (denoted as $S(u,\ u_1)$) and with u_2 (denoted as $S(u,\ u_2)$). Here we use the average value of $S(u,\ u_1)$ and $S(u,\ u_2)$ to measure the similarity between u and x_1. In Fig.2 the number of branches (denoted as b) of each index node is 2, and thus each user will be assigned to either of them, depending on whether their similarity belongs to the interval of [0, 0.5) or [0.5, 1]. Since the similarity between u_3 (or u_4, u_5) and x_1 is less than 0.5, it is assigned to x_2. As the similarity between u_6 (or u_7) and x_1 is not less than 0.5, it is assigned to x_3. Therefore the node sets of x_2 and x_3 are $\{u_3, u_4, u_5\}$ and $\{u_6, u_7\}$, respectively. And there are two index edges connecting x_1 to x_2, and x_1 to x_3 respectively.

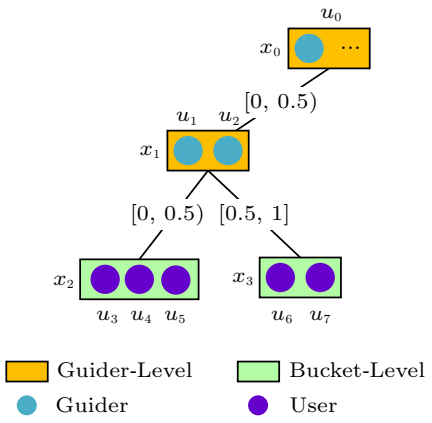


Fig.2. Example of index nodes in USI.

4.2 USI Construction

For USI construction, it consists of two main steps. First, we preprocess the data to extract user attributes and structural features from multiple perspectives. Second, for each user node in G_A, it is assigned to USI according to its extracted features. Initially a bucket-level index node is constructed to store users. And then it will be split into a guider-level index node and several bucket-level index nodes, once it consists of enough user nodes.

4.2.1 *Data Preprocessing*

Users can be described by attributes such as user name, age and occupation. Through data preprocessing, the users' attributes will be extracted. In the paper, we consider three types of attributes, i.e., numeric, string and label (category). More specially, for numeric attributes, we use the Euclidean distance and cosine similarity to calculate the similarity between them. For continuous string attributes, we use the edit distance to calculate the similarity between them. For token string attributes, we use Jaccard similarity to measure their similarity. For label attributes, we first convert them to numeric attributes or string attributes, and then use the corresponding similarity measures to calculate. In addition, the similarity measures between label attributes can be divided into strict measures and relaxed measures. The former refers to that two labels are considered similar only if they are identical (the similarity is 1); otherwise they are considered dissimilar (the similarity is 0). For example, strict measures are often adopted for gender labels. Relaxed measures refer to that two labels may be considered similar even if they are not identical. For example, the relaxed measures can be adopted for occupation labels.

Besides user attributes, we also consider users' structural features. In addition to common structural features such as out-degree, in-degree and the number of neighbors, we define two new structural features: social circle and social chain. Fig.3 shows an example of extracting a user u's social circle and social chain by traversing a social network.

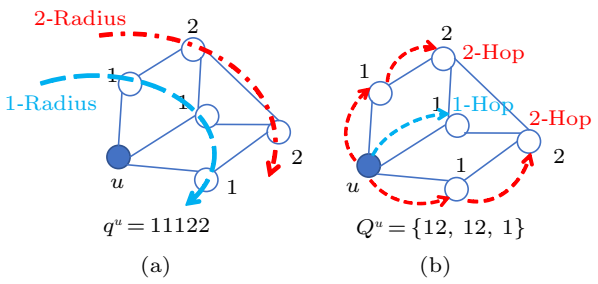


Fig.3. Example of extracting u's structural features. (a) Social circle feature. (b) Social chain feature.

On the one hand, a sequence $q^u = 11122$ representing u's social circle feature can be generated by using the breadth-first traversal. With u as the center node, there are three users with 1-radius (each is represented as 1 in q^u) and two users with 2-radius (each is represented as 2 in q^u). Each neighbor with 1-radius represents a direct friend of the current user, and each neighbor with 2-radius or larger radius represents an indirect friend. As the radius increases, the influence of friends on the current user decreases. Given the social circles of u and u' (denoted as q^u and $q^{u'}$ respectively), we can calculate their similarity in social circle (as in (1)). Here $d_{\max}$ is the maximum radius in the sequences, and

q_i^u (or $q_i^{u'}$) is the number of u's (or u''s) i-radius neighbors.

$$S_{\text{cir}}(q^u, q^{u'}) = \frac{1}{d_{\max}} \sum_{i=1}^{d_{\max}} \frac{1}{1 + |q_i^u - q_i^{u'}|/i}. \quad (1)$$

On the other hand, by adopting the depth-first traversal, a sequence set $Q^u = \{12, 12, 1\}$ representing u's social chain feature can be obtained, from which it can be known that there are two 2-hop social chains and one 1-hop social chain. Each chain is a sequence, the first digit in the sequence represents a 1-hop friend of u, and the second digit represents a 2-hop friend of u, who is known by its previous 1-hop user, and so on. However, the depth-first traversal is random and may lead to inconsistent social chains obtained by two traversals. This problem is caused by the fact that each user has multiple friends, resulting in multiple possibilities when choosing a path. In order to solve this problem, the neighbor node with the smallest degree is preferentially selected for traversal. To reduce computational complexity, if there are multiple neighbors with the same degree, we randomly select one of them for traversal. Given the social chains of u and u' (denoted as Q^u and $Q^{u'}$ respectively), we can calculate their similarity (as in (2) and (3)). Here Q_i^u (or $Q_j^{u'}$) is the i-th (or j-th) sequence in Q^u (or $Q^{u'}$).

$$S_{\text{cha}}(Q^u, Q^{u'}) = \frac{\min(|Q^u|, |Q^{u'}|)}{\max(|Q^u|, |Q^{u'}|)} \times h. \quad (2)$$

$$h = \sum_{Q_i^u \in Q^u, Q_j^{u'} \in Q^{u'}} \frac{\text{Overlap}(Q_i^u, Q_j^{u'})}{\min(|Q_i^u|, |Q_j^{u'}|)}. \quad (3)$$

Social networks are often incomplete, that is, lacking in some features. Since it is impossible to determine whether the null features are similar or dissimilar, it is not feasible to calculate the similarity between users by weighted accumulation on all features. To this end, we use a switch function to dynamically remove these null features when computing the similarity between users (as in (4)). Here f_i^u (or $f_i^{u'}$) is the i-th feature of u (or u'). And $S(f_i^u, f_i^{u'})$ is the similarity between f_i^u and $f_i^{u'}$. $SW(f_i^u, f_i^{u'})$ is a switch function, which is equal to 1 if and only if both f_i^u and $f_i^{u'}$ are not null; otherwise it is equal to 0.

$$S_f(u, u') = \frac{\sum_i w_i \times SW(f_i^u, f_i^{u'}) \times S(f_i^u, f_i^{u'})}{\sum_i w_i \times SW(f_i^u, f_i^{u'})}. \quad (4)$$

4.2.2 Accumulation and Splitting

Based on the features extracted above, we can construct a USI by performing accumulation and splitting iteratively. In the accumulation phase, a bucket-level index node is initialized to load user nodes. When the number of user nodes reaches the split threshold (denoted as ϵ), the splitting phase will be enabled. In the splitting phase, a guider-level index node and several bucket-level index nodes are created to replace the original index node. The user nodes in the original index node are re-assigned to a newly-created index node. Then the remaining user nodes in the social network continue to be assigned in this way until all the user nodes have been assigned to the USI.

The key for building a USI is how to insert a user, i.e., to locate an index node to load it. The target index node should contain the users similar to the inserted user. We propose a user insertion algorithm (Algorithm 1). Given the current USI with its *root*, the user u to be inserted and the split threshold ϵ, Algorithm 1 is performed according to the following three cases.

