國立臺灣大學電機資訊學院電機工程學研究所

碩士論文

Graduate Institute of Electrical Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master Thesis

社群網路中群體演變的偵測及預測

Community Evolution Detection and Prediction in Online Social Network

陳昶亦

Chang-Yi Chen

指導教授：廖婉君 博士

Advisor: Wanjiun Liao, Ph.D.

中華民國 105 年 7 月

July 2016

doi:10.6342/NTU201601504

國立臺灣大學碩士學位論文
口試委員會審定書

社群網路中群體演變的偵測及預測
Community Evolution Detection and Prediction in Online Social Network

本論文係陳昶亦君（學號 R03921031）在國立臺灣大學電機工程學系完成之碩士學位論文，於民國一百零五年七月二十二日承下列考試委員審查通過及口試及格，特此證明。

口試委員：　廖婉君　（簽名）

（指導教授）

張正尚

陳和麟　　李宏毅

系主任　劉志文　（簽名）

doi:10.6342/NTU201601504

誌謝

這篇論文的完成，要感謝許多人的幫助。在整個碩士的兩年時光中，我最要感謝的，是我的指導教授廖婉君教授，以及共同指導的張正尚教授。正是因為他們的提攜與提拔，我才能開始了解如何做研究。他們投注了許的的心血，培養每一位學生，不僅讓帶領我們進入了研究領域，也給我了很多，學術以外的知識，譬如做事的方法，以及人生道理哲理。十分感謝陳和麟教授和李宏毅教授擔任我的口試委員，給我許多論文上的建議。我也要特別感謝我的學長林維栩，我的論文有很大一部份，是以他的演算法為基礎的，我能完成這篇論文，是因為我站在他的肩膀上，希望我有把他的演算法充分發揮，達到效果，並且更上層樓。

很感謝陪伴我度過兩年研究所時光的伯翰、繼唐、彥增、以及柏璇，還有實驗室的學長學姊們。從進入研究所開始，這兩年的時光過的精彩，偶爾在實驗室的談天，一起去過的苗栗大湖，還有冬至的吃湯圓活動，我永遠記得。在碩士最後的一學期，我與伯翰、繼唐、彥增、以及柏璇一同經歷了許多辛苦艱難的挑戰，我們互相扶持，這一段的時光，令我印象深刻。

我要感謝我的女朋友顏于娟。她一直在我身旁陪伴我，陪我走過大學，走過研究所。我每一個感動的時刻，都有妳在我身邊。有很多事都麻煩了妳，不管是論文的修改，或是口試的練習，我的研究因為有妳的幫助，才有了個完美的成果。對妳的感謝，是無法在這裡以三言兩語說完的，我未來繼續說給妳聽吧！

最後，這部分的感謝要留給家人，永遠的避風港，永遠最溫暖的地方，因為有我的家人，我才能走到現在，而我會不斷向前，感謝一路上你們最真摯的陪伴。

中文摘要

在社群網路當中，有相同特性，或是連結較緊密的人，會形成社群網路當中的群體。有非常多的論文都在探討如何準確找出網路當中的群體。由於社群網路會隨時間演進而改變，網路當中的群體組成也是日新月異，如何去偵測群體的演變，已經成為了社群網路分析當中的新議題。有越來越多的論文在研究新的方法，去偵測或追蹤群體的演變。但我們想要更進一步，不僅要準確預測群體的演化，還要預測群體會如何演化。這是一個嶄新的研究，我們決定使用林維栩的演算法(Long-term Evolution Method)去偵測群體的演化，首先利用一個自己設計的人造網路，來檢驗他的演算法是否正確，以及分析他的演算法的結果。接下來我們將他的演算法用在 Facebook 和 DBLP，同時套用我們自行定義的群體演化模式，去偵測這些資料的社群和演化，並且我們更進一步去討論 Facebook 的群體演變，發現群體的演變能夠反映到發生在 Facebook 的事件上。最後，我們利用 Libsvm，從我們的資料中抽取足夠的 features，去建立預測的模型。預測的結果顯示，我們所建立的模型有非常好的表現，而我們所抽取的 feature 也的確達到良好的效果。此外，我們還比較了 SGCI 演算法的預測結果，SGCI 為一偵測群體演化的演算法，比較的結果顯示出我們所定義的演化模式，以及我們所用的演算法，比 SGCI 更加準確和有代表性。

