

國立交通大學

資訊科學與工程研究所

碩 士 論 文

PLPF – 探勘網路行為履歷於連線行為預測

PLPF - A Profile-based Link Prediction Framework in a
Time-evolving Graph

研 究 生：李政輝

指 導 教 授：彭文志 教授

李素瑛 教授

中 華 民 國 一 百 年 六 月

PLPF: 探勘網路行為履歷於連線行為預測
PLPF: A Profile-based Link Prediction Framework in a Time-evolving Graph

研究生：李政輝

Student : Zheng-Hui Lee

指導教授：彭文志

Advisor : Wen-Chih Peng

李素瑛

Suh-Yin Lee

國立交通大學
資訊科學與工程研究所
碩士論文

A Thesis

Submitted to Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

June 2011

Hsinchu, Taiwan, Republic of China

中華民國一百年六月

PLPF：探勘網路行為履歷於連線行為預測

研究生： 李政輝

指導教授： 彭文志博士
李素瑛博士

國立交通大學資訊科學與工程研究所 碩士班

摘要

人們的生活中處處充滿了不同的連結行為，例如連上網站漫遊、寄信給朋友或是打電話給友人等。這些連結行為最適合透過演進圖來觀察使用者的連結現象。假設我們有一個持續紀錄使用者連結行為的日誌，則當一個不同的連結對象出現時，我們可能會相當好奇還有那些使用者也會跟這個新的對象建立連結。

在這裡，我們提出了一個預測新連線行為的架構。透過持續紀錄、更新使用者的連結行為履歷，我們可以了解使用者的喜好與興趣，並以此為基準預測新的連線行為。我們的中心想法是：如果使用者與他人有一致的喜好或興趣，則他們有很大的可能會與相同的對象建立連結，當然也包括了新出現的連結對象。在這裡，我們不止紀錄使用者的常態連結行為於其履歷，同時也紀錄了使用者最近才有的、不同以往的連結行為。

我們用了一個現實生活中的連結資料來驗證我們的想法，並與目前最新、最好的方法進行比較。實驗結果顯示，我們的方法不僅比已知最新、最好的方法有更好的效能，而且擁有更少的計算時間，不會隨著的歷史資料的長度而增加。除此之外，我們了解到預測這類新的連結行為時，觀察使用者最近改變的興趣、喜好，比觀察使用者長久以來的興趣、喜好來得有用。

關鍵字：連線行為預測、連線行為履歷、演進圖。

PLPF: A Profile-based Link Prediction Framework in a Time-evolving Graph

Student: Zheng-Hui Lee

Advisor: Dr. Wen-Chih Peng
Dr. Suh-Yin Lee

Institute of Computer Science and Engineering
National Chiao Tung University

ABSTRACT

Connection logs are widely used to record connection behavior between sources and destinations, such as users and sites in the internet network, and are best represented as an evolving graph to illustrate the evolving connection behavior of users. Given a continuously recording connection log, when a new destination appears (i.e., connected by a few sources initially), one may want to know who else will then likely to connect it.

In this work, we propose a framework using profiles of sources to predict the top- k possible links to the specific destination from k different sources which have never connected to it before, where a profile records the connection behavior of a source including the static and surprising features to represent his/her long-term and short-term interests. The concept behind our framework is that sources have similar connection behavior before are likely to have similar connection behavior after, hence sources having similar profiles as the first source who connects to the new destination are best candidates to connect the same destination later.

In the experiment, we use a real dataset of the internet network to evaluate the performance of our method, and compare it to the state-of-the-art method driven by information flow [20]. It shows that our method has improvement in the effectiveness while comparing to the previous method, and our method has a consistent computation time cost rather than the increasing computation time cost using previous method while the whole connection log increases. Furthermore, we show that surprising features of connection behavior are much more useful when predicting new links in the internet network.

Keywords: link prediction, profile, evolving graph.

誌 謝

這篇碩士論文能夠順利完成，首先要感謝指導老師彭文志教授與李素瑛教授，老師不僅在學術上的悉心教導令我受惠良多，對於求學過程中的照顧、以及做人處事上的提點，皆令我感念在心，由衷地感謝老師。同時也謝謝戴碧如教授、黃俊龍教授擔任我的口試委員，並在口試時給予提點、意見，以及對論文提供修正建議，都使得本篇論文能更加完整。

感謝 Advanced Database System 實驗室的學長姐們，和一同來到資工所的同學翔任、端賢，不只是研究上的討論、或是生活上的經驗分享，都給了我很大的幫助，謝謝你們。

感謝系計中的學長姐、同學以及學弟妹們，有你們的陪伴與合作，讓我的研究生活更加充實。還有應數系的朋友們，時常與我保持聯絡、交流，讓我在研究生活中也感受到你們的活力、謝謝你們。

此外要特別感謝在交大室外場一起打排球的同好們，除了舉辦各式各樣新奇有趣的活動，讓我在研究生活之外充滿活力與歡笑，也常讓我有溫暖、窩心的感受，這都是很重要的一部分，在此特別感謝文鈞、揚威、泰毓、佑崧和育豪。

最後要感謝我的家人，感謝父母的栽培以及在我求學過程中給我的建議與支持，謝謝哥哥的關心，你們的支持是我順利完成學業過程中不可或缺的一部分。

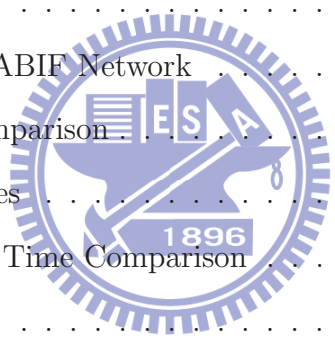
感謝所有在我研究生涯中曾幫助過我的人，謹以此文論獻給我摯愛的家人、以及所有關心我的師長、朋友，特別是字卿，謝謝你們。

Contents

中文摘要	i
Abstract	ii
Acknowledge	iii
Table of Contents	iv
List of Tables	vi
List of Figures	vii
1 Introduction	1
2 Related Works	4
3 Preliminary	6
3.1 Connection Log & Connection Network	6
3.1.1 Connection Log	6
3.1.2 Connection Network	7
3.2 Evolving Graph Model	9
3.3 Problem Formulation	11
4 PLPF: Profile-based Link Prediction Framework	12
4.1 Framework Overview	12



4.2	Graph Generation	15
4.3	Profiles	16
4.3.1	Connection Features	17
4.3.2	Profile Construction	20
4.4	Similarity Calculation	22
4.4.1	Similarity Combinations	23
4.4.2	Similarity of Static Features	24
4.4.3	Similarity of Surprising Features	25
4.4.4	Similarities in PLPF	26
4.5	Additional Domain Knowledge	27
5	Experiments	29
5.1	Experiment Settings	29
5.2	Comparisons with EABIF Network	31
5.2.1	Precision Comparison	31
5.2.2	Instant Profiles	33
5.2.3	Computation Time Comparison	35
5.3	Parameters in PLPF	36
5.3.1	Parameters for Static Features	36
5.3.2	Parameters for Surprising Features	37
5.3.3	Parameters for Both Features	39
6	Conclusion and Future Work	41



List of Tables

3.1	5 connections in the internet network, sorted by time.	7
3.2	5 connections (email transmissions) in an email system, sorted by date.	7
4.1	Edge attributes of a graph snapshot $G(d)$ from the connection log in Figure 3.1a.	16
4.2	The four features used in a profile.	19
4.3	A profile consists of three different attributes.	20
4.4	An example of calculating the advanced similarity based on common ratio and connection orders with an originator whose connected sequence: $\langle a, b, c, d, e, f \rangle$ and $\beta = 0.1$	27
5.1	MAP scores for different propagation models used in EABIF Network	32
5.2	MAP scores for each method	32
5.3	4 different combination of parameters β, γ, s_s, s_l for getting best similarities in static features and surprising features	39
5.4	3 different parameter sets for PLPF to use static features only, surprising features only and both static and surprising features	40

List of Figures

1.1	An example of the evolving graph.	2
1.2	A rough example of the connection profile derived from Figure 1.1.	3
1.3	System framework.	3
3.1	Viewing a connection log as a connection network.	9
	(a) Connection Log	9
	(b) Connection Network	9
3.2	An evolving graph consists of several graph snapshots $G(1), G(2), \dots$	10
4.1	The whole framework of PLPF.	13
5.1	Precision & Recall to rank k for each method.	33
	(a) Precision at rank k	33
	(b) Recall at rank k	33
5.2	The history data used in LastDay profiles and Instant profile.	34
5.3	MAP scores when using LastDay profiles and Instant profiles.	34
5.4	Computation time cost for a single query.	35
5.5	Tuning γ and s_l for similarity using static features only.	37
5.6	Tuning γ and s_l for similarity using static features only.	37
5.7	Tuning β and s_s for similarity using surprising features only.	38
5.8	Tuning β and s_s for similarity using surprising features only.	38
5.9	Tuning α to combine both similarities using static features and surprising features.	40

Chapter 1

Introduction

Link prediction problem in social networks has been considered as a useful and interesting research issue to capture the social activities in advance, and is widely used in other domains such as bioinformatics, computer networks and recommendation systems [8, 10, 16]. A general link prediction problem aims to predict *future links* [12, 14, 18, 22] or *unobserved links* in current graph [2, 5, 9, 22], but in this work, we focus on the links from sources to a specific destination they have never connected before as [20]. For example, when a user (the source) connect to Facebook (the destination), which is a newly appeared website, we may predict that who else will also establish connections to Facebook. Empirically, predicting such nonexistent connections in internet helps us to recommend interesting site to users [11], identify anomalous connection links [13] and prevent network attacks such as phishing [4]. To resolve this problem, users of similar connection behavior should be identified since they were candidates who will have similar behavior and are likely to connect to the same site in the future.

Previous studies model the similarity between users by calculating their common features [2, 14, 19], reachable distances [14], consistent structures [19, 24] and information transition probability [20]. However, the above works only considered the historical behavior as a static graph in a single time slot, instead of the evolution of users' behavior. The evolution of user's behavior could help us to find users' stable interests such as regular connection behavior and new interests such as newly connected sites which are only seen recently in the temporal activities.

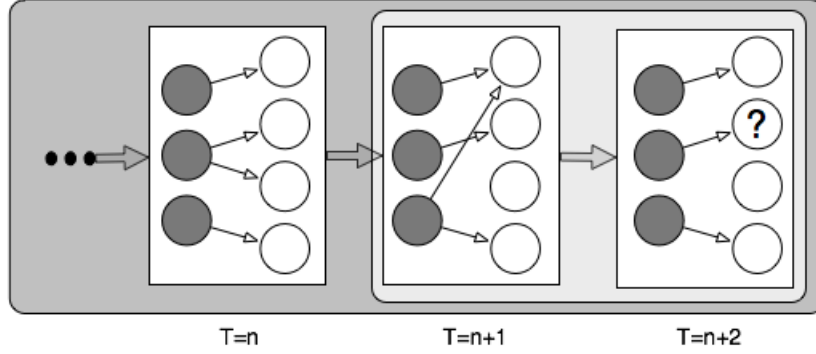


Figure 1.1: An example of the evolving graph.

In this study, we construct a connection profile for each user to record their evolving behavior. Figure 1.1 depicts the difference between previous works and ours in the scope of capturing users' historical behavior. A connection profile possesses information of connected nodes, connection order, connection count, and connection frequency as connection features. Figure 1.2 shows an rough example of the connection profile. The information attached in connection profile reflects the interests of users. The similarity of users' behavior thus is revealed by comparing users' connection profiles. However, a connection profile is a kind of semi-structured data consisting of categorical attributes (connected nodes), sequential data (connection order), and continuous values (connection count and connection frequency). Evaluating the difference between two connection profiles directly is not intuitive. In this study we use a hierarchical combination to the combine the similarities of different connection features in two connection profiles. In addition, we proposed an efficient algorithm to construct connection profile through a sliding windows manner.