Algorithm 1. insertUser(*root*, u, ϵ)

Input: *root*, u, ϵ
Output: root of the new USI *root*

```
1  if root is null do //Initialize a USI
2    root ← initializeBucket(u);
3  else if isGuiderLevel(root) do //Insert u into USI
4        child ← locateChild(root, S(u, root));
5        insertUser(child, u, ϵ); //Insert u into child
6      else add u to root //Insert u into root
7        if satisfySplit(root, ϵ) do //Split root into a 2-USI
8          g ← selectGuiders(root);
9          root′ ← initialize2-USI(g);
10         foreach u′ in root do
11           insertUser(root′, u′, ϵ);
12         root ← root′; //Replace root with root′
13 return root;
```

1) If *root* is null, then a bucket-level index node is initialized to load u (lines 1 and 2). In order to make the USI balance as soon as possible, it is vital to select an exact user as the initial u. The median user is usually suggested to be chosen so as to build a balanced tree directly [36]. Therefore, it should be an intuitive idea to build the USI based on presorted users. Here we use the method proposed in [37] to presort the users in the social network, and then select the median user as the initial u.

2) If *root* of the current USI is a guider-level index node (lines 3–5):

step 1: calculate the similarity between u and *root*.

step 2: locate to one child of *root* according to the

similarity. The insertion process is performed recursively with the child as the new root.

3) If the root of the current USI is a bucket-level index node (lines 6–12):

insert u into *root* and check whether the number of users in *root* reaches the split threshold ϵ. If so, *root* will be split. During splitting, one or more users are randomly selected from *root* as guiders to initialize a new two-layer USI (denoted as a 2-USI). Then the remaining users in *root* are inserted into the 2-USI. Finally, *root* is replaced by the 2-USI.

The basic idea of USI is similar to that of VP-Tree [38], an index for the nearest neighbor search. However, the direct application of VP-Tree cannot solve the problem of user identification across social networks. First, VP-tree does not consider the processing of missing features during similarity calculation. However, most social networks are incomplete due to the missing of attribute features and structural features. Second, VP-Tree requires clear partition boundaries. Due to the complexity of social networks, we often cannot get a clear partition boundary to judge whether users are similar or not. Compared with VP-Tree, our USI can better adapt to the characteristics of user identification across social networks. On the one hand, to solve the problem of missing features, we use a switch function to dynamically remove the missing features when computing the similarity between users. On the other hand, aiming at the error amplification resulted from the clear partition boundary, we adopt the interval overlapping strategy, the similarity scaling strategy and the backtracking strategy to make USI adapt to the user identification better.

4.3 Complexity Analysis

The above USI construction process uses iterative splitting to ensure that the number of users per index node does not exceed the threshold ϵ. When inserting a user into a USI, we need to go through a path from the root to the bucket-level index node where the user will be stored. The average length of the path is equal to the depth of the USI. Therefore the average time complexity of traversal is $O(\log_b n + \epsilon)$, where b represents the number of branches of each index node and n is the total number of users in the social network. Since ϵ is a constant, the time complexity is $O(\log_b n)$.

The above analysis assumes that the similarity between users and index nodes is uniformly distributed. This is because the depth of the USI approximates $\log_b n$ only if the similarity conforms to the uniform distribution. But that is not really the case. Most users in a social network are not similar to each other. Even if users share certain features such as common preferences or belong to the same community, they are not necessarily very similar. To this end, we introduce the similarity skew ratio (denoted as p_{skew}) to represent the maximum proportion of users that a subtree in USI contains. Since the most time-consuming traversal must occur in the most skew subtree, the maximum depth of traversal is the depth of such a subtree. The maximum number of possible users in each child node of the root is $n \times p_{\text{skew}}$, the maximum number of users in each next-layer child node is $n \times (p_{\text{skew}})^2$, and so on. In a leaf node, the maximum number is $n \times (p_{\text{skew}})^d$, where d is the depth of the USI. The number of users in each leaf must not exceed ϵ, that is, d is less than $(\log \epsilon - \log n)/\log p_{\text{skew}}$. Therefore, by considering the similarity skew, the worst time complexity of traversal is $O((\log \epsilon - \log n)/\log p_{\text{skew}})$.

5 Two-Phase User Identification

During the USI construction, we only consider the user features within one social network. However, there are some valuable features across social networks such as user seeds. In this section, we propose a two-phase user identification strategy, which first matches users via USI, and then expands the matching results based on user seeds.

5.1 Phase 1: USI-Based Bidirectional User Matching

Supposing a USI has been constructed based on G_A, for each user u^B ($u^B \in G_B$), the first-phase matching is to search the users from the USI which are similar to u^B. In this subsection, we need to distinguish which social network a user is from; thus we add a superscript A or B to u to indicate whether u is from G_A or G_B.

Since the USI is built based on a probabilistic model, strict interval divisions for guider-level index nodes may result in the missing of real matching users. As the search progresses, more and more such users might be lost, i.e., error amplification. For example, supposing the interval divisions are [0, 0.5) and [0.5, 1.0] and the similarity between u^B and a guilder-level index node is 0.48, u^B will be strictly allocated to only one child node to continue matching. However, there may be some matching results in the other child node, because 0.48 is very close to the interval boundary.

In order to solve the error amplification, we propose the following strategies.

1) *Interval Overlapping Strategy.* Interval overlapping means that the intersection of two adjacent intervals of a guilder-level index node is not empty. Here we introduce an overlapping rate during interval division, so that the users contained in adjacent child nodes have intersection. As shown in Fig.4, the intervals of each guilder-level index node are set to [0, 0.6] and [0.4, 1] respectively with the overlapping rate of 0.33. Supposing the similarity between u^B and x_0 is 0.5, both x_1 and x_2 will be located for further matching.

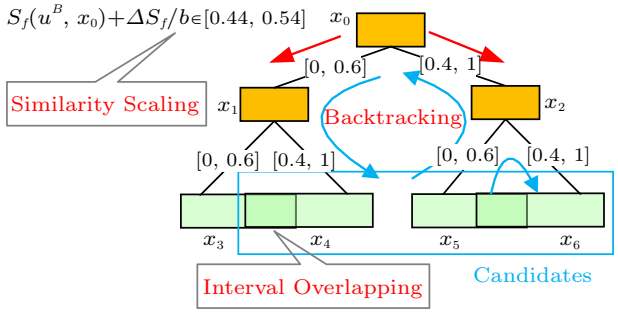


Fig.4. Three strategies for solving the error amplification.

2) *Similarity Scaling Strategy.* Although the interval overlapping strategy can alleviate the error amplification to a certain extent, it may cause a waste of space, especially when the overlapping rate is too high. Thus we define a scaling rate (denoted as ΔS_f) and propose a similarity scaling strategy. If the similarity between u^B and an index node x is $S_f(u^B, x)$, we compare $S_f(u^B, x)+\Delta S_f/b$ with each interval to determine which child nodes to be searched further. Here b is the number of branches of each index node. As shown in Fig.4, supposing $S_f(u^B, x_0)$=0.49 and ΔS_f=0.1, the scaled similarity will be [0.44, 0.54], which overlaps both [0, 0.6] and [0.4, 1]. Therefore it will continue matching with both branches of x_0.

3) *Backtracking Strategy.* The above two strategies rely on the overlapping rate and the scaling rate respectively. If they are set too small, the error amplification has not been effectively solved. Therefore, we propose a backtracking strategy. Firstly, during USI construction, we adopt the interval overlapping strategy to assign user nodes. Secondly, the similarity scaling strategy is performed to search the leaf nodes similar to u^B in the USI. Finally, these leaf nodes are backtracked up layer by layer to expand the search range. As shown in Fig.4, suppose the leaf node similar to u^B is x_5. During backtracking, we start from x_5, then backtrack to x_4 and x_6 along two paths x_5-x_2-x_0-x_1-x_4 and x_5-x_2-x_6 respectively, and match more users from x_4 and x_6 as candidates, so as to expand the result data.

Based on the above strategies, we propose a USI-based bidirectional candidates search algorithm (Algorithm 2). Given the current USI, the target user u^B, the number of candidates (denoted as K), the scaling rate ΔS_f and the similarity threshold θ_f, the output of Algorithm 2 is a queue ψ used to store candidate users from the USI which are similar to u^B. Algorithm 2 includes the following steps.