關鍵字：社群網路分析，群體，群體演化，偵測群體演化，預測群體演化，林維栩，SGCI

doi:10.6342/NTU201601504

ABSTRACT

In a social network, a group in which people are similar to each other or have tight connections form a community. There are many works trying to precisely detect communities in a social network. As the social network change from time to time, the communities in the social network keep changing. Detecting community's evolution become a new topic in social network analysis. More and more papers are about finding new algorithms to detect or track communities' evolution. But, we want to go further in this topic. We want to not only detect communities' evolution but predict the communities' evolution. This is a brand new problem, and we decide to take advantage of Weiux Lin's algorithm (Long-term Evolution Method) to detect the communities' evolution. We first generate our synthetic data to verify and analysis the algorithm. We apply Weiux's algorithm on DBLP and Facebook dataset, and use our own defined evolution types to analysis the community evolution. We deeply discuss the community evolution in Facebook dataset, and find that the communities' evolution can match to the events happening in Facebook. Finally, we use Libsvm, select enough features from our detecting data, and build a prediction model. The prediction result shows that our model performs pretty well, and the feature we selected help a lot to our prediction

doi:10.6342/NTU201601504

model. Besides, we compare our result with SGCI algorithm, which is a community evolution detection algorithm. The comparison shows the evolution types we defined and the algorithm we used are more accurate and more representative than SGCI.

Keywords: social network analysis, community, community evolution, community evolution detection, community evolution prediction, Weiux Lin, SGCI

doi:10.6342/NTU201601504

CONTENTS

doi:10.6342/NTU201601504

doi:10.6342/NTU201601504

doi:10.6342/NTU201601504

List of Figures

doi:10.6342/NTU201601504

doi:10.6342/NTU201601504

List of Tables

doi:10.6342/NTU201601504

doi:10.6342/NTU201601504

Chapter 1
Introduction

Social network has kept growing its popularity. It's common to see more and more people join social media. People spend their time on Facebook, Twitter, and other online social media. Take Facebook as an example. Facebook started from an online social network for students in Harvard University, and for only 12 years it becomes a huge and famous online social network with 1.65 billion monthly active users. The growth of online social media such as Facebook is so rapid that many works are motivated to analyze social network. Social network raises many interesting issues to be investigated including how to detect communities in a social network, how to predict a user's preference, how to spread a message fast among users, and etc. To address these issues, some applications, like recommendation system or some strategies of advertise

doi:10.6342/NTU201601504

products, are introduced and implemented.

In the analysis of online social network, finding community structure is a well-known problem. With high concentrations of edges within special groups of nodes and low concentrations between these groups, this structure is called “community structure [1]”. There are already many studies conducting how to detect communities. In [2], the authors summarize many community detection algorithms.

The connections in social network are always changing, so the communities in network is also changing from time to time. A community in present timeslot may be merged with other communities and forms a new community in next timeslot. In other cases, a community may split into multiple new communities in next timeslot. The change of communities is called the evolution of community. Community evolution can be used in many ways. For companies, they can track the evolution of a product’s users to realize how to adjust their strategy of advertisement. If an advertisement or marketing strategy increases number of users and what’s more, attracts the users of other products to use the services, it’s clear that the advertising or strategy is useful. As a result, detecting community evolution is a growing topic in the researches of social network. For example, in [8], it provide a method called SGCI: stable group change identification.

doi:10.6342/NTU201601504

It is an algorithm to detect the communities' change. The word "stable" means by default it focus on communities which live more than two timeslots.