Our proposed profile-based link prediction system possesses both off-line profile construction and on-line link prediction which are shown in Figure 1.3. In the off-line part, a general connection log is translated into an evolving graph of a series of consecutive graph snapshots. Then, connection profiles are constructed and updated incrementally. In the on-line part, when a query (e.g. user X connects to a new site Y) entered, we dynamically calculating the similarities between user X and other users. Finally, the top- k users most similar to user X are predicted as the successors who will

Notes:	{A, B, C, E}
Order:	<A, B, A, C, E>
Count:	(5, 7, 3, 8, 4)
Frequency:	(0.3, 0.5, 0.2, 0.4, 0.1)

Figure 1.2: A rough example of the connection profile derived from Figure 1.1.

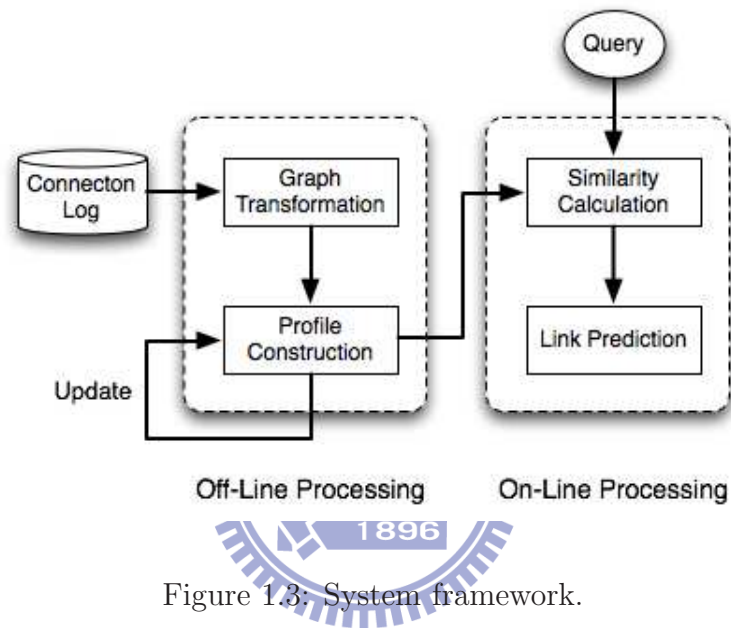


Figure 1.3: System framework.

establish connections to site Y later.

The rest of this paper is organized as follows. Chapter 2 presents the related work of link prediction. Chapter 3 gives the preliminary. In Chapter 4, the details of our method are provided. In Chapter 5, we compare our method with the most similar work driven by information flow scheme, and finally this study concludes in Chapter 6.

Chapter 2

Related Works

Researchers devoted in the problem of link prediction on various domains, such as social networks, bioinformatics, computer networks and recommendation systems [10]. The problem of link prediction first introduced and defined in [14]. Consequently, authors in [3, 1, 6, 7] proposed various algorithms to resolve the problem from a connection graph. In [19], the authors investigated the methods of link prediction into two main approaches, the generative approaches and the similarity-based approaches.

The generative approaches build a probability model to describe the historical connection behavior. Then the link could be predicted by calculating the probability from the built model. For example, the authors in [22] and [17] proposed a relational Markov Model and local conditional probability respectively to model the historical behavior. However, there are three drawbacks related to the generative approaches. First, constructing such model is costly due to the computation of each pair and combination of nodes. Second, the model can not be built online so that it will lack of freshness. Third, the connection graph is sparse and the predicted probability would be too small to have representative.

By contrast, the similarity-based approaches only need to identify users of similar behavior as the candidates who will generate similar links in the future. Thus, we calculate only the links related to the target instead of the whole connection graph. Previous studies modelled the similarity between users by calculating their common features [2, 14, 19], reachable distances [14], and consistent structures [19]. In [19], the

attributes of users are utilized to assess their similarity. The authors in [14] modelled the similarity of users as their structural proximity, such as graph distance and common neighbors. However, the above works only considered the historical behavior in a single time slot, instead of the evolution of users' behavior. Obviously, without the evolution of user's behavior, we could not describe the user's behavior of different sequential order.

To the best of our knowledge, only [20], which is a generative approach, discusses the situation of predicting links from an evolving graph. Tseng, et. al. claimed that behavior would be propagated among users according to a proposed information flow. For example, if user X and X' frequently connect to the same websites and with a specific order that X was followed by X'. They believed that there must exist an explicit information flow which causes the behavior propagating from user X to user X'. Thus, when user X connects to website Y, they would predict that user X' will also connect to website Y.

Although [20] is currently the state-of-the-art on resolving the link prediction problem, the nature of generative approach drawbacks leads to a poor efficiency and effectiveness. Therefore, in this study, we proposed a novel approach combining the evolving graph and similarity-based approach. A profile is utilized to record each user's previous behavior and calculate the similarity. When the behavior changed, only the user's profile needs to be updated instead of the probability model constructed in generative approach. Thus, we can obtain a better performance and accuracy when predicting potential links.

Chapter 3

Preliminary

In this section we will first illustrate the format of general connection logs with two examples in the real world, and show how to view a connection log as a *connection network* directly. Then we will introduce the evolving graph model we used while accessing the connection log/network in this work. Finally, we will formally define our problem and introduce the symbols and notations used in the following sections.

3.1 Connection Log & Connection Network

Here we will first illustrate the format of general connection logs with two real world instances: the world wide web (WWW) in the internet and the email systems. Then we show that any connection log can be viewed as a *connection network*, which is like a bipartite network with repeating links in the same node pairs.

3.1.1 Connection Log

Connection logs record connection behavior between *sources* and *destinations*. For example, in a web connection log, we can find out the connection behavior, such as a user may connect to Google, Yahoo!, or Facebook. Here the user is a source and each website is a destination. Table 3.1 is a short web connection log which contains five connections between two users and three websites. Similarly, sending emails from a sender to a recipient is also one type of connection behavior, where the sender is a

source and the recipient is a destination. A partial log in an email systems with five email transmissions between two senders and two recipients is revealed in Table 3.2.

Table 3.1: 5 connections in the internet network, sorted by time.

User	Website	Time
Alice	Facebook	2011-01-29 19:20:36
Bob	Yahoo!	2011-01-29 19:20:41
Bob	Google	2011-01-29 19:23:05
Bob	Facebook	2011-01-29 19:23:52
Alice	Yahoo!	2011-01-29 19:24:17

Table 3.2: 5 connections (email transmissions) in an email system, sorted by date.

Sender	Recipient	Date
zhlee@cs.nctu.edu.tw	zxliiao@cs.nctu.edu.tw	2011-01-29
hjchang@cs.nctu.edu.tw	zxliiao@cs.nctu.edu.tw	2011-01-30
zhlee@cs.nctu.edu.tw	wcpeng@cs.nctu.edu.tw	2011-01-30
hjchang@cs.nctu.edu.tw	wcpeng@cs.nctu.edu.tw	2011-01-31
zhlee@cs.nctu.edu.tw	wcpeng@cs.nctu.edu.tw	2011-01-31

As showed in the two examples, a connection consists of three attributes: **source**, **destination** and **time** to record a connection behavior representing a relationship with one-way direction from a source to a destination. Time attribute is used to identify the sequential order of records. According to the time information, we can transform a connection log into an evolving graph of several consecutive graph snapshots which can help us discover the profiles of users.

3.1.2 Connection Network

As mentioned in previous subsection, a connection consists of three attributes: source, destination and time. Therefore, an entry in a connection log can be denoted as (u, v, t) which represents a connection from the source u to the destination v at time t . By

viewing all the sources as a set U and all the destinations as another set V , then each entry (u, v, t) in a connection log can be viewed as a directed link (edge) from the source u to the destination v in a bipartite network with two different nodes sets U, V . Hence a connection log can be simply viewed as a bipartite network $G = (U, V, E)$, where U is the node set of sources, V is the node set of destinations and E is the set of all links (connections). The transferred network view of a connection log is called a **connection network**.

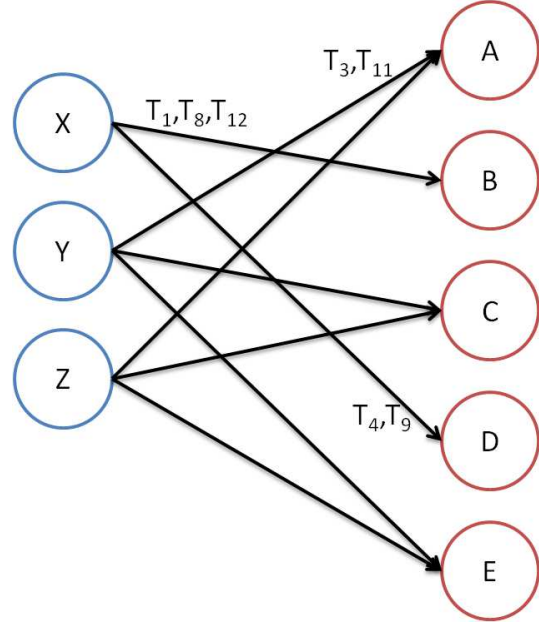
Definition 1. (*Connection Network*): A connection log can be viewed as a connection network G with three components (U, V, E) , where U, V are two different node sets consist of sources and destinations respectively, and $E = U \times V \times T$ is the link set. Each link $(u, v, t) \in E$ represents a connection from a source u to a destination v at time t in the connection log.

The connection network G of a connection log is formally defined in Definition 1. It's notable that multiple links between the same source-destination pair can coexist in a connection network G , while it's not possible in a general bipartite network. This is achieved by embedding the time dimension T into the link set E , hence $E = U \times V \times T$ and links $(u, v, t), (u, v, t')$ can coexist in the link set if $t \neq t'$. So given a general connection log as in Figure 3.1a, we can viewing it as the connection network in Figure 3.1b.

One should notice that both U, V in a connection network just stand for two sets of different node types generally, and the links (edges) in E just represents an relationship (connection behavior) between the two node sets U and V . The nodes types and relationship will be changed for different domains and applications, but for convenience we will say the first type of nodes in U as users, the second type of nodes in V as sites and the links in E as the connections that users connect to sites in the internet network in following sections.

Source	Destination	Time
X	B	T_1
Y	C	T_2
Y	A	T_3
X	D	T_4
Z	A	T_5
Y	E	T_6
Z	E	T_7
X	B	T_8
X	D	T_9
Z	C	T_{10}
Y	A	T_{11}
X	B	T_{12}

(a) Connection Log



(b) Connection Network

Figure 3.1: Viewing a connection log as a connection network.

3.2 Evolving Graph Model

An **evolving graph** is a graph consisting of several consecutive *graph snapshots*. Here we first define what a graph snapshot (of a connection network) is, then illustrate the evolving graph with an example.

Definition 2. ($G_{[t_1, t_2]} = (U_{[t_1, t_2]}, V_{[t_1, t_2]}, E_{[t_1, t_2]})$): The snapshot of a connection network G whose links (connections) are only appeared in the time interval $[t_1, t_2]$ is denoted as $G_{[t_1, t_2]}$.