Algorithm 2. getSimilar($root$, u^B, K, ΔS_f, θ_f)

Input: the root of USI $root$, u^B, K, ΔS_f, θ_f

Output: ψ

```
1  ψ ← initializeQueue(K);
2  if isBucketLevel(root) do
3    foreach u^A in root do //Get similar users from root
4      if S_f(u^B, u^A) > θ_f and u^A.mark==available do
5        ψ ← addSimilar(u^A, S_f(u^B, u^A));
6      else foreach u^A in root do //Get similar users from
       root
7      if S_f(u^B, u^A) > θ_f do
8        ψ ← addSimilar(u^A, S_f(u^B, u^A));
9      b ← |root.branches|;
10     C ← locateChildren(root, S_f(u^B, root) ± ΔS_f/b);
11     foreach root_c in C do //Get similar users from root_c
12     ψ ← ψ∪getSimilar(root_c, u^B, K-size(ψ), ΔS_f, θ_f);
13     while size(ψ) < K and existUnvisitedChild(root)
14     neighbors ← locateNeighbors(C);
15     foreach root_n in neighbors do //Backtracking search
16       ψ ← ψ∪ getSimilar(root_n, u^B, K-size(ψ), ΔS_f, θ_f);
17 return ψ;
```

Step 1. Initialize a queue ψ for storing the search result (line 1).

Step 2. Perform one of the following two cases according to the type of the root.

Step 2.1. If the root of the USI is a bucket-level node, the similarity between u^B and each u^A in the root $root$ is calculated directly. The user, who meets the threshold θ_f and is marked as available (see Subsection 6.1), is inserted into ψ as a candidate user (lines 2–5).

Step 2.2. Otherwise (i.e., the root is a guilder-level node), the similarity between u^B and each u^A in $root$ is calculated, and the users that meet threshold θ_f are inserted into ψ (lines 6–8). Next, we calculate the average of all the similarities and take it as the similarity between u^B and $root$. Then, according to the scaled similarity, we locate the child nodes of $root$, and use them as the new roots for iterative search (lines 9–12).

If the number of users in ψ is less than K or there are some unvisited nodes, we start from these child nodes and conduct a backtracking search in the USI until the termination condition is satisfied (lines 13–16).

Step 3. Return queue ψ (line 17).

5.2 Phase 2: Seed-Based User Matching

The completeness of matching results cannot be guaranteed if only the first-phase matching is carried out. The reasons are as follows. Firstly, the USI is constructed based on the probability model; thus there is a certain probability of losing matching results. Secondly, the first-phase matching relies on the similarity between features, which is sensitive to features and may lead to a lower similarity between real matching users, especially when features are seriously missing. Therefore, we adopt the second-phase matching, i.e., seed-based user matching. The basic idea is to propagate the mapping relationship between identified user pairs (i.e., seed users) to unidentified pairs along the social networks. For each pair to be matched, the similarity between them is calculated by accumulating all the mapping relationship propagated to them, so as to expand the matching results generated in the first phase.

For u^B to be matched in G_B, how to determine its candidate users in G_A? If all the unidentified users in G_A are compared with u^B, it inevitably leads to high computational complexity. To solve this problem, we first obtain u^B's seed neighbors in G_B (i.e., the intersection of u^B's neighbors and seeds), and then determine the users in G_A to be compared with u^B via these seed neighbors. As shown in Fig.5, suppose u_1^B's seed neighbors are $\{u_2^B, u_6^B\}$, corresponding to the seed pairs (u_2^B, u_2^A) and (u_6^B, u_6^A) respectively. Then the candidate users of u_1^B are u_1^A and u_5^A, who are the neighbors of u_2^A and u_6^A respectively.

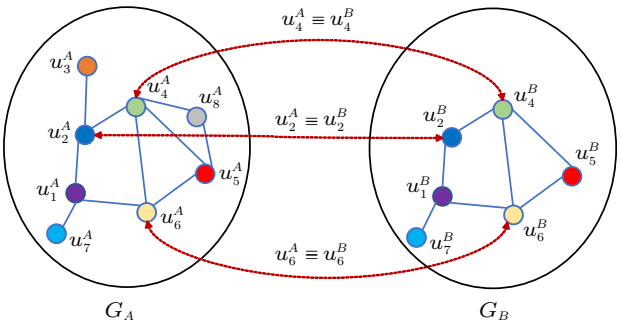


Fig.5. Example of seed-based candidate users determination.

In addition to direct neighbors, indirect neighbors can also propagate their respective mapping relationship to the user, and the influence will gradually decrease with the increase of the distance between them. Therefore, we measure the similarity between users based on such propagation relationship (as in (5)). Here $N_i^{u^B}$ (or $N_i^{u^A}$) means u^B's (or u^A's) neighbor set with i-radius ($i \in [0, d_{\max}]$). And d^{v^B} (or d^{v^A}) means user v^B's (or v^A's) degree in G_B (or G_A). $\mathcal{M}$ is the set of seed pairs connecting G_B and G_A.

$$S_p(u^B, u^A) = \sum_{i=1}^{d_{\max}} \frac{1}{i+1} \sum_{\substack{v^B \in N_i^{u^B} \wedge v^A \in N_i^{u^A} \\ \wedge (v^B,\ v^A) \cap \mathcal{M} \neq \emptyset}} \frac{1}{\sqrt{d^{v^B} \times d^{v^A}}}. \quad (5)$$

In the second phase, we comprehensively consider S_p and S_f to measure the similarity between u^B and each candidate user (as in (6)), where w is the weight of propagation-based similarity.

$$S(u^B, u^A) = w \times S_p(u^B, u^A) + (1-w) \times S_f(u^B, u^A). \quad (6)$$

In order to further expand the matching results generated in the first phase, we propose a seed-based user matching algorithm (Algorithm 3). Given G_A and G_B, the set of seed pairs connecting them (denoted as $\mathcal{M}$), the current unidentified user set in G_B (denoted as $\mathcal{R}$) and the similarity threshold θ_p, the output of Algorithm 3 is a set (denoted as χ) to store the matching user pairs.

Algorithm 3. expandResult(G_A, G_B, $\mathcal{M}$, $\mathcal{R}$, θ_p)

Input: G_A, G_B, $\mathcal{M}$, $\mathcal{R}$, θ_p
Output: χ

```
1  χ ← ∅;
2  foreach u^B in R do
3    c ← getCandidates(u^B, G_A, G_B, M);
4    foreach u^A in c do
5      if S(u^B, u^A) > θ_p do
6        χ ← χ ∪ (u^B, u^A); //Expand the current result
7  return χ;
```

Algorithm 3 further utilizes seed users to propagate the similarity from the matching results to the unidentified ones. Firstly, for each unidentified user u^B in G_B, his/her candidate users are determined in G_A (lines 2 and 3). Then, the similarity between u^B and each candidate is calculated (lines 4 and 5). Finally, the user pairs whose similarity is greater than the similarity threshold are added to the current identification results (line 6).

6 Incremental Maintenance

In this section, we present incremental maintenance strategies for the changes of social networks, including the maintenance for the constructed USI and for the current results of user identification.

6.1 Incremental Maintenance for USI

Due to the dynamics of social networks, the USI needs to be maintained dynamically. Assuming a USI has been built based on G_A, if it is completely rebuilt when G_A changes, it will cause higher computational cost. Therefore we propose incremental maintenance strategies for the following cases.

1) *User Insertion.* When a new user joins G_A, we need to insert it into the USI. The insertion process is similar to the process of building a USI (shown in Fig.6(a)). First, according to the feature similarity between u and each index node, the index nodes similar to u are located layer by layer in the USI, and u is inserted into the leaf nodes similar to it. Then, each leaf node is checked about whether it reaches the split threshold. If so, the leaf node will be split and be replaced by a 2-USI.

2) *User Deletion.* When a user is deleted from G_A, we consider the following two cases to maintain the USI (shown in Fig.6(b)). 1) If u is in a bucket-level index node, it can be deleted directly, because this will neither result in any impact on the USI structure nor affect other users in USI. 2) If u is in a guilder-level index node (supposing it is x), it cannot be simply deleted from x. There are some other users that have been guided by u and have been loaded into x's subtrees. These users will be affected by the deletion of u. If the subtree is completely rebuilt when u is deleted, this will undoubtedly cause a higher computation cost. Therefore, we do not delete any users from the USI, but give each of them a mark to indicate whether it is available. When a user has been deleted from G_A, it is marked as unavailable in the USI (Algorithm 4). This ensures that the traversal of the USI will not be affected by the deletion of users from G_A. While matching, only users marked as available can become the matching result.