Considering the potential usage of the community evolution detection, we are highly interested in addressing the issues of community evolution prediction. It is a brand new topic. If we are able to predict the community evolution, we can do more than detecting the community evolution. For example, we predict that our costumer will split into two groups. One will keep using our product, and the other will change to use other company's product. If we know this information, we can find a solution to prevent this happened. We suggest that the community evolution is predictable, and propose a method to efficiently predict the community evolution. From now, there are a few papers talk about community evolution prediction. In [10], it selecs features to build a classification model, and predict whether a community will live or not. In [11], they use SGCI and GED as the basic algorithm, and again select features to build a classification model.

To achieve a good community evolution prediction, it is required to have a great community evolution detection algorithm. Weiux Lin who is one of our lab provide an algorithm for community evolution detection [3]. The algorithm shows its robustness

doi:10.6342/NTU201601504

on real world dataset. So, in this work, we use Weiux's algorithm for evolution detection. However, Weiux's algorithm is not fully verified because the real world dataset does not contain ground truth of evolution. Before we use this algorithm, we must first verify his algorithm under a fair condition. For example, we can build an artificial graphs which contain evolution ground truth to test Weiux's algorithm.

In this work, we first verify Weiux's algorithm by synthetic dataset, and analysis his algorithm. We provide a method to generate synthetic for proving the algorithm. Then, we apply the Weiux's algorithm on real world dataset. We change the detection result with our own defined evolution types, and shows how we detect events happened in Facebook dataset by analysis the evolution types. After detection on real world datasets, we select features and build a prediction model to predict the evolution types. We propose a process of selecting feature, and prove that the features we selected help us predict the evolution. And, the most important, we show that the community evolution can be detected.

The remainder of the thesis is organized as follows. Chapter 2 provides a detail introduction of Weiux' algorithm and the SGCI algorithm. Chapter 3 elaborates our method for generating synthetic, our definition or evolution types, how we transfer

doi:10.6342/NTU201601504

evolution chains to evolution types, and how we predict community evolution types. Chapter 4 is the experiment of synthetic data. We verify Weiux's algorithm and deeply analyze the algorithm in Chapter 4. Chapter 5 contains the experiment of real world datasets, which are DBLP and Facebook dataset. We provide a case study on Facebook in Chapter 5. In Chapter 6, we do the community evolution prediction on DBLP and Facebook dataset, and shows Weiux's algorithm and our method are better than SGCI. Finally, we conclude the findings and contributions of the work in Chapter 7.

Chapter 2
Related Work

In this chapter, we are going to introduce the community evolution detection we will used as our based algorithm and other algorithm we are going to compare with in the detection and prediction experiment. Community evolution is a growing topic in social network analysis. There are many works provide their community evolution detection algorithm. Weiux Lin provide his own algorithm of community evolution detection. We decide to use Weiux's algorithm as our based algorithm. We will first introduce his works about community evolution detection. Also, we introduce SGCI, which is another algorithm for community evolution detection, since we will compare our result with SGCI in detection and prediction.

2.1 Weiux’s Short-term Evolution Method

We first introduce Weiux’s Short-term Evolution Method. It is a method to detect community evolution in two graphs in adjacent timeslots. Weiux proposed two kinds of evolution detection algorithm. One is his Short-term Evolution Method, and the other is his Long-term Evolution Method. We are going to use his Long-term Evolution Method as our main algorithm for detection and prediction. Before we talk about the Long-Term Evolution Method, we should introduce the Short-term Evolution Method since it is the base of the Long-Term Evolution Method. Given two graphs in adjacent timeslots, Weiux’s Short-term Evolution can detect the evolution of communities in these graphs. The results of his algorithm are sequences of communities, which in this work we called it “community evolution chains”. A community evolution chain contains many communities in different timeslots. If there are two communities in different graphs are in the same evolution chain, the community in the graph at present timeslot is evolved from the other community in the graph at previous timeslot. Here we describe the notations and the official definition of community evolution chain. $G(t)$ is the graph at timeslot t, and $C_t(i)$ is the i-th community in $G(t)$. The definition of community evolution chain is below, which T is the maximum number

doi:10.6342/NTU201601504

of timeslot.