A graph snapshot of the connection network G is just the connection network G in a fixed time interval, say $[t_1, t_2]$. We use the notation $G_{[t_1, t_2]} = (U_{[t_1, t_2]}, V_{[t_1, t_2]}, E_{[t_1, t_2]})$ as the graph snapshot of a connection network G whose links (connections) are only appeared in the time interval $[t_1, t_2]$. That is,

$$E_{[t_1, t_2]} = \{(u, v, t) \in E | t \in [t_1, t_2]\}, \quad (3.1)$$

and similarly,

$$U_{[t_1, t_2]} = \{u \in U | \exists (u, v, t) \in E_{[t_1, t_2]}\}, \quad (3.2)$$

$$V_{[t_1, t_2]} = \{v \in V | \exists (u, v, t) \in E_{[t_1, t_2]}\}. \quad (3.3)$$

Given the connection log in the time interval $[1, t]$, one can view it as a single static graph $G_{[1, t]}$ as the graph snapshot in time interval $[1, t]$. However, doing such thing may loss the information that how connections are evolving in the time interval $[1, t]$. A better way is viewing it as several consecutive graph snapshots in a series of consecutive time intervals with smaller lengths, say $[1, 2], [2, 3], \dots, [t - 1, t]$. Hence one can easily gain the evolving connection behavior in different time interval.

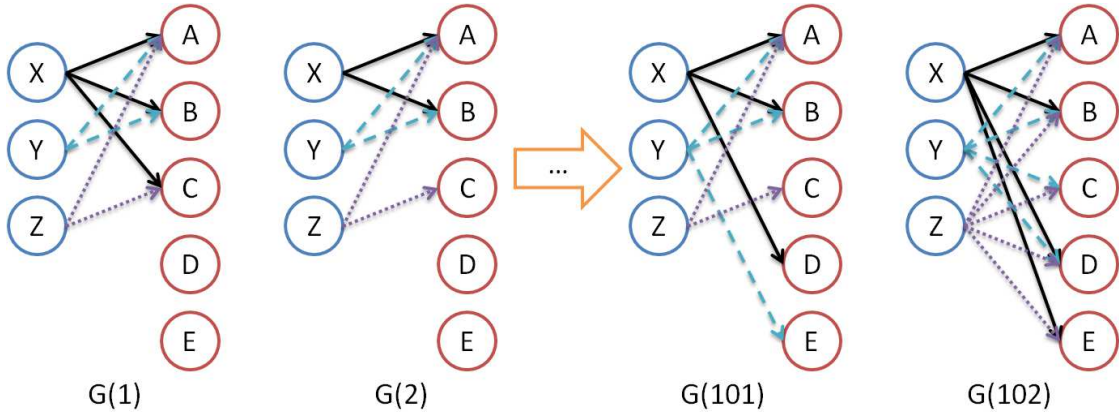


Figure 3.2: An evolving graph consists of several graph snapshots $G(1), G(2), \dots$

An example of the evolving graph in time interval $[1, 103]$ is illustrated in Figure 3.2. Each graph snapshot in Figure 3.2 denoted by $G(t)$ is the graph snapshot in the time interval $[t, t + 1]$. One can know that all three users (X,Y,Z) connect to the 5 sites (A,B,C,D,E) in the time interval $[1, 103]$ by viewing it as a static graph snapshot $G_{[1, 103]}$. But by viewing it as an evolving graph as in Figure 3.2, we can easily know the evolving of connection behaviors such as: 1) both sites D,E are only connected in the time interval $[101, 103]$, 2) the two users X,Y tends to connect sites A,B in each smaller time interval while only the user Z tends to connect A,C which is different, and 3) the user X has connected to the site C in the time interval $[1, 2]$ once but never connected it again. These things are only presented in the evolving graph model.

3.3 Problem Formulation

Definition 3. (*Link Prediction to a Specific Site v*): Given a connection network G and the link (o, v, t_1) be the first link to the site v , predict whether a link (u, v, t) to the same site v will exist in G with $u \neq o, t > t_1$.

Assuming that all users are known initially (i.e., $U = U_{[t, t']}$ for any time interval $[t, t']$) and given the whole connection log in an internet network, then the question asks: “if a user o connects to the site v , then which else user in U will likely connect to the same site v later?” The question can be formulated as a *link prediction problem to a specific node v* defined in Definition 3: given the connection network G and the first link (o, v, t_1) to the site v from a user o , predict whether a link to the same site v by another user u will exist later.

To best formulate this problem with its correct answers in a prediction time interval, we define the problem **K-Predict** using the connection network in the past time interval $[t_0, t_1]$ to predict k possible links in the future time interval $[t_1, t_2]$: given the snapshot of a connection network $G_{[t_0, t_1]}$, and the link (o, v, t_1) to be the first link to the site v , then predict k possible links $(u_1, v, t_1), (u_2, v, t_2), \dots, (u_k, v, t_k)$ in the later snapshot $G_{[t_1, t_2]}$ from k different users. The formal definition is illustrated in Definition 4.

Definition 4. (*K-Predict*): Given times t_0, t_1, t_2 along with the earlier snapshot of the network $G_{[t_0, t_1]}$ and the link (o, v, t_1) to be the first link to the site v , then predict k possible links $(u_1, v, t_1), (u_2, v, t_2), \dots, (u_k, v, t_k)$ in the later snapshot $G_{[t_1, t_2]}$ (i.e., $(u_i, v, t_i) \in E_{[t_1, t_2]}$ for $1 \leq i \leq k$), where $u_i \in U \setminus \{o\}, t_i \in [t_1, t_2]$ for all $1 \leq i \leq k$.

Chapter 4

PLPF: Profile-based Link Prediction Framework

In this section, we first take an overview of the Profile-based Link Prediction Framework (PLPF). Then we explain the graph generation process which transfers the connection log into a series of consecutive graph snapshots of the corresponding connection network in every time period. These consecutive graph snapshots are used to construct profiles and update them periodically in the off-line component of PLPF to capture connection behavior of users. Finally, we discuss the similarity measurements used to do link predictions in the on-line component of PLPF.

4.1 Framework Overview

Our major concept is that: “users with similar connection behavior before are likely to have similar connection behavior after,” which is the same as the concept of collaborative filtering (CF): “users with similar preferences are likely to access the same items later.” When a new site v appears in the connection log, there must be one user o who just connects to v and we note him/her as the *originator* to v since he/she is the first user who connects to v . Other users who connects to the same site v later are called *successors* respectively.

Definition 5. (*Originator and Successor*): Given a connection network G , the users

u_1, u_2, \dots, u_k which have connected to the site v before and the information that u_i is connected to v earlier than u_j for all $i < j$. Then the user u_1 is called the **originator** to v since he is the first who connect to v , other users u_2, u_3, \dots, u_k who connect to v later are called **successors**.

So in PLPF, given a newly appeared site v with its originator o , we aim to find possible successors through the remaining users (i.e., $U \setminus \{o\}$) which have similar connection behavior as the originator o in the earlier snapshot of the connection network $G_{[t_0, t_1]}$. The connection behavior is abstracted into a profile for each user through the connection network $G_{[t_0, t_1]}$ (i.e., the partial connection log in $[t_0, t_1]$), and similarities between users are estimated based on the similarities of profiles. However, accessing the huge connection log takes a lot of time, so we separate the framework of PLPF to two components: an off-line component which constructs/updates profiles for each user periodically, and an on-line component which accepts queries and extracts the top- k possible successors based on the similarities of pre-computed profiles in the off-line component. The whole framework of PLPF is illustrated in Figure 4.1.

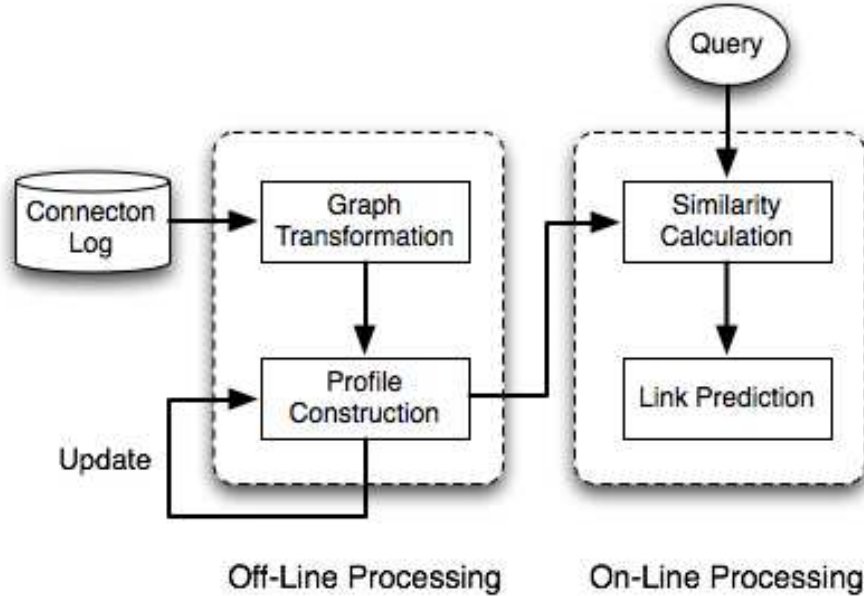


Figure 4.1: The whole framework of PLPF.

In the off-line component, we first transfer the connection logs into a series of

Algorithm 1: Link Prediction in the On-line component of PLPF

Input: the query $(o, v, t), k$
Output: the top- k possible links
1 fetch profile p_u for all users in U
2 **foreach** *User* $u \in U \setminus \{o\}$ **do**
3 calculate the similarity $s_o(u)$ according to p_o, p_u
4 **end**
5 rank users $u \in U \setminus \{o\}$ according to $s_o(u)$
6 **return** the links from the first k ranked users to v

consecutive graph snapshots of the connection network in every time period. Then, profiles containing the features for describing users connection behavior are constructed. To keep the freshness of profiles, the off-line component updates profiles incrementally with a fixed time interval. The time interval should be set according to the domain knowledge for applications. For convenience, we will use one day as the unit time interval since we set it to one day in the experiment for capturing the daily changes of users' connection behavior.

In the on-line component, when a query (i.e., a new connection to v , say (o, v, t_1)) entered, we dynamically calculate the similarities between the originator o and other users and then report the top- k users most similar to the originator o as the successors who will establish links to v later. The flow to make link predictions in the on-line component is illustrated in Algorithm 1. When a query comes, the on-line component first fetches the most recent profiles of users. Then for each user $u \in U \setminus \{o\}$, the similarity $s_o(u)$ is computed according the fetched profiles p_o, p_u . Finally, the top- k possible successors are returned as the predicted successors who will establish connections to v later.

In the following sections, we will first describe graph generation process which transfer the connection log into a series of daily graph snapshots of the connection network, and then explain what features are selected into profiles and how to construct/update profiles from the graph snapshots. Finally, the similarity measurement used to do link predictions is illustrated.

4.2 Graph Generation

As described in Section 3.1.2, a connection log can be simply viewed as a connection network by accepting the connections as links in a bipartite network. We then further transfer the connection network into a series of consecutive graph snapshot as an evolving graph. However, such links are still too many especially for internet network, where the number of connections between a user-site pair could be reached to one thousand even in a single graph snapshot. To reduce the computational cost of capturing the connection behavior for a user, we only focus on the sites connected by the user and record the attributes describing these connected sites. Therefore, two attributes are selected in each graph snapshot of the connection network:

1. **Connection count** $c(u, v)$: the number of connections made by the user u to the site v in this graph snapshot.
2. **Adoption time** $a(u, v)$: the time when the first connection to the site v is established by the user u in this graph snapshot.