3) *User Updating.* When user features are updated, we need to measure the difference between the features before and after the update. If the difference is less enough, there is no need to update the USI because the current change will only have a slight impact on the USI. Otherwise, the current USI needs to be updated (shown in Fig.6(c)). In order to avoid a great deal of cascade updating, we propose an algorithm for user updating (Algorithm 5). Supposing a user changes from u to u', the input of Algorithm 5 includes u, u', the root of USI *root*, the split threshold ϵ and the threshold for S_f. If u and u' are still similar, the current USI remains unchanged. Otherwise, the USI will be updated. Similar to the incremental user deletion proposed above, u is deleted from the USI, that is, u is either deleted directly from the USI or marked as unavailable in the USI (see Algorithm 4). Then, as a new user, u' is inserted into the USI by performing the incremental user insertion strategy proposed above (see Algorithm 1).

6.2 Incremental Maintenance for Identification

In addition to USI, the dynamic changes of social networks also affect the current identification results. Here we discuss how to incrementally maintain the identification results to be suitable for the dynamics of networks, including user insertion, deletion and updating. This impact on the identification results is mainly reflected in two aspects.

6.2.1 *From Similar to Dissimilar*

Two users were similar before, but when the social network changed, they became dissimilar. Here we discuss the following two scenarios. First, when u^B is deleted from G_B, the candidate queues which contain u^B will be affected because u^B has been already unavailable. Second, when the features of u^B change, the original users similar to u^B may no longer be similar to

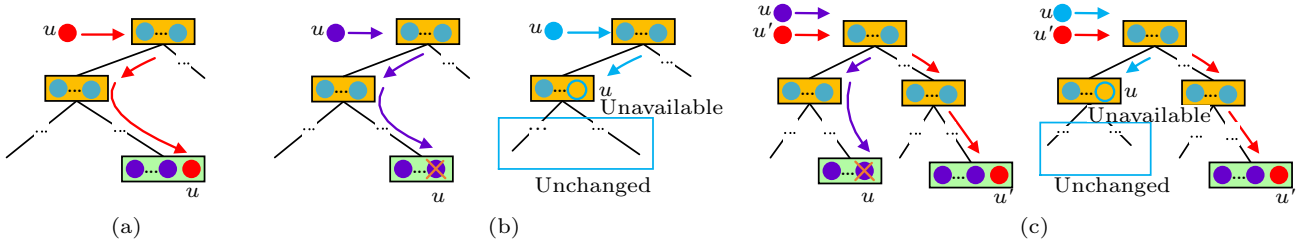


Fig.6. Illustration of incremental maintenance for USI. (a) User insertion. (b) User deletion. (c) User updating.

the state (denoted as $u^{B'}$) after u^B changes, and thus the candidate queues of these users will be affected.

Algorithm 4. deleteUser(*root*,*u*)

Input: the root of USI *root*, u

```
1  if containUser(root, u) do //Delete u from root
2    if isBucketLevel(root) do
3      removeUser(root, u); //Remove u directly
4    else u.mark ← "unavailable"; //Mark u as unavailable
5  else C ← locateChildren(root);
6      foreach root_c in C do
7        deleteUser(root_c, u); //Delete u from root_c
8  return;
```

Algorithm 5. updateUser(*root*, u, u', ϵ, θ_f)

Input: the root of USI *root*, u, u', the split threshold ϵ, θ_f

```
1  if S_f(u, u') < θ_f do
2    deleteUser(root, u); //Delete u from root
3    insertUser(root, u', ε); //Insert u' into USI
4  return;
```

For the above changes of users from similar to dissimilar, we will directly delete the dissimilar user pairs from the affected identification results. Although some space will be released in these candidate queues at this time, in order to reduce the computational cost, we will not immediately fill new user pairs, which has been excluded previously due to the length limitation of the candidate queue, into the queues. Although this will reduce the recall of identification, the accuracy of identification will not be affected. For the cases insensitive to the value of K, the impact of directly deleting candidate pairs from queues can be ignored.

Fig.7 shows some examples of incremental maintenance for the above two scenarios. As shown in Fig.7(a), when u^B exits G_B, u^B should be removed from u^A's candidate queue. As shown in Fig. 7(b), when u^B is updated to $u^{B'}$, the users previously similar to u^B (suppose they are u_1^A and u_2^A) will be affected. Since $S_f(u^{B'}, u_1^A)$ is still greater than θ_f (supposing θ_f is 0.6), $u^{B'}$ remains in the candidate queue of u_1^A. While $S_f(u^{B'}, u_2^A)$ is smaller than 0.6, $u^{B'}$ needs to be removed from the candidate queue of u_2^A.

6.2.2 *From Dissimilar to Similar*

The changes of users from dissimilar to similar are mainly caused by the changes of user features. This scenario is relatively complex, because it is difficult to determine which users will be affected by them.

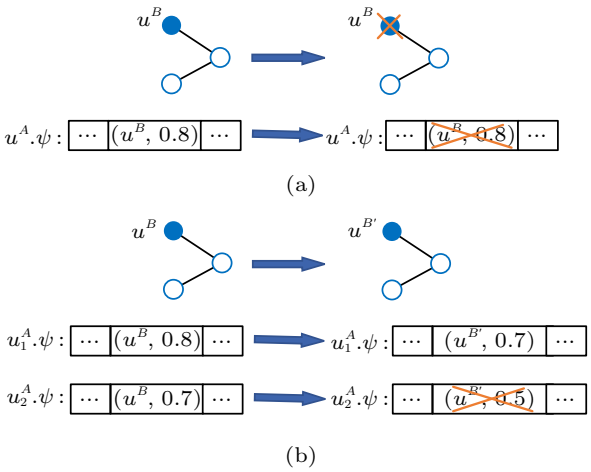


Fig.7. Examples of incremental maintenance for the scenarios from similar to dissimilar. (a) User deletion. (b) User updating.

Suppose the updated user is u^B (supposing it changes from u^B to $u^{B'}$). A simple way is to calculate the similarity between $u^{B'}$ and each user in G_A, so as to determine which users in G_A have newly become the users similar to $u^{B'}$. But this way is too expensive, because it needs to compare $u^{B'}$ with every user in G_A. In fact, the change of u^B will only affect those users who are similar to $u^{B'}$. We can search such users via the USI. Then the users needing to be recomputed are only limited within a local region of the USI, and thus it will not cause too much computational cost.

Fig.8 shows an example of incremental maintenance for the scenario from dissimilar to similar. Before u^B changes, neither u_1^A nor u_2^A is in u^B's candidate queue. Similarly, u^B is not in the candidate queues of u_1^A and u_2^A too. After changing, USI can be used to quickly search users similar to $u^{B'}$. Supposing both u_1^A and u_2^A become similar to $u^{B'}$, they constitute an affecting subset of the identification results. At this time, the candidate queues of $u^{B'}$, u_1^A and u_2^A need to be updated respectively.

For two social networks, we usually select the relatively stable one from them to build a USI. Then for each user that changes in the other network, we need to determine its affecting subset via the USI. Finally only the users in the affecting subset should be recomputed. However, if both networks change frequently, we need to build USIs for both of them, because efficient similarity computing needs to be supported by both networks.

7 Experiments

In this section, we conduct experimental studies to evaluate the effectiveness and the efficiency of our pro-

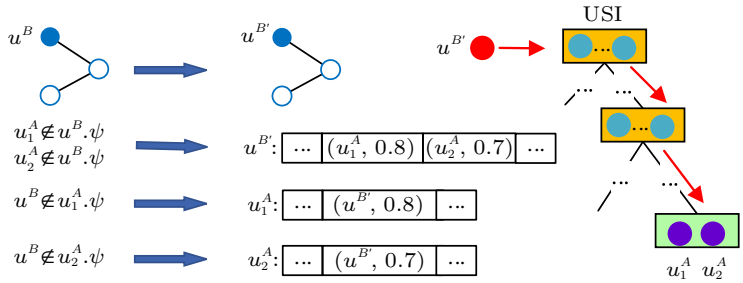


Fig.8. Example of incremental maintenance for the scenario from dissimilar to similar.

posed method.

7.1 Experimental Settings

We implement the experiments on a PC with Intel® Xeon® E5-2620 V4 CPU @ 2.10 GHZ, 32 GB main memory and 512 GB hard disk. As shown in Table 2, we use three real-world social graphs: EMAIL④, DBLP⑤ and Last.fm⑥.

Table 2. Description of Real Datasets

Dataset	Number of Nodes	Number of Edges	Graph Type	Number of Labels
EMAIL	1 133	5 451	Undirected	5
DBLP	9 142	32 676	Undirected	1
Last.fm	136 420	1 685 524	Undirected	1

The three cross-network datasets are generated by synthesis, following the existing work [39, 40]. As described in Section 1, the incompleteness of social networks mainly lies in the lack of user attributes and seeds. Formally, given a graph G_A with adjacency matrix $\boldsymbol{A}$, we generate a noisy graph G_B with matrix $\boldsymbol{B}$, where $\boldsymbol{B}$ is randomly generated by perturbing $\boldsymbol{A}$. By adding different levels (i.e., the perturbation rate, denoted as η) of synthetic noise, both the graph structure and labels are perturbed to simulate real-world scenarios. That is, different degrees of incomplete G_B can be achieved via perturbation on G_A. The higher the value of η, the higher the incompleteness of G_B. We simulate the effect of perturbation on G_A as follows: 1) add edges to G_A in proportion to η; 2) remove edges from G_A in proportion to η; 3) change the labels of nodes in G_A in proportion to η (the new values are not null); 4) set the labels of nodes in G_A to null in proportion to η.

The problem of user identification can be considered as a binary classification problem, which needs to determine whether two users are matching or mismatching. For classification, the samples are divided into four types: true positive (TP), false positive (FP), true negative (TN) and false negative (FN). To evaluate the performance of user identification, we employ the standard evaluation metrics in information retrieval: precision (denoted as P) and recall (denoted as R), which are defined as (7) and (8) respectively. Besides, we use rank score (denoted as RS), which is a widely-used evaluation metric in many real user identification applications, to evaluate the top-k candidates for user identification (as shown in (9)). Here T is the set of candidate queues and l^i is the position of real matching users in the candidate matching queue.

$$P = \frac{TP}{TP + FP}. \tag{7}$$

$$R = \frac{TP}{TP + FN}. \tag{8}$$

$$RS = \sum_{i \in T} \frac{1}{l^i}. \tag{9}$$

The baselines can be divided into three groups: an attribute-based user identification method (ULink [1]), two structure-based user identification methods (NetAlignMP [14] and IsoRank [13]) and two hybird user identification methods (HashAlign [35] and MAUIL [30]).

ULink [1]. The profiles of users are compared in the latent user space.

NetAlignMP [14]. The overlap of structures is considered for message passing.

IsoRank [13]. It simultaneously uses the network data and the sequence similarity data to compute network alignments.

HashAlign [35]. It leverages structural properties and node attributes simultaneously.

④konect.uni-koblenz.de/networks, Aug. 2022.

⑤dblp.uni-trier.de/db, Aug. 2022.

⑥aminer.cn/cosnet, Aug. 2022.

MAUIL[30]. It combines the text attributes for each network with user relationships to learn the final representation of each user.

7.2 Performance Comparison

Fig.9 shows the precision, recall, rank score and time cost of different user identification methods on the three cross-network datasets, respectively.

For precision (shown in Figs.9(a), 9(d) and 9(g)), our first observation is that with the increase of η, the precision of identification decreases gradually. Second, HashAlign may lose some useful information (such as the association between labels) when transforming user features into feature vectors, and thus its precision is lower in the three datasets. Third, for EMAIL, the precision of our CURIOUS method is basically equal to that of ULink, NetAlignMP, IsoRank and MAUIL. This is because the dataset is rich in features, and in this case, the advantage of our method is not obvious enough. Compared with EMAIL, there are fewer node features (e.g., node labels) that can be obtained from DBLP or Last.fm, resulting in a lower precision than EMAIL. ULink only considers user attributes, and ignores the structure features. In contrast, NetAlignMP and IsoRank rely on the structural features of the networks without considering user attributes. Although both user attributes and structure features are considered by MAUIL, it focuses on node embedding by user

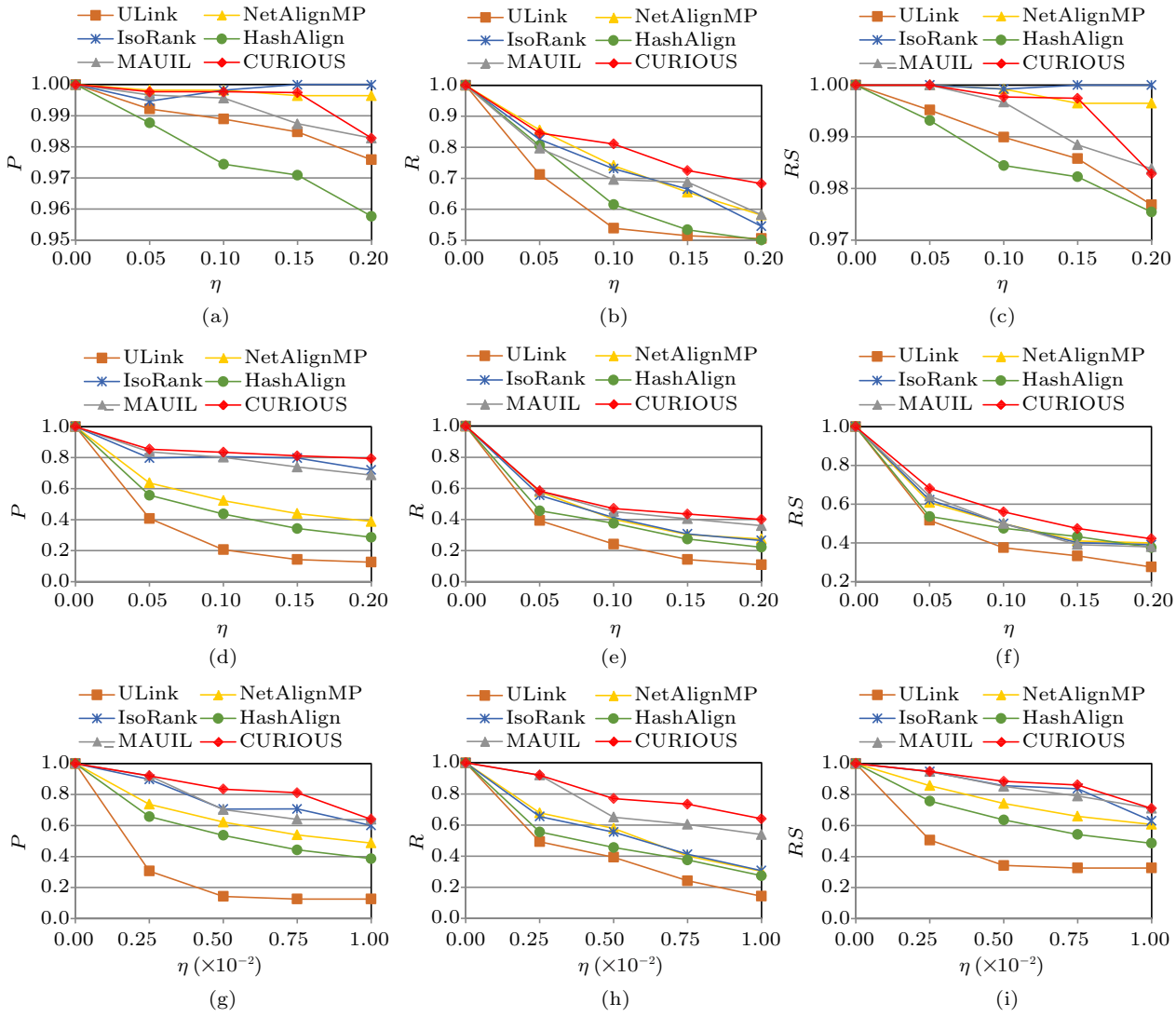


Fig.9. Performance comparison of different user identification methods. (a) Precision on EMAIL. (b) Recall on EMAIL. (c) Rank score on EMAIL. (d) Precision on DBLP. (e) Recall on DBLP. (f) Rank score on DBLP. (g) Precision on Last.fm. (h) Recall on Last.fm. (i) Rank score on Last.fm.

attributes, while only simple neighborhood information (first-order proximity and second-order proximity) is considered. When few user attributes are available, its advantage will be weakened. Compared with them, our method comprehensively considers user attributes, social circle features, social chain features and neighborhood information, which is still effective for user identification across incomplete networks and thus has a higher precision.