Algorithm 2: Generate graph snapshot $G(d)$ from $G_{[d,d+1]}$

Input: $G_{[d,d+1]}$
Output: $G(d)$

```
1 initialize a bipartite graph  $G$  with user set and site set as  $G_{[d,d+1]}$ 
2 foreach  $link (u, v) \in G$  do
3    $c(u, v) = 0$ 
4    $a(u, v) = \text{NULL}$ 
5 end
6
7 foreach  $link (u, v, t) \in G_{[d,d+1]}$  do
8   if  $c(u, v) = 0$  then
9     update  $c(u, v) = 1, a(u, v) = t$ 
10  else
11    update  $c(u, v) = c(u, v) + 1$ 
12
13    if  $a(u, v) > t$  then
14      update  $a(u, v) = t$ 
15    end
16  end
17 end
18 return graph  $G$ 
```

The whole flow to derive a graph snapshot $G(d)$ at the d -th snapshot of the connection network $G_{[d,d+1]}$ is illustrated in Algorithm 2. The process is efficient in the

aspect of computation time since it only needs to read the connection log once. For example, given the connection log in Figure 3.1a, we can transfer it to a single graph snapshot $G(d)$ with edge attributes listed in Table 4.1.

Table 4.1: Edge attributes of a graph snapshot $G(d)$ from the connection log in Figure 3.1a.

Edge (u, v)	Adoption Time $a(u, v)$	Connection Count $c(u, v)$
(X, B)	T_1	3
(X, D)	T_4	2
(Y, A)	T_3	2
(Y, C)	T_2	1
(Y, E)	T_6	1
(Z, A)	T_5	1
(Z, C)	T_{10}	1
(Z, E)	T_7	1

For convenience, we further define the notation $V_u(d)$ as the set of all sites connected by u in the graph snapshot $G(d)$.

$$V_u(d) = \{v | c(u, v) > 0 \text{ in } G(d)\} \quad (4.1)$$

In the above example, $V_X(d) = \{B, D\}$ and $V_Y(d) = V_Z(d) = \{A, C, E\}$. This notation is used for constructing/updating the profiles later. Given the consecutive graph snapshot $G(0), G(1), \dots, G(d)$, we can build the profiles to capture the evolving connection behavior of users from day 0 to day d .

In the next section, we will first explain what features are selected, then explain the profile format, and finally illustrate the methods to construct/update profiles from the daily graph snapshots.

4.3 Profiles

A profile represents various connection features of a user to capture his/her interest from the evolving connection behavior. We will first discuss what features are selected

into the profile to effectively reflect the similar interests between users. Then the profile construction process using the consecutive graph snapshots is illustrated.

4.3.1 Connection Features

Connection features are used to capture the users' interests from the evolving connection network. Traditional link prediction methods often extract features by viewing the evolving connection network as a static network in a fixed time interval, i.e., the long-term behavior (interest). However, as mentioned in [23]: "overall behavior of a user may be determined by his/her long-term interest, but at any given time, a user is also affected by his/her short-term interest due to transient events, such as new product releases and special personal occasions such as birthdays." So here we don't only focus the *static features* that is extracted in a static network as the long-term interest, but also the *surprising features* which only appears recently and must be extracted by viewing the connection network as an evolving graph as the short-term interest.

Definition 6. (*Static features & surprising features*): **Static features** are used to represent the long-term interests which can be extracted from the static network view of the connection network in a fixed time interval. Besides, **surprising features** are used to represent the short-term interests which only appears recently and should be extracted from the evolving graph view of the connection network.

Usually the *connected sites* by a user are often used to represent the interest of that user. However, to separate the long-term interests and short-term interests, we use two sliding window to capture the connected sites in two time intervals with different lengths. The larger sliding window with size s_l is used to derive the connected sites in the last s_l days as the static features of long-term interests, and the smaller sliding window with size s_s is used to gain the *newly connected sites* only appeared in the last s_s days as the surprising features of short-term interests. Furthermore, since the number of sites connected by a user is very large, we record the *connection count* and *connection frequency* to help us identify the sites which are more interesting to the user. And for the newly connected site, we also record the order that which newly connected is connected earlier than the other by a user.

In summary, 4 different types of connection features are used to describe the long-term and/or short-term interest of users from his/her connection behavior:

1. **Connected sites** or **newly connected sites**: the sites connected by a user in a fixed time interval or “only” in a fixed time interval.
2. **Connection count** for each connected site: the number of connections made to a connected site in a fixed time interval.
3. **Connection frequency** for each connected site: how often a site is connected in a fixed time interval.
4. **Connection order** for newly connected sites: the order that which newly connected site is connected earlier than the other.

We will illustrate why the last three features are selected below.

The features of **connection count** is inspired from the recommendation systems. Since our concept is the same as collaborative filtering (CF), we first observe what features are used in CF for recommendation systems. A typical recommendation system consists of users and items along with user *feedbacks* on items. These user feedbacks are often described in real numbers, for example, using 1 or 0 to represent that a user likes or dislikes an item, or using a score from 0 to 10 to measures how this user is favorite on that item. And CF estimates that two users have similar preferences if they have similar feedbacks on the common accessed items. In connection logs, we don't have explicit user feedbacks on sites (even the connected sites), but we can estimated them through the connection counts implicitly. Considering that a site is connected by a user means that this user is interested in it, then more connections to the same site directly means that the user is more interested in it.

Connection count for each site can represent the preference of a user in a fixed time interval, but a site with many connections initially and fewer (or no) connections recently is not still interested for a user now. So we seek another feature to help use find the more stable interest for a user in a fixed time interval. Then the *regular connected sites* attract us since if a user is interested on something for a long time, he may

continuously accesses it, i.e., connects to the site continuously. A regular connected sites is gained by measuring that how often the user connects to it. Hence, we record the **connection frequency** for each connected site in a fixed time interval to know which site is regularly connected by a user.

Besides, to capture the short-term interest for a user with surprising features, we do not only consider the *newly connected sites*, but also the **connection order** of newly connected sites. The authors in [20] has said: “information will flow from earlier adopters to the late adopters, but may not flow back from late adopters to the earlier adopters.” This viewpoint inspire us that two users having the same new interest recently may have the same access sequence on new items, for example, users who bought new digital camera will then buy a tripod and many lens later if they all are interested in photography recently. The order of how user connects to the new sites can reflect his/her new interest directly. So we also use the connection order of newly connected sites as another surprising feature to describe the short-term interest of a user.

Table 4.2: The four features used in a profile.

L/S	Features	Descriptions
L	Connection count	the number of connections made to a connected site
L	Connection frequency	how often a site is connected in a fixed time period
S	Newly connected sites	the sites only connected in a recent time period
S	Connection order	the order that each newly connected site is connected

To give a conclusion, we list all the four features we selected in Table 4.2. A feature starts with an **L** is a static feature for long-term interest derived in a relatively larger sliding window, and a feature starts with an **S** is a surprising feature for the short-term interest which is derived in a relatively smaller sliding window, respectively. The larger sliding window with size s_l is used to capture the static interest for users in the last s_l days and the smaller sliding window with size s_s is used to capture the surprising interest for users in the last s_s days. Both s_l and s_s are fixed parameters in PLPF. In the following paragraphs, we will illustrate the implementation of profile format in

PLPF along with its construction/updating algorithm.

4.3.2 Profile Construction

We use 3 attributes to represent the 4 connection features – either static or surprising features – in a profile; the profile format is illustrated in Table 4.3 with an example on each attribute. The **connection count distribution** records the mapping of each connected site to the number of all connections made to the site in the last s_l days. Similarly, the **connection frequency distribution** records the mapping of each connected site to the frequency – how often a site is connected in the last s_l days. Both the connection count distribution and connection frequency distribution represent the static features for long-term interest – connection count, connection frequency – directly. Besides, the **connection sequence** records the order of the first connection to each newly connected site in the last s_s days as a sequence, and is the only attribute to represent the surprising features including the newly connected site (simply the sites in the connection sequence) and the connection order (the order of each site in the connection sequence) for short-term interest.

Table 4.3: A profile consists of three different attributes.

Notation	Attributes	Example
$CCD_u(d)$	Connection Count Distribution	$\{a : 10, b : 105, c : 99, \dots, v : c(v)\}$
$CFD_u(d)$	Connection Frequency Distribution	$\{a : 0.8, b : 0.6, c : 1.0, \dots, v : f(v)\}$
$CS_u(d)$	Connection Sequence	$\langle c', d', b', a', \dots \rangle$

Static features for the long-term interest are derived in the last s_l days. The connection count $c(v)$ for a connected site v is calculated as the number of all connections to v in the last s_l days.

$$c(v) = \text{number of connections to } v \text{ in the last } s_l \text{ days} \quad (4.2)$$

And the connection frequency $f(v)$ is calculated in a similar way, where the frequency means that how many days v is connected relatively to the last s_l days.

$$f(v) = \frac{\text{number of days where } v \text{ is connected to in the last } s_l \text{ days}}{s_l} \quad (4.3)$$

The connection sequence is simply all the newly connected sites in the last s_s days, which are ordered by their first connection time. A connection sequence can be expressed as a vector below.

$$\langle v_1, v_2, \dots, v_n \rangle \quad (4.4)$$

It's notable that v_i is connected earlier than v_j for all $i < j$ in the last s_s days.

Connection counts and connection frequencies for different sites in the last s_l days can be easily derived from the evolving graph with the last s_l consecutive daily graph snapshots. However, to derive the newly connected sites which are connected “only” in last s_s days, we cannot only look the last s_s consecutive daily graph snapshots but also other graph snapshots in earlier days is needed to know whether a site has been connected before or not. So we build an *adoption history* $H_u(d)$ for each user u to record the *first connection time* and the *last connection time* for each site that are connected in the last s_s days before day d . The adoption history $H_u(d)$ can be formatted as a vector below:

$$H_u(d) = \langle (v_1, f_1, l_1), (v_2, f_2, l_2), \dots, (v_m, f_m, l_m) \rangle, \quad (4.5)$$

where m is the number of sites connected by user u in the last s_s days before day d , and f_i (l_i) is the first (last) connection time to site v_i in the last s_s days before day d . At any day d , given the adoption history $H_u(d)$, we can derive the newly connected sites by the user u in the last s_s days before day d by simply filtering out the sites in $H_u(d)$ whose first connection time is earlier than day d with s_s days.

The adoption history $H_u(d)$ can be easily updated from the adoption history in the previous day $H_u(d-1)$ and the daily snapshot of graph $G(d)$ in the current day. The incremental updating process of an adoption history $H_u(d)$ is illustrated in Algorithm 3. It's notable that the newly connected sites may be connected by the user far before day d with s_s days. But if the user has not connected to it in the last s_s days and occasionally connects to it again now, then the site is still counted as a newly connected site since it's new in the last s_s days.

To conclude this section of the profile construction, we list the 3 construction processes for the attributes of connection count distribution, connection frequency distribution, connection sequence in Algorithm 4, 5 and 6, respectively.

Algorithm 3: Update Adoption History $H_u(d)$ from $H_u(d-1), G(d)$

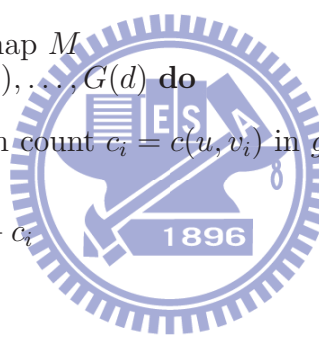
Input: $H_u(d-1), G(d)$
Output: $H_u(d)$

```
1 foreach  $v_i \in V_u(d)$  do
2   fetch the adoption time  $a_i = a(u, v_i)$  in  $G(d)$ 
3
4   if  $v_i \in H_u(d-1)$  then
5     update  $l_i = a_i$  in  $H_u(d-1)$ 
6   else
7     insert  $(v_i, a_i, a_i)$  into  $H_u(d-1)$ 
8   end
9 end
10 foreach  $(v_i, f_i, l_i) \in H_u(d-1)$  do
11   if  $(d - l_i) > s_s$  then
12     remove  $(v_i, f_i, l_i)$  from  $H_u(d-1)$ 
13   end
14 end
15 return the modified  $H_u(d-1)$  as  $H_u(d)$ 
```

Algorithm 4: Construct the attribute of connection count distribution $CCD_u(d)$

Input: $G(d-s_s+1), \dots, G(d)$
Output: $CCD_u(d)$

```
1 initialize an empty hash map  $M$ 
2 foreach  $g$  in  $G(d-s_s+1), \dots, G(d)$  do
3   foreach  $v_i \in V_u(g)$  do
4     fetch the connection count  $c_i = c(u, v_i)$  in  $g$ 
5
6     if  $v_i \in M$  then
7        $M[v_i] = M[v_i] + c_i$ 
8     else
9        $M[v_i] = c_i$ 
10    end
11  end
12 end
13 return  $M$ 
```



4.4 Similarity Calculation

The similarity between users is calculated through the similarities of their static and/or surprising features for the long-term and short-term interest separately. In this section, we first explain that how we combine two similarity values to obtain a new one. Then we describe how to calculate the similarity values of static features and surprising features separately. Finally, we give a summary of all the 3 similarities used in PLPF.