As shown in Figs.9(b), 9(e), and 9(h), with the increase of η, the recall of identification decreases gradually. Different from others, our method adopts two-phase user identification to maintain the recall of the results. Specifically, in the first phase we adopt three strategies (including interval overlapping, similarity scaling and backtracking) to avoid losing candidates. Also we adopt the second-phase matching, i.e., seed-based user matching, to expand the matching results generated in the first phase. Therefore, our method outperforms the others in recall.

Figs.9(c), 9(f), and 9(i) show that the change trend of rank score is consistent with that of precision. However, because the rank score is used to measure the matching of a group of candidate users, it is higher than the corresponding precision.

We compare the time cost of our method (i.e., the sum of the time cost of phase-1 matching and phase-2 matching) and baselines (shown in Tables 3–5). Besides, we evaluate the time cost for constructing a USI. Both HashAlign and our CURIOUS method use indexes to quickly locate users similar to the incoming user. The time cost of CURIOUS and HashAlign is similar, which is lower than that of the other methods. However, as the data scale increases (e.g., Last.fm), the time cost of HashAlign increases significantly, which is higher than that of CURIOUS obviously. This indicates CURIOUS has a more obvious advantage on time cost with the increase of data scale.

Compared with the traditional methods, our method improves the precision, recall and rank score by an average of 0.19, 0.16 and 0.09 respectively, and reduces the time cost by an average of 81%.

7.3 Ablation Analysis

In this subsection, we compare our method with its following variants: 1) StaticUSIMatch: only the first phase, i.e., USI-based bidirectional user matching, is adopted; 2) StaticMatch: both USI-based bidirectional user matching and seed-based user matching are adopted, but the incremental maintenance is lacked; 3) DynamicUSIMatch: based on StaticUSIMatch, the incremental maintenance is adopted, but the second phase matching is lacked.

Table 3. Time Cost (s) Comparison on EMAIL

Method	η			
	0.05	0.10	0.15	0.20
ULink	237.76	232.13	239.63	242.19
NetAlignMP	162.23	146.01	150.30	161.15
IsoRank	634.27	633.18	620.67	585.27
HashAlign	10.01	13.87	11.85	16.75
MAUIL	767.18	771.65	769.33	774.54
USI Construction	6.21	6.45	6.61	6.16
Phase-1	12.97	12.56	12.47	12.79
Phase-2	0.22	0.41	0.29	0.42
CURIOUS	13.19	12.97	12.76	13.21

Table 4. Time Cost (s) Comparison on DBLP

Method	η			
	0.05	0.10	0.15	0.20
ULink	1 612.25	1 632.56	1 589.45	1 627.37
NetAlignMP	4 294.35	4 116.71	4 273.57	4 173.55
IsoRank	5 277.06	5 034.54	5 032.54	5 294.29
HashAlign	242.67	538.43	191.73	319.87
MAUIL	3 998.31	3 976.17	4 002.19	3 010.63
USI Construction	32.59	33.87	33.29	32.46
Phase-1	249.52	181.52	198.14	170.68
Phase-2	0.83	1.72	2.56	2.08
CURIOUS	250.35	183.24	200.70	172.76

Table 5. Time Cost (s) Comparison on Last.fm

Method	η			
	0.002 5	0.005 0	0.007 5	0.010 0
ULink	13 492	16 672	17 296	17 373
NetAlignMP	28 824	26 182	24 257	24 832
IsoRank	47 763	44 903	47 026	46 792
HashAlign	15 772	17 254	14 771	15 665
MAUIL	20 675	20 714	20 824	20 139
USI Construction	59	56	58	73
Phase-1	1 398	1 490	1 425	1 489
Phase-2	898	816	1 347	1 765
CURIOUS	2 296	2 306	2 772	3 254

We simulate the dynamics of social networks by inserting new users. To evaluate the effectiveness of our proposed method, experiments are carried out on dynamic networks by varying the changing rate (denoted as μ). Since the experimental results on DBLP are consistent with those on EMAIL, only the results on EMAIL and Last.fm are demonstrated here.

As shown in Figs.10(a) and 10(d), our first observation is that the one-phase identification (i.e., StaticUSI-

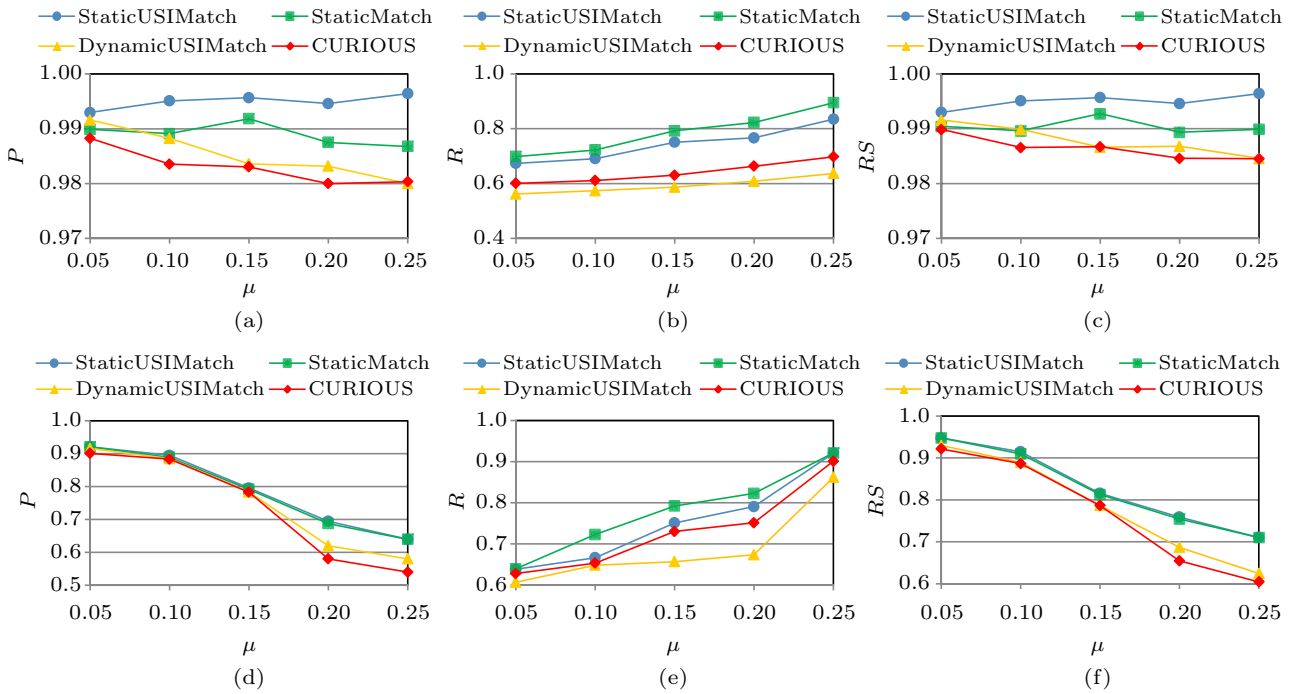


Fig.10. Performance comparison of variants. (a) Precision on EMAIL. (b) Recall on EMAIL. (c) Rank score on EMAIL. (d) Precision on Last.fm. (e) Recall on Last.fm. (f) Rank score on Last.fm.

Match and DynamicUSIMatch) slightly outperforms the two-phase identification (i.e., CURIOUS and StaticMatch) on precision. This is because the second phase only extends the results, but does not refine them. Second, as shown in Figs.10(b) and 10(e), the two-phase identification outperforms the one-phase identification on recall. This is because the former utilizes the seeds to further expand the matching results generated by the latter. Also, with the increase of user scale, the recall increases gradually, because the structural features of users become richer with the increase of users. Third, Figs.10(c) and 10(f) show that the change trend of rank score is consistent with that of precision. Finally, as shown in Table 6 and Table 7, the dynamic methods (i.e., CURIOUS and DynamicUSIMatch) outperform the static methods (i.e., StaticUSIMatch and StaticMatch) on time cost. This is because the former adopts incremental maintenance strategies that do not re-identify all the users. The latter adopts static identification strategies. Once the networks change, the entire networks need to be identified again. With the increase of data scale, the dynamic methods have a more obvious superiority to the static methods.

7.4 Parameter Settings

In this subsection, we evaluate the effect of parameters, including b (the number of branches of each index node), ϵ (the threshold for splitting), K, θ_f (the threshold for S_f), θ_p (the threshold for S_p), ΔS_f (the similarity scaling rate) and w (the weight of propagation-based similarity). We take the EMAIL dataset as an example to show the effect of different parameter settings on various metrics. The experimental results are shown in Table 8 and Table 9.

Table 6. Time Cost (s) Comparison of Variants on EMAIL

Method	μ			
	0.05	0.10	0.15	0.20
StaticUSIMatch	12.83	12.64	14.37	15.91
StaticMatch	13.11	12.95	14.62	16.25
DynamicUSIMatch	2.13	2.59	3.53	4.12
CURIOUS	2.64	3.16	4.25	4.87

Table 7. Time Cost (s) Comparison of Variants on Last.fm

Method	μ			
	0.05	0.10	0.15	0.20
StaticUSIMatch	1 391	1 471	1 459	1 498
StaticMatch	2 285	2 467	2 543	2 696
DynamicUSIMatch	65	70	68	67
CURIOUS	959	1 066	1 152	1 265

With the increase of b, the time cost and the recall decrease, but its increase has little effect on the preci-

Table 8. Parameter Settings for b, ϵ and K

Metric	b					ϵ					K				
	10	40	70	100	130	10	20	30	40	50	2	4	6	8	10
P	0.99	1.00	0.99	0.99	1.00	1.00	0.99	0.99	0.98	0.97	0.98	0.99	1.00	1.00	1.00
R	0.90	0.85	0.80	0.76	0.69	0.84	0.85	0.85	0.84	0.85	0.85	0.85	0.84	0.83	0.84
RS	0.99	1.00	0.99	0.99	1.00	1.00	0.99	0.99	0.98	0.97	0.98	0.99	1.00	1.00	1.00
Time (s)	46.61	14.62	10.10	9.30	8.41	12.66	18.46	17.66	15.76	25.67	12.45	12.88	12.72	13.72	13.31

Table 9. Parameter Settings for θ_f, θ_p, ΔS_f and w

Metric	θ_f				θ_p				ΔS_f					w			
	0.6	0.7	0.8	0.9	0.6	0.7	0.8	0.9	0.25	0.50	0.75	1.00	1.25	0.2	0.4	0.6	0.8
P	0.95	0.99	1.00	1.00	0.96	0.98	0.98	1.00	1.00	0.99	1.00	0.99	1.00	0.98	0.98	0.71	0.52
R	0.86	0.85	0.83	0.79	0.83	0.81	0.80	0.76	0.82	0.89	0.89	0.92	0.92	0.53	0.65	0.77	0.82
RS	0.95	0.99	1.00	1.00	0.96	0.98	0.98	1.00	1.00	0.99	1.00	0.99	1.00	0.98	0.98	0.78	0.59
Time (s)	12.98	12.33	14.01	13.67	12.35	12.46	13.98	13.59	12.38	18.21	22.63	35.77	37.87	12.38	12.36	14.42	14.01

sion and the rank score. There are fewer users in each subtree; thus the ability of the USI to distinguish and filter dissimilar users is enhanced. But simultaneously, it increases the probability of dividing similar users into different branches, which further aggravates the error amplification, leading to a lower recall.

For the parameter ϵ, when it increases, the time cost of identification increases too. This is because the larger the value of ϵ, the more the users in each index node, resulting in a higher computational cost in the index node. The recall almost remains unchanged. For the precision and rank score, they decrease slightly with the increase of ϵ. This is because a higher ϵ might result in mixing more users who are not similar to each other within the same index node.

For the number of candidates K, the rank score increases slightly with the increase of K. This is because a higher precision means a higher probability that user pairs with a high similarity are the same users.

For θ_f and θ_p, they have almost no impact on time consumption. For precision, it increases with the increase of θ_f and θ_p. For recall, its change trend is opposite to precision. When the similarity threshold is too high, the evaluation conditions are too strict, and the real similar users may be judged as dissimilar, resulting in the decline of recall.

The parameter ΔS_f is to expand the search region of the USI to adjacent subtrees or index nodes, so as to alleviate the error amplification. As ΔS_f increases, the recall increases, but the time cost increases too. The experimental results show that better performance can be achieved without too high ΔS_f. For precision and rank score, they are not particularly sensitive to the change of ΔS_f.

We also evaluate the performance as w varies. The larger the value of w is, the more dominant the second phase is. At this time, the recall is higher, while the precision is lower. Through a comprehensive evaluation, its best value is 0.4, which illustrates the first phase is more decisive for the final result.

8 Conclusions

We proposed an incremental user identification method across social networks based on the user-guider similarity index. First, we constructed a novel user-guider similarity index to speed up the matching between users. Then we proposed a two-phase user identification strategy to efficiently identify users, which is still effective for incomplete networks. We also proposed incremental maintenance for both USI and the identification results, which dynamically captures the instant states of social networks. Currently, two phases of matching are carried out separately. Next, we will take advantage of their interplay. Beyond that, we will work on incremental user identification for very large datasets in a distributed environment.

References

[1] Mu X, Zhu F, Lim E, Xiao J, Wang J, Zhou Z. User identity linkage by latent user space modelling. In *Proc. the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Aug. 2016, pp.1775-1784. DOI: 10.1145/2939672.2939849.

[2] Kemp S. Global overview report. Technical Report, Hootsuite, 2022. https://datareportal.com/reports/digital-2022-global-overview-report, Jan. 2022.

[3] Liu J, Zhang F, Song X, Song Y, Lin C, Hon H. What's in a name? An unsupervised approach to link users across communities. In *Proc. the 6th ACM International Conference on Web Search and Data Mining*, Feb. 2013, pp.495-504, DOI: 10.1145/2433396.2433457.

[4] Zhang J, Kong X, Yu P S. Transferring heterogeneous links across location-based social networks. In *Proc. the 7th ACM International Conference on Web Search and Data Mining*, Feb. 2014, pp.303-312, DOI: 10.1145/2556195.2559894.

[5] Li Y, Peng Y, Zhang Z, Yin H, Xu Q. Matching user accounts across social networks based on username and display name. *World Wide Web*, 2019, 22(3): 1075-1097. DOI: 10.1007/s11280-018-0571-4.

[6] Liu J, Chai G, Luo Y, Feng J, Tang N. Feature augmentation with reinforcement learning. In *Proc. the 38th IEEE International Conference on Data Engineering*, May 2022, pp.3360-3372. DOI: 10.1109/ICDE53745.2022.00317.

[7] Backstrom L, Leskovec J. Supervised random walks: Predicting and recommending links in social networks. In *Proc. the 4th International Conference on Web Search and Web Data Mining*, Feb. 2011, pp.635-644, DOI: 10.1145/1935826.1935914.

[8] Zafarani R, Liu H. Connecting users across social media sites: A behavioral-modeling approach. In *Proc. the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Aug. 2013, pp.41-49. DOI: 10.1145/2487575.2487648.

[9] Shao J, Wang Y, Gao H, Shen H, Li Y, Cheng X. Locate who you are: Matching geo-location to text for user identity linkage. In *Proc. the 30th ACM International Conference on Information and Knowledge Management*, Nov. 2021, pp.3413-3417. DOI: 10.1145/3459637.3482134.

[10] Feng J, Li Y, Yang Z, Zhang M, Wang H, Cao H, Jin D. User identity linkage via co-attentive neural network from heterogeneous mobility data. *IEEE Trans. Knowl. Data Eng.*, 2022, 34(2): 954-968. DOI: 10.1109/TKDE.2020.2989732.

[11] Nilizadeh S, Kapadia A, Ahn Y. Community-enhanced de-anonymization of online social networks. In *Proc. the 21st ACM SIGSAC Conference on Computer and Communications Security*, Nov. 2014, pp.537-548. DOI: 10.1145/2660267.2660324.

[12] Zhou X, Liang X, Zhang H, Ma Y. Cross-platform identification of anonymous identical users in multiple social media networks. *IEEE Trans. Knowl. Data Eng.*, 2016, 28(2): 411-424. DOI: 10.1109/TKDE.2015.2485222.

[13] Singh R, Xu J, Berger B. Global alignment of multiple protein interaction networks with application to functional orthology detection. *Natl. Acad. Sci. USA*, 2008, 105(35): 12763-12768. DOI: 10.1073/pnas.0806627105.