Algorithm 5: Construct the attribute of connection frequency distribution $CFD_u(d)$

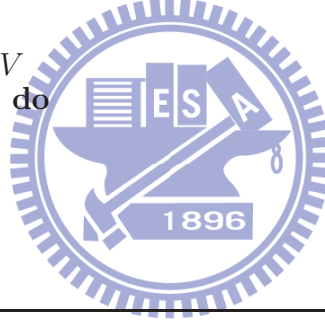
Input: $G(d - s_s + 1), \dots, G(d)$
Output: $CFD_u(d)$

- 1 initialize an empty hash map M
- 2 **foreach** g in $G(d - s_s + 1), \dots, G(t)$ **do**
- 3 **foreach** $v_i \in V_u(g)$ **do**
- 4 **if** $v_i \in M$ **then**
- 5 $M[v_i] = M[v_i] + 1$
- 6 **else**
- 7 $M[v_i] = 1$
- 8 **end**
- 9 **end**
- 10 **end**
- 11 **foreach** $M[v_i] \in M$ **do**
- 12 $M[v_i] = \frac{M[v_i]}{s_l}$
- 13 **end**
- 14 **return** M

Algorithm 6: Construct the attribute of connection sequence $CS_u(d)$

Input: $H_u(d)$
Output: $CS_u(d)$

- 1 initialize an empty vector V
- 2 **foreach** $(v_i, f_i, l_i) \in H_u(d)$ **do**
- 3 **if** $f_i > (d - s_s)$ **then**
- 4 add v_i into V
- 5 **end**
- 6 **end**
- 7 sort V according f_i
- 8 **return** V



4.4.1 Similarity Combinations

Before we explain the method to combine two similarity values, we first explain the notation of a general similarity (function) $s(\cdot)$ used in PLPF. Since the similarity in PLPF is used to inform that how a user is similar to an originator o , the similarity is often mentioned as $s_o(u)$ instead of $s(o, u)$. The notation $s_o(u)$ also tells that a similarity function in PLPF can be asymmetric, that is, $s_o(u)$ may not be equal to $s_u(o)$. Moreover, since the originator o is fixed in every single query, the similarity $s_o(u)$ can be further simplified as $s(u)$ only; the originator o can be omitted.

Given two similarity scores $s_1(u)$ and $s_2(u)$, we can use the *convex combination*

with a weight w to combine a new one $s(u)$:

$$s(u) = w * s_1(u) + (1 - w) * s_2(u), \quad (4.6)$$

where the weight w should be in $[0, 1]$. $w = 1$ gives all weight on $s_1(u)$ and $w = 0$ gives all weight on $s_2(u)$. However, the similarity values calculated by $s_1(\cdot)$ may be relatively larger or smaller than the ones calculated by $s_2(\cdot)$. So we first normalize the similarity values $s_1(u_1), s_1(u_2), \dots, s_1(u_n)$ by dividing their maximum value $s_1(u')$ before combine them with other similarity values. The normalized similarity $\bar{s}_1(u)$ is calculated as following:

$$\bar{s}_1(u) = \frac{s_1(u)}{s_1(u')}, \quad (4.7)$$

where

$$u' = \underset{u}{\operatorname{argmax}} s_1(u). \quad (4.8)$$

The algorithm to calculate the combined similarities is illustrated in Algorithm 7. Following we will discuss the similarities of static interest and surprising interest separately.

Algorithm 7: Calculate the combined similarity $s(u)$ from $s_1(u), s_2(u)$ with a weight w

Input: $s_1(u), s_2(u), w$

Output: $s(u)$

- 1 calculate the normalized similarity $\bar{s}_1(u)$ according to equation 4.7
 - 2 calculate the normalized similarity $\bar{s}_2(u)$ according to equation 4.7
 - 3 $s(u) = w * \bar{s}_1(u) + (1 - w) * \bar{s}_2(u)$
 - 4 **return** $s(u)$
-

4.4.2 Similarity of Static Features

Similarity of static features is the combined similarity through the similarities of the static features – connection count and connection frequency – separately, with a weight γ for combination.

Connection count can be derived directly from the connection count distribution, and is often represented as a mapping or two-dimension array as below:

$$\langle (v_1, c_1), (v_2, c_2), \dots, (v_m, c_m) \rangle,$$

Considering the count of non-connected sites as 0, then each user can have their connection count distribution as a one-dimension array with the unified indexes to the same sites.

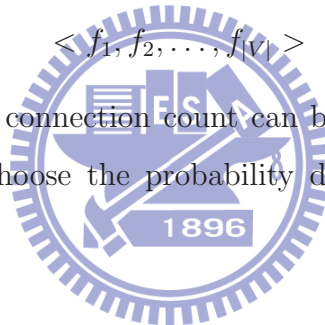
$$\langle c_1, c_2, \dots, c_{|V|} \rangle$$

This form of data are usually applied with naive vector similarities such as the *cosine similarity*. Besides, one can convert the array as a discrete probability distribution as the possibility to connected to each site, and then apply statistical distance functions like *probability distance*. In this work, we tried both and decided to use the probability distance since it performs better.

Connection frequency can be similarly derived from the connection frequency distribution, too. Also, considering the frequency of non-connected sites as 0, then each user can have their connection frequency distribution as a one-dimension array with the unified indexes as the connection count distribution.

$$\langle f_1, f_2, \dots, f_{|V|} \rangle$$

So the similarities applied to connection count can be applied to the connection frequency, too. And we also choose the probability distance for this feature since it performs better.



4.4.3 Similarity of Surprising Features

Similarity of surprising features is derived in the similar way through the similarities of surprising features – newly connected sites and connection order – separately, with a weight β for combination. However, there is a difference that the connection order is used to enhance the similarity values of newly connected sites, so we don't use the convex combination here.

The newly connected sites can be gained through the connection sequence directly by viewing the sites in the vector as a set of sites.

$$V(u) = \{v_1, v_2, \dots, v_n\}$$

Then several set-based similarity measurements can be applied to the this feature, such as the size of set intersection (*number of common items*), the *Jaccard coefficient* [21]

and so on. These similarity measurements are derived from the concept of *common items* and then weighted by other features. In other words, they count the number of common connected sites by two users recently, and gives higher similarity score if the number is larger. In here we define a new asymmetric similarity, the *common ratio* $c_o(u)$, which is modified from the Jaccard coefficient to measures how many sites connected by o is also connected by u , since we only focus on how similar a user is related to the originator.

$$c_o(u) = \frac{|V(o) \cap V(u)|}{|V(o)|} = \frac{|V(o, u)|}{|V(o)|} \quad (4.9)$$

And we used the common ratio to measure the similarity of newly connected sites.

Connection order here is used to enhance the similarity on newly connected sites. Users with common connected sites recently are said to have similar interests recently, but users with common connection order on sites are much more similar because they behave more consistently. To capture this consistency, we calculate the number of ordered site pairs which are connected by both two users with the same order:

$$pair(u_i, u_j) = |\{(v_k, v_l) | v_k, v_l \in V(u_i, u_j), v_k \text{ is connected earlier than } v_l \text{ for both } u_i, u_j\}| \quad (4.10)$$

where $V(u_i, u_j) = V(u_i) \cap V(u_j)$ is the set of common connected sites by both u_i, u_j recently. And then we define the enhanced similarity as:

$$s_o(u) = c_o(u) \cdot (1 + \beta)^{pair(o,u)} \quad (4.11)$$

where $\beta > 0$ is a tunable parameter to weight the importance of common connection orders. Table 4.4 shows an example of calculating this similarity to an originator with $\beta = 0.1$.

4.4.4 Similarities in PLPF

Given the calculated similarities of static features and surprising features, we can combine them with a weight α to derive an overall similarity using all features. Assuming the similarity of static features $s_{static}(u)$ and the similarity of surprising features

Table 4.4: An example of calculating the advanced similarity based on common ratio and connection orders with an originator whose connected sequence: $\langle a, b, c, d, e, f \rangle$ and $\beta = 0.1$

User	Connection sequence	Common Ratio	Common Pairs	Similarity
u_1	$\langle f, e, d, c, b, a \rangle$	1	0	1.0
u_2	$\langle f, x, b, c, d, y \rangle$	0.66	3	0.87846
u_3	$\langle x, a, b, c, y, z \rangle$	0.5	3	0.6655
u_4	$\langle b, a, c \rangle$	0.5	2	0.605
u_5	$\langle o, p, q, r, s, t \rangle$	0	0	0
u_6	$\langle f \rangle$	0.16	0	0.16

$s_{surprising}(u)$ are both normalized, then the overall similarity $s(u)$ is derived through the equation 4.6:

$$s(u) = \alpha * s_{static}(u) + (1 - \alpha) * s_{surprising}(u) \quad (4.12)$$

Totally we have 3 similarities can be used in PLPF:

1. similarity of static features only: set $\alpha = 1$.
2. similarity of surprising features only: set $\alpha = 0$.
3. the overall similarity using both static features and surprising features: set $\alpha \in (0, 1)$.

In Chapter 5, we will reveal the result using each similarity in PLPF.

4.5 Additional Domain Knowledge

As we mentioned above, the similarity is calculated through the connection features, which are all (evolving) topology features since we only have information about the connection log/network. But if additional domain knowledge is given, such as the node attributes, then we can use them to improve our similarity measurement and better predictions can be done.

One of such information is especially important but can not be derived from the connection log alone: the information of the newly appeared site v . As the site v just appears in the connection log, we only gain information about who connects to it and when the connection is established. But if more domain knowledge is given, for example, the newly appeared site v is very similar to another site v' which is connected by the originator o before, then we can use this information to add weight on the static (surprising) features if v' belongs to the long-term (short-term) interest of o .

Generally speaking, there are 4 different cases:

1. v is similar to the connected sites in the long-term interest of o ,
2. v is similar to the connected sites in the short-term interest of o ,
3. v is similar to the connected sites in both long-term and short-term interest of o ,
and
4. v is not similar to any connected sites in the long-term or short-term interest of o .

This information is not useful in the last case, but for the first three cases, we can derive a relative adjusting weight w_v by how similarly v belongs to the long-term interest rather than the short-term interest, where $0 < w_v < 1$. Then we can adjust the weight of long-term and short-term interests by w_v :

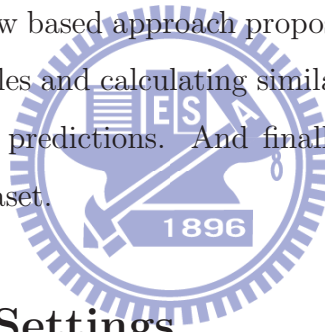
$$s_o(u) = w_v * s_o^l(u) + (1 - w_v) * s_o^s(u), \quad (4.13)$$

where $s_o^l(u)$ is the similarity of u related to o on static features only, and $s_o^s(u)$ is the similarity of u related to o on surprising features only.