[14] Bayati M, Gleich D, Saberi A, Wang Y. Message-passing algorithms for sparse network alignment. *ACM Trans. Knowl. Discov. Data*, 2013, 7(1): Article No. 3. DOI: 10.1145/2435209.2435212.

[15] Nassar H, Gleich D. Multimodal network alignment. In *Proc. the 17th SIAM International Conference on Data Mining*, Apr. 2017, pp.615-623. DOI: 10.1137/1.9781611974973.69.

[16] Nassar H, Veldt N, Mohammadi S, Grama A, Gleich D. Low rank spectral network alignment. In *Proc. the 27th International World Wide Web Conference*, Apr. 2018, pp.619-628. DOI: 10.1145/3178876.3186128.

[17] Zhou X, Liang X, Du X, Zhao J. Structure based user identification across social networks. *IEEE Trans. Knowl. Data Eng.*, 2018, 30(6): 1178-1191. DOI: 10.1109/TKDE.2017.2784430.

[18] Mao X, Wang W, Wu Y, Lan M. Boosting the speed of entity alignment 10 X: Dual attention matching network with normalized hard sample mining. In *Proc. the 30th International World Wide Web Conference*, Apr. 2021, pp.821-832. DOI: 10.1145/3442381.3449897.

[19] Chen X, Song X, Peng G, Feng S, Nie L. Adversarial-enhanced hybrid graph network for user identity linkage. In *Proc. the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, Jul. 2021, pp.1084-1093. DOI: 10.1145/3404835.3462946.

[20] Li X, Cao Y, Li Q, Shang Y, Li Y, Liu Y, Xu G. RLINK: Deep reinforcement learning for user identity linkage. *World Wide Web*, 2021, 24(1): 85-103. DOI: 10.1007/s11280-020-00833-8.

[21] Chu X, Fan X, Yao D, Zhu Z, Huang J, Bi J. Cross-network embedding for multi-network alignment. In *Proc. the 28th International World Wide Web Conference*, May 2019, pp.273-284. DOI: 10.1145/3308558.3313499.

[22] Chen H, Yin H, Sun X, Chen T, Gabrys B, Musial K. Multi-level graph convolutional networks for cross-platform anchor link prediction. In *Proc. the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, Aug. 2020, pp.1503-1511. DOI: 10.1145/3394486.3403201.

[23] Shun F, Wang G, Xia S, Liu L. Deep multi-granularity graph embedding for user identity linkage across social networks. *Knowl. Based Syst.*, 2020, 193: Article No. 105301. DOI: 10.1016/j.knosys.2019.105301.

[24] Yan J, Yang S, Hancock E. Learning for graph matching and related combinatorial optimization problems. In *Proc. the 29th International Joint Conference on Artificial Intelligence*, Jan. 2021, pp.4988-4996. DOI: 10.24963/ijcai.2020/683.

[25] Zhou F, Zhang K, Xie S, Luo X. Learning to correlate accounts across online social networks: An embedding-based approach. *INFORMS J. Comput.*, 2020, 32(3): 714-729. DOI: 10.1287/ijoc.2019.0911.

[26] Chu X, Fan X, Zhu Z, Bi J. Variational cross-network embedding for anonymized user identity linkage. In *Proc. the 30th ACM International Conference on Information and Knowledge Management*, Nov. 2021, pp.2955-2959. DOI: 10.1145/3459637.3482214.

[27] Qian J, Li X, Zhang C, Chen L. De-anonymizing social networks and inferring private attributes using knowledge graphs. In *Proc. the 35th Annual IEEE International Conference on Computer Communications*, Apr. 2016. DOI: 10.1109/INFOCOM.2016.7524578.

[28] Heimann M, Shen H, Safavi T, Koutra D. REGAL: Representation learning-based graph alignment. In *Proc. the 27th ACM International Conference on Information and Knowledge Management*, Oct. 2018, pp.117-126. DOI: 10.1145/3269206.3271788.

[29] Yasar A, Çatalyürek Ü. An iterative global structure-assisted labeled network aligner. In *Proc. the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Aug. 2018, pp.2614-2623. DOI: 10.1145/3219819.3220079.

[30] Chen B, Chen X. MAUIL: Multilevel attribute embedding for semisupervised user identity linkage. *Inf. Sci.*, 2022, 593: 527-545. DOI: 10.1016/j.ins.2022.02.023.

[31] Fang Z, Cao Y, Liu Y, Tan J, Guo L, Shang Y. A co-training method for identifying the same person across social networks. In *Proc. the 5th IEEE Global Conference on Signal and Information Processing*, Nov. 2017, pp.1412-1416. DOI: 10.1109/GlobalSIP.2017.8309194.

[32] Zhong Z, Cao Y, Guo M, Nie Z. CoLink: An unsupervised framework for user identity linkage. In *Proc. the 32nd AAAI Conference on Artificial Intelligence*, Feb. 2018, pp.5714-5721. DOI: 10.1609/aaai.v32i1.12014.

[33] Xie Z, Zhu R, Zhao K, Liu J, Zhou G, Huang J. A contextual alignment enhanced cross graph attention network for cross-lingual entity alignment. In *Proc. the 28th International Conference on Computational Linguistics*, Dec. 2020, pp.5918-5928. DOI: 10.18653/v1/2020.coling-main.520.

[34] Xiang Y, Zhang Z, Chen J, Chen X, Lin Z, Zheng Y. OntoEA: Ontology-guided entity alignment via joint knowledge graph embedding. In *Proc. the Joint Conference of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*, Aug. 2021, pp.1117-1128. DOI: 10.18653/v1/2021.findings-acl.96.

[35] Heimann M, Lee W, Pan S, Chen K, Koutra D. HashAlign: Hash-based alignment of multiple graphs. In *Proc. the 22nd Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, Jun. 2018, pp.726-739. DOI: 10.1007/978-3-319-93040-4_57.

[36] Agarwal P, Fox K, Munagala K, Nath A. Parallel algorithms for constructing range and nearest-neighbor searching data structures. In *Proc. the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, June 26-July 1, 2016, pp.429-440. DOI: 10.1145/2902251.2902303.

[37] Brown R. Building a balanced k-d tree in $O(kn \log n)$ time. *Journal of Computer Graphics Techniques*, 2015, 4(1): 50-68.

[38] Yianilos P. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proc. the 4th Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms*, Jan. 1993, pp.311-321.

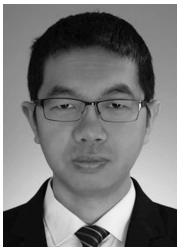
[39] Narayanan A, Shmatikov V. De-anonymizing social networks. In *Proc. the 30th IEEE Symposium on Security and Privacy*, May 2009, pp.173-187. DOI: 10.1109/SP.2009.22.

[40] Zhang S, Tong H. FINAL: Fast attributed network alignment. In *Proc. the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Aug. 2016, pp.1345-1354. DOI: 10.1145/2939672.2939766.

Yue Kou is an associate professor at School of Computer Science and Engineering, Northeastern University, Shenyang. She received her Ph.D. degree in computer software and theory from Northeastern University, Shenyang, in 2009. She is a member of CCF. Her main research interests include Web data management and social networks analysis.

Dong Li is an associate professor at School of Information, Liaoning University, Shenyang. He received his Ph.D. degree in computer software and theory from Northeastern University, Shenyang, in 2019. He is a member of CCF. His main research interests include social networks analysis and data mining.

De-Rong Shen is a professor at School of Computer Science and Engineering, Northeastern University, Shenyang. She received her Ph.D. degree in computer software and theory from Northeastern University, Shenyang, in 2004. She is a senior member of CCF. Her main research interests include Web data management and data integration.



Tie-Zheng Nie is an associate professor at School of Computer Science and Engineering, Northeastern University, Shenyang. He received his Ph.D. degree in computer software and theory from Northeastern University, Shenyang, in 2009. He is a senior member of CCF. His main research interests include big data management and data integration.

Ge Yu is a professor at School of Computer Science and Engineering, Northeastern University, Shenyang. He received his Ph.D. degree in computer science from Kyushu University of Japan, Fukuoka, in 1996. He is a senior member of CCF, and a member of ACM and IEEE. His research interesting includes database theory and technology, distributed and parallel systems, and big data management.