Chapter 5

Experiments

In this section, we first describe the experimental environments including 1) the real datasets, 2) the query formats, and 3) the evaluating measurements. Then we show the prediction results derived by our algorithm and compare both the effectiveness and efficiency with information flow based approach proposed in [20]. The parameters used in PLPF to constructing profiles and calculating similarities are discuss later to inform how they effect the result of predictions. And finally, we analyze the properties of PLPF with the synthetic dataset.



5.1 Experiment Settings

For this work, we collect the TCP connection logs between the dormnet to the internet in our university from 2010-09-14 to 2011-01-31 as a connection log in one semester. There are more than 1,431,000,000 connections established by total 723 students and 15436399 outside sites. A set of *queries* is created for experiments with following properties:

1. Each query contains the information of the first connection to the newly appeared site. The information contains the originator (represented by an IP address), the newly appeared site (represented by an IP address) and the time to make this connection.
2. For each query, it must have 50 to 70 true successors (about $7 \sim 10\%$ of users) who

really connects to the same site in the following week after the new site appears, and these links established by these successors are used as ground truth.

Total 125 queries are selected with above properties, and have averagely 59.4 true successors in the following week after the new site appears.

Input for each method is the query (i.e., the originator, the newly appeared sever and the time of the first connection) along with the connections logs in the last day, and the output is the top- k possible links to the same site. To evaluate the effectiveness for each method, we consider the *precision/recall at rank k* and the *mean average precision (MAP)* since MAP provides a single-figure measure of quality across recall levels, which has been shown to have especially good discrimination and stability [15].

Precision at rank k for a single query q is simply the value that how many users in the top- k possible successors are true successors (which establish links to the specific site in the following week), and recall at rank k for a single query q is simply the value that how many true successors are in the top- k possible successors. Precision and recall at rank k for a query q are formalized as below:

$$precision_q(k) = \frac{|\text{number of true successors in the top-}k \text{ possible successors}|}{k} \quad (5.1)$$

$$recall_q(k) = \frac{|\text{number of true successors in the top-}k \text{ possible successors}|}{m_q} \quad (5.2)$$

The notation m_q denotes the number of all true successors for a query q . For a query set Q , the overall precision/recall at rank k is computed by the average of precision/recall at rank k for each query q in Q .

$$precision_Q(k) = \frac{1}{|Q|} \sum_{q=1}^{|Q|} precision_q(k) \quad (5.3)$$

$$recall_Q(k) = \frac{1}{|Q|} \sum_{q=1}^{|Q|} recall_q(k) \quad (5.4)$$

To compute MAP for a query set Q , we should compute the *average precision* for each query $q \in Q$ first. Average precision of a query q is the average of precision at rank k for each k from 1 to m_q , and MAP is the average of average precisions among

all queries in a query set Q . Average precision for a query q and MAP for a query set Q can be formalize as following:

$$AveragePrecision(q) = \frac{1}{m_q} \sum_{k=1}^{m_q} \{precision_q(k)\} \quad (5.5)$$

$$MAP(Q) = \frac{1}{|Q|} \sum_{q=1}^{|Q|} AveragePrecision(q) \quad (5.6)$$

Besides, to evaluate the efficiency for each method, we consider the computation time cost for each method despite the disk I/O for reading logs and reading/writing intermediate data structures. For each method, the computation time cost is cumulated from accessing the connection logs to deriving the final result of top- k possible links. This comparison is based on the computation time cost to derive the prediction result for one single query.

5.2 Comparisons with EABIF Network

In this section, we compare our method – *PLPF* – with the most similar work driven by information flow scheme – *EABIF Network* – in both effectiveness and efficiency. The comparison of precisions is performed first as operating a link prediction system using *PLPF*, which uses *LastDay Profiles* built in last day to make predictions. Then we include the most refresh data which is just before the query into profiles as *Instant Profiles*, and illustrate the difference between using *LastDay Profiles* and *Instant Profiles*. In the end, we compare the computation time cost for a single query despite the huge I/O cost for each method.

5.2.1 Precision Comparison

In the most similar work [20], their method models an information flow network called *EABIF Network*, and uses many different propagation models to compute the propagation probabilities for predicting the late adopters (possible successors). Here we list the MAP scores of the *EABIF Network* using the most two useful propagation models with 5 different parameters in Table 5.1. It is obvious that the propagation model

Table 5.1: MAP scores for different propagation models used in EABIF Network

Propagation Models	MAP Score
Exponential Weighted Summation with $\beta = 1.0$	0.208167
Exponential Weighted Summation with $\beta = 2.0$	0.199635
Exponential Weighted Summation with $\beta = 3.0$	0.197339
Exponential Weighted Summation with $\beta = 4.0$	0.196231
Exponential Weighted Summation with $\beta = 5.0$	0.195724
Summation to M Step with $M = 1$	0.229598
Summation to M Step with $M = 2$	0.228144
Summation to M Step with $M = 3$	0.225049
Summation to M Step with $M = 4$	0.221385
Summation to M Step with $M = 5$	0.218087

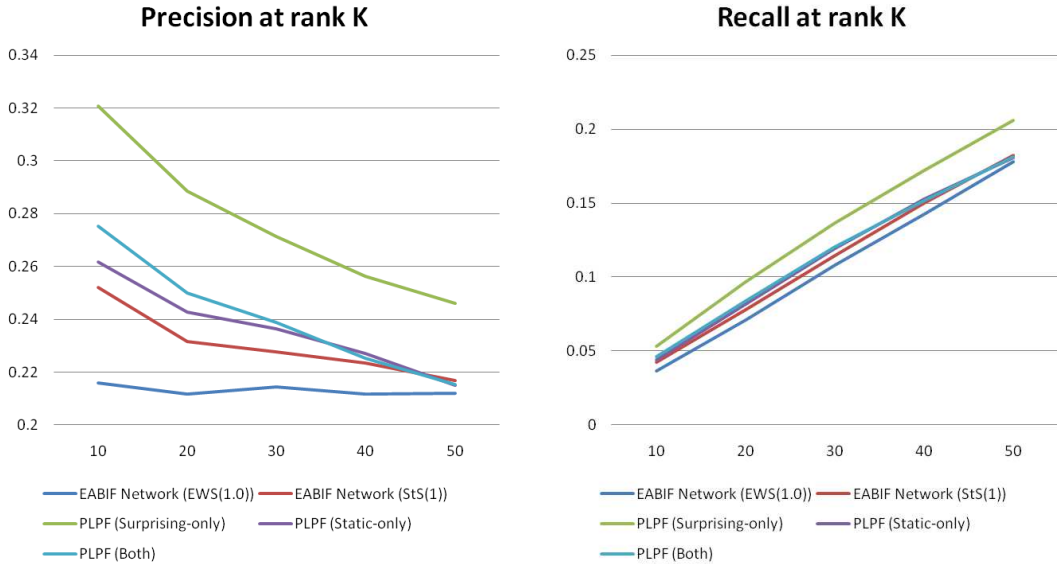
Summation to M Steps ($StS(M)$) performs better than **Exponential Weighted Summation** with weight β ($EW S(\beta)$) in all parameters. Besides, the best parameters in both propagation models show that short propagation paths are much more important than long propagation paths to predict the new links. Later we will use the propagation model $StS(1)$ in EABIF Network to compare the precisions with our methods.

Table 5.2: MAP scores for each method

Method	MAP Score	Improvement
EABIF Network ($StS(1)$)	0.229598	1
PLPF (Surprising-only)	0.279514	1.217406
PLPF (Static-only)	0.242124	1.054056
PLPF (Both)	0.248188	1.080968

We compare our method with EABIF Network in Table 5.2. Since we have two different types of features (static and surprising features for static and surprising interests), we compare the results using each type of features only and using both types of features. It shows that PLPF performs better than EABIF Network in all three combi-

nation of two different types of features, and PLPF with surprising features only gives the best MAP score 0.279514 which has the maximum improvement (21.7%) compared to EABIF Network.



(a) Precision at rank k

(b) Recall at rank k

Figure 5.1: Precision & Recall to rank k for each method.

Besides the MAP scores, the results of overall precision/recall at rank k are illustrated in figure 5.1 with $k = 10, 20, 30, 40, 50$. As the result of MAP scores, PLPF with surprising feature only has the best precision and recall values rather than other methods in each rank. All methods have a better precision value when k is smaller expect for the method of EABIF Network using propagation model $EWS(1.0)$. In addition to this, PLPF with surprising feature only give a better precision value even when k is larger ($k = 50$). This shows that surprising features (interests) are much useful to predict new links in internet.

5.2.2 Instant Profiles

All above comparisons use the history data which includes at most to the last day before the new site appears, and here we do a comparison on precisions with the most recent history data which includes the connections in the connection log just before the

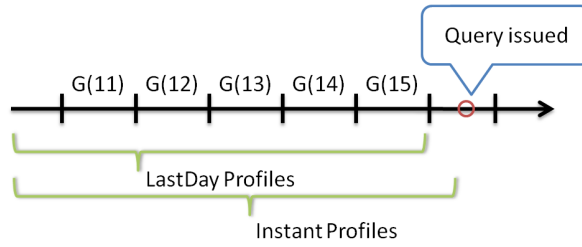


Figure 5.2: The history data used in LastDay profiles and Instant profile.

new site appears. The two types of history data are illustrated in Figure 5.2. We call the above profile a **LastDay Profile** since it uses the history data which includes at most to the last day before the new site appears, and the other profile which uses the history data including to the most recent connections just before the new site appears is called an **Instant Profile**. However, one should notice that building *Instant Profiles* needs to access the newest raw connection log directly and will cost lots of time. Hence a query using *Instant Profiles* does not have an instant result, and one should wait a long time to complete the query.

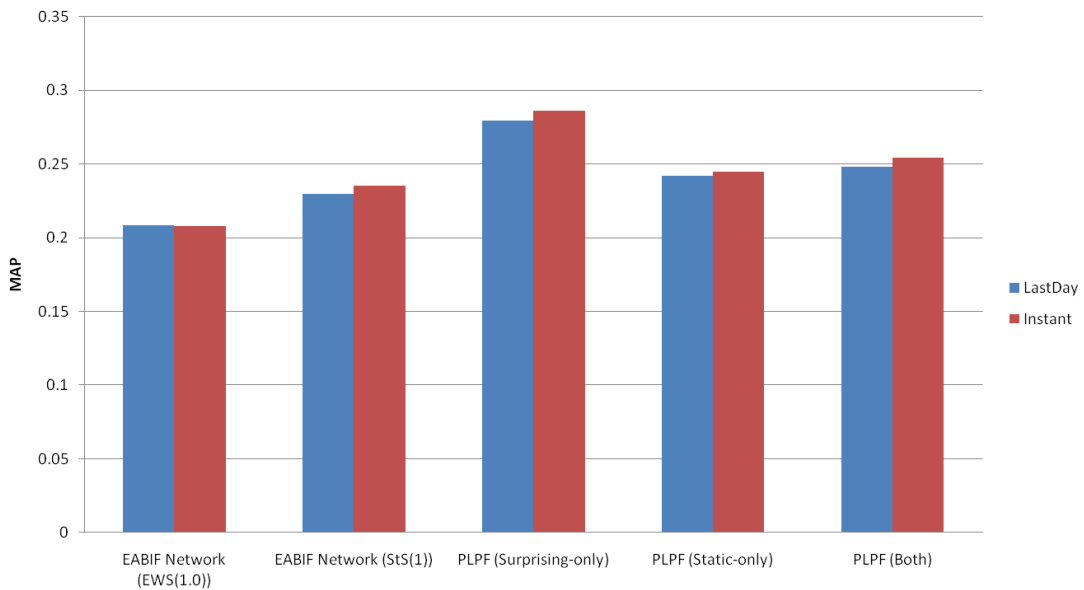


Figure 5.3: MAP scores when using LastDay profiles and Instant profiles.

The MAP scores using different profiles for each method are illustrated in Figure 5.3. It can be obviously seen that MAP score gets higher when more recent (fresh)

data are included (using *Instant Profiles*) expect the EABIF Network with *EWS(1.0)*. This again supports that surprising feature for the surprising (short-term) interest is much more important for predicting links to a newly appeared site.

5.2.3 Computation Time Comparison

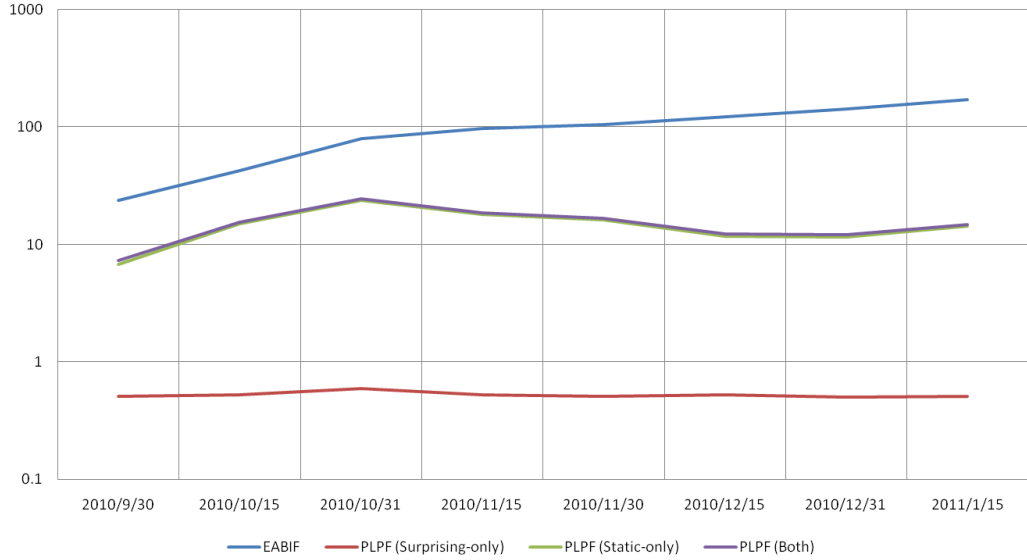


Figure 5.4: Computation time cost for a single query.

To evaluation the efficiency, we compare the computation time cost against to different length of history data we used as the input. Since we collect connection logs from 2010-09-14, we increase length of history data in half month until 2011-01-15. In figure 5.4 we compare the total computation time cost for a query from reading the connection logs to deriving the prediction result in each method. It can be seen that the computation time cost using PLPF is slightly less than using EABIF Network, and using PLPF with surprising features only (with the smaller sliding window size $s_s = 7$) gives the smallest computation time cost which is less than one second. PLPF with static features (with a larger sliding window size $s_l = 35$) has a relatively consistent time cost about 10 to 15 seconds after the history data includes more than 35 days (after 2010-10-31). The method using EABIF Network has an increasing computation time cost because it records the adoption time of each user-site pair, and the number

of user-site pairs it records increases as the length of history data increases. Besides, all methods have a slightly higher computation cost using the history data from 2010-09-14 to 2010-10-31. The reason is that the number of connections made in these days (2010-09-22 ~ 2010-10-30) is much larger than other days, hence it prolong the computation cost for every method.

5.3 Parameters in PLPF

There are total 5 parameters in PLPF: 1) s_l and s_s are two sliding window sizes used to drive static features and surprising features separately, 2) γ is used to derive the similarity for static features by combine the similarities of connection count and connection frequency. 3) β is used in the similarity of surprising features by adding weight on the common ratio, and finally 4) α is used to combine the similarities calculated with static features only and surprising features only.

In following paragraphs, we will first choose the best values of (γ, s_l) for calculating similarity using static features only. Then we choose the best values of (β, s_s) for calculating similarity using surprising features only. After all, we chose α to derive the best similarity using both static and surprising features.

5.3.1 Parameters for Static Features

First we set the value of $s_l = 7, 14, \dots, 70$ and $\gamma = 0, 0.05, \dots, 1$ to observe how the MAP score changes with different γ values. The result is illustrated in Figure 5.5. It then shows that using similarity of connection frequency only ($\gamma = 0$) gives much better MAP scores rather than using similarity of connection count only ($\gamma = 1$). $(\gamma, s_l) = (0, 35)$ gives a very high MAP score using connection frequency only. Besides, the MAP scores get higher only when γ is near to but not equal to 1.

We then zoom in for $\gamma = 0.9, 0.91, \dots, 1$, and the result is illustrated in Figure 5.6. It shows that γ value changes in 0.9 to 0.99 has little influence to MAP scores, but the larger sliding window s_l effects the MAP scores directly. $s_l = 35$ gives the best MAP scores against other values of s_l , so we choose $(\gamma, s_l) = (0.95, 35)$ as the parameter

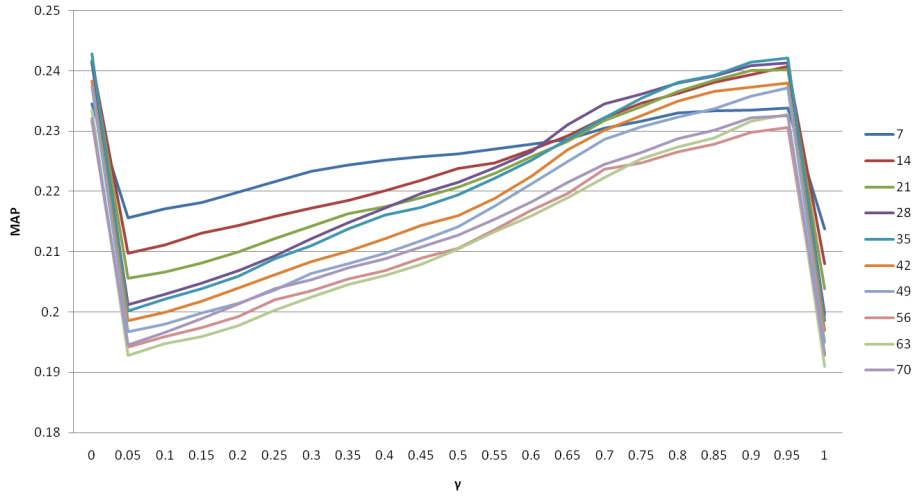


Figure 5.5: Tuning γ and s_l for similarity using static features only.

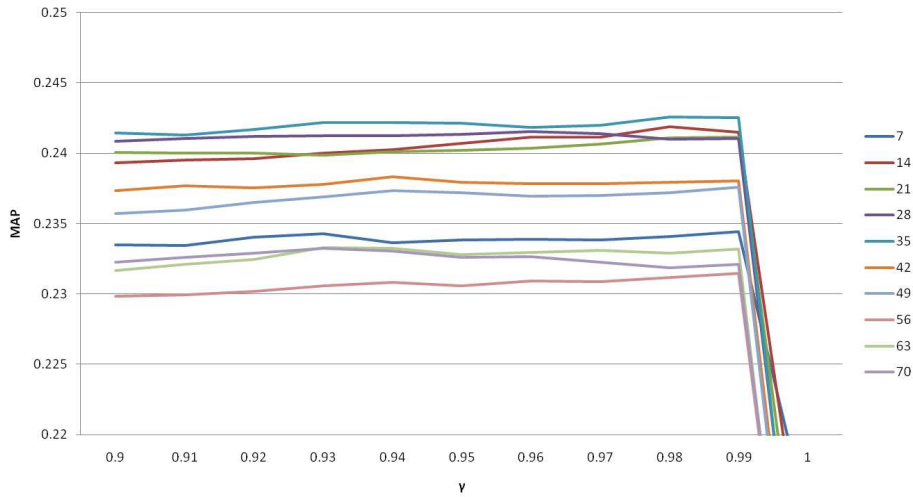


Figure 5.6: Tuning γ and s_l for similarity using static features only.

pairs using to calculate the similarity of static features using both connection count distributions and connection frequency distributions.

5.3.2 Parameters for Surprising Features

Two tunable parameters are used for surprising features: the smaller sliding window size s_s and the weight β for calculating the similarity of surprising features only. We first set β to several values from $2^{-9}(= 0.001953125)$ to $2^{-1}(= 0.5)$ and observe how

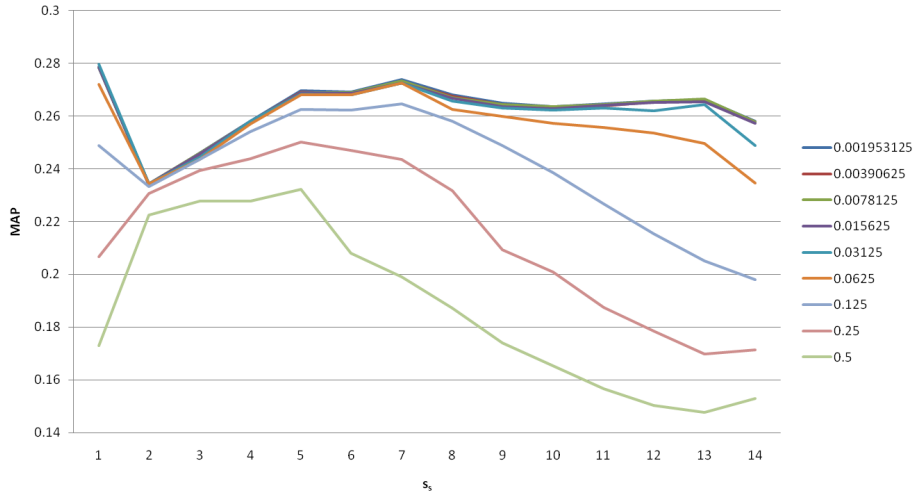


Figure 5.7: Tuning β and s_s for similarity using surprising features only.

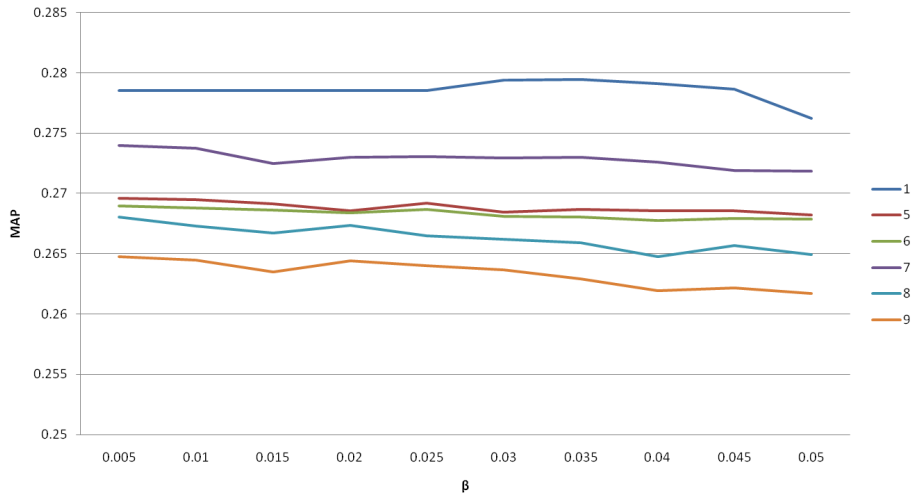


Figure 5.8: Tuning β and s_s for similarity using surprising features only.

β will effect on MAP scores as in figure 5.7. Then we found that $\beta > 2^{-4}(= 0.0625)$ makes MAP score worse, and β values in range $(0.002, 0.03)$ give almost the same effects. Besides, MAP scores are good only for the sliding window size 1, 5-9 (if $s_s > 9$ then MAP score decreases; the only exception is $s_s = 13$), and have the very worst value for $s_s = 2$.

Then we focus the smaller sliding window size s_s to 1, 5-9 and let β be values of 0.005, 0.01, \dots , 0.05. The result is illustrated in Figure 5.8. We found that $s_s = 1$

gives the best MAP score with any β values. However, since $s_s = 1$ is a special case to view every new site connected in previous day as a newly connected site, we also select another sliding window size $s_s = 7$ which gives the second good MAP scores. In the end, two different pairs of parameters (β, s_s) are chosen: $(0.035, 1)$ and $(0.005, 7)$. Both of them give the best MAP score when fixed the smaller sliding window size s_s .

5.3.3 Parameters for Both Features

Table 5.3: 4 different combination of parameters β, γ, s_s, s_l for getting best similarities in static features and surprising features

	Static Features (γ, s_l)	Surprising Features (β, s_s)
A	(0.95, 35)	(0.005, 7)
B	(0.95, 35)	(0.035, 1)
C	(0, 35)	(0.005, 7)
D	(0, 35)	(0.035, 1)

Since we have derived 2 parameter pairs for calculating the similarities using static features (γ, s_l) and surprising features (β, s_s) separately, there are total 4 different combinations of parameters β, γ, s_s and s_l . Those 4 different combinations are listed in Table 5.3.

Set $\alpha = 0, 0.05, \dots, 1$ and the result of MAP scores for the 4 different parameter combinations is illustrated in Figure 5.9. Even it is obviously seen that using surprising features only ($\alpha = 1$) brings the best MAP scores, but we still need a parameter set using all features in this experiment to compare the influences. So we choose the $\alpha = 0.85$ with parameter combination *B*, which has the highest MAP score while using all static features and surprising features. The final parameter sets for PLPF is illustrated in Table 5.4.

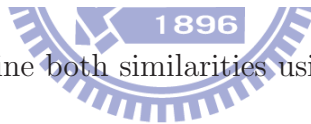
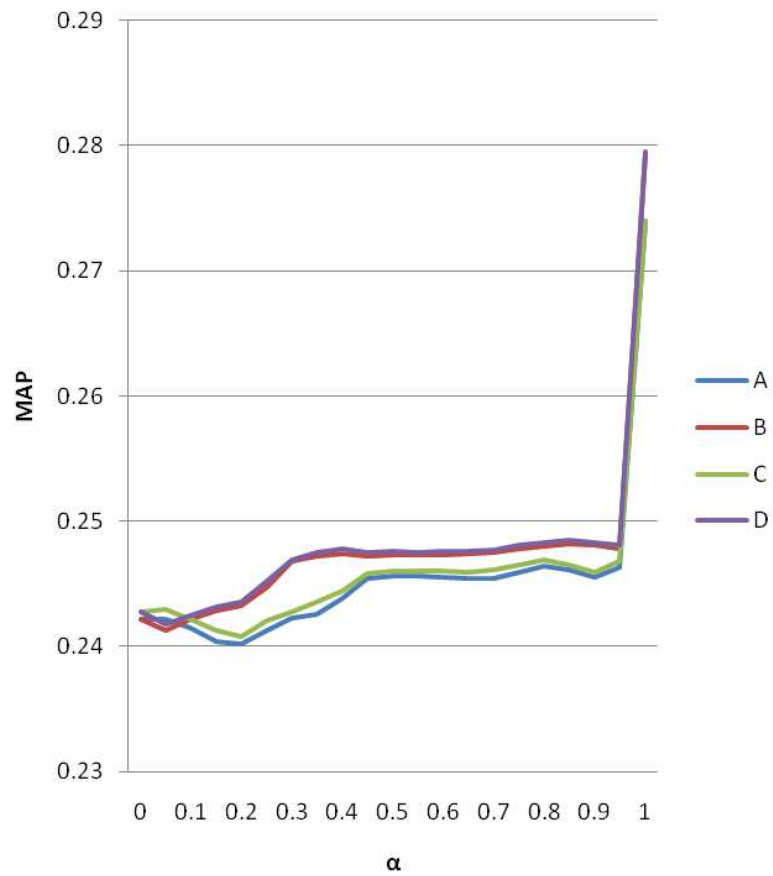


Figure 5.9: Tuning α to combine both similarities using static features and surprising features.

Table 5.4: 3 different parameter sets for PLPF to use static features only, surprising features only and both static and surprising features

Parameter	PLPF	PLPF (Static-only)	PLPF (Surprising-only)
α	0.85	0	1
β	0.035	X	0.035
γ	0.95	0.95	X
s_s	1	X	1
s_l	35	35	X

Chapter 6

Conclusion and Future Work

In this paper, we propose a framework – PLPF – to do link predictions based on profiles. PLPF consists of two components: 1) an off-line component which converts the connection log into an evolving graph of several consecutive graph snapshots and then constructs/updates user profiles with features extracted from the evolving graph periodically, and 2) an on-line component which uses the profiles to do link prediction for the top- k possible links from k different users who never connect to the specific site before. Four different type of connection features are used in the profiles to capture the connection behavior of users. In addition to the connection count which is widely used in any traditional method, we also bring up other features such as the connection frequency, newly connected sites and common connection order on the newly connected sites, which can only be derived by the evolving graph view of connection network.

In the experiment, we compare our method to the state-of-the-art method – EABIF Network – proposed by Tseng et. al. [20] in a real dataset of internet connections. In effectiveness, PLPF performs better than EABIF Network when using either static features or surprising features, and PLPF with surprising features only gives the maximum improvement of 21.7% while comparing to EABIF Network with its best propagation model. In efficiency, PLPF shows a consistent computation time cost rather than the increasing computation time cost of EABIF Network, which is caused by recording the old information which should be faded away as time evolving. Comparing to PLPF with different types of features, PLPF with surprising features only always performs

better than with static features only. It shows that user connects to new sites in the internet based on his/her short-term interest (represented by surprising features) much more than the long-term interest (represented by static features).

The future work is to dynamically adjust the lengths of sliding window for each user. In this work, we fix two sliding windows of different lengths (s_l, s_s) to capture the static features as long-term interests and surprising features as short-term interests for all users. However, for different users, the length of sliding window to capture long/short-term interests may be different or even dynamic as time evolving. Capturing users' interest in the correct sliding window can reflect the users' connection behavior more precisely, hence enhance the profiles content and improve the prediction result.



Bibliography

- [1] Evrim Acar, Daniel M. Dunlavy, and Tamara G. Kolda. Link prediction on evolving data using matrix and tensor factorizations. *Data Mining Workshops, International Conference on*, 0:262–269, 2009.
- [2] Lada A. Adamic and Eytan Adar. Friends and neighbors on the web. *SOCIAL NETWORKS*, 25:211–230, 2001.
- [3] Lars Backstrom and Jure Leskovec. Supervised random walks: predicting and recommending links in social networks. In *Proceedings of the fourth ACM international conference on Web search and data mining*, WSDM '11, pages 635–644, New York, NY, USA, 2011. ACM.
- [4] Mohamad Badra, Samer El-Sawda, and Ibrahim Hajjeh. Phishing attacks and solutions. In *Proceedings of the 3rd international conference on Mobile multimedia communications*, MobiMedia '07, pages 42:1–42:6, ICST, Brussels, Belgium, Belgium, 2007. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).
- [5] Aaron Clauset, Cristopher Moore, and M. E. J. Newman. Hierarchical structure and the prediction of missing links in networks. *Nature*, 453(7191):98–101, May 2008.
- [6] Dongsheng Duan, Yuhua Li, Yanan Jin, and Zhengding Lu. Community mining on dynamic weighted directed graphs. In *Proceeding of the 1st ACM international workshop on Complex networks meet information & knowledge management*, CNIKM '09, pages 11–18, New York, NY, USA, 2009. ACM.

- [7] Daniel M. Dunlavy, Tamara G. Kolda, and Evrim Acar. Temporal link prediction using matrix and tensor factorizations. *ACM Trans. Knowl. Discov. Data*, 5:10:1–10:27, February 2011.
- [8] Lise Getoor and Christopher P. Diehl. Link mining: a survey. *SIGKDD Explor. Newsl.*, 7:3–12, December 2005.
- [9] Anna Goldenberg, Jeremy Kubica, and Paul Komarek. A comparison of statistical and machine learning algorithms on the task of link completion. In *In KDD Workshop on Link Analysis for Detecting Complex Behavior*, page 8, 2003.
- [10] Mohammad Al Hasan and Mohammed J. Zaki. A survey of link prediction in social networks. In Charu C. Aggarwal, editor, *Social Network Data Analytics*, pages 243–275. Springer US, 2011.
- [11] Zan Huang, Xin Li, and Hsinchun Chen. Link prediction approach to collaborative filtering. In *Proceedings of the 5th ACM/IEEE-CS joint conference on Digital libraries*, JCDL '05, pages 141–142, New York, NY, USA, 2005. ACM.
- [12] Zan Huang and Dennis K. J. Lin. The time-series link prediction problem with applications in communication surveillance. *INFORMS J. on Computing*, 21:286–303, April 2009.
- [13] Zan Huang and D.D. Zeng. A link prediction approach to anomalous email detection. In *Systems, Man and Cybernetics, 2006. SMC '06. IEEE International Conference on*, volume 2, pages 1131–1136, oct. 2006.
- [14] David Liben-Nowell and Jon Kleinberg. The link prediction problem for social networks. In *Proceedings of the twelfth international conference on Information and knowledge management*, CIKM '03, pages 556–559, New York, NY, USA, 2003. ACM.
- [15] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

- [16] Galileo Mark Namata, Hossam Sharara, and Lise Getoor. A survey of link mining tasks for analyzing noisy and incomplete networks. In Jiawei Han Philip S. S. Yu and Christos Faloutsos, editors, *Link Mining: Models, Algorithms, and Applications*. Springer, 2010.
- [17] Joshua O'Madadhain, Jon Hutchins, and Padhraic Smyth. Prediction and ranking algorithms for event-based network data. *SIGKDD Explor. Newsl.*, 7:23–30, December 2005.
- [18] Alexandrin Popescul, Rin Popescul, and Lyle H. Ungar. Statistical relational learning for link prediction, 2003.
- [19] J. Scripps, Pang-Ning Tan, Feilong Chen, and A.-H. Esfahanian. A matrix alignment approach for link prediction. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pages 1–4, dec. 2008.
- [20] Xiaodan Song, Belle L. Tseng, Ching-Yung Lin, and Ming-Ting Sun. Personalized recommendation driven by information flow. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '06, pages 509–516, New York, NY, USA, 2006. ACM.
- [21] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining, (First Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.
- [22] Ben Taskar, Ming fai Wong, Pieter Abbeel, and Daphne Koller. Link prediction in relational data. In *in Neural Information Processing Systems*, 2003.
- [23] Liang Xiang, Quan Yuan, Shiwan Zhao, Li Chen, Xiatian Zhang, Qing Yang, and Jimeng Sun. Temporal recommendation on graphs via long- and short-term preference fusion. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '10, pages 723–732, New York, NY, USA, 2010. ACM.

- [24] Haiyuan Yu, Alberto Paccanaro, Valery Trifonov, and Mark Gerstein. Predicting interactions in protein networks by completing defective cliques. *Bioinformatics*, 22:823–829, 2006.