$$Chain_j = \{C_t(k) | t < T, k < \text{Number of communities in } G(t)\}$$

Weiux's Short-term Evolution Method takes two graphs as input. It will first apply community detection to graphs, and then decide the evolution between communities in different graphs. The following is the detail of the process of Short-term Evolution Method.

Step1 Apply community detection on two graphs. As this method is doing community evolution detection, we should get communities from graphs first. We will use any community detection algorithm to get communities. Figure 2-1 below illustrate the process in this step.

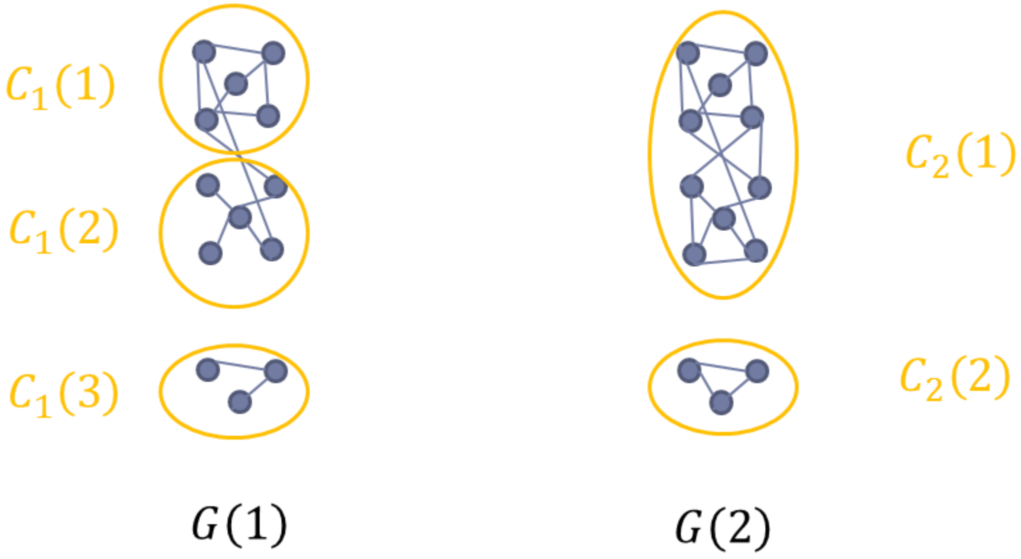

Figure 2-1 Step 1 of Short-term Evolution Method

Step2 Calculate the similarity of communities between two graphs. If the

doi:10.6342/NTU201601504

similarity is larger than threshold, add an edge between the communities. After we detect the communities in two graphs, we will calculate the similarity between communities in different timeslot. If two communities in the adjacent graphs are very similar, the community in the graph of latest timeslot may be evolved from the other community in the graph of previous timeslot. In this step, there are two arguments in the algorithm, p-core and threshold. When we calculate communities' similarity, we may not need to compare all the nodes in communities. Instead, the algorithm calculates the similarity of two communities only considering the important nodes in two communities. P-core is the argument to control how many important nodes we need to consider. The following is the definition of p-core from Weiux's paper. Consider a set of nodes $S = \{v1, v2, . . . , vn\}$ in a graph G. Suppose that $deg(vi) \geq deg(vi+1)$ for $1 \leq i \leq n - 1$, where deg(vi) is be the degree of node vi. For $0 \leq p \leq 1$, let n(p) be the smallest integer such that more than np nodes in S are connected by the set of nodes $\{v1, . . . , vn(p)\}$. The set of nodes $\{v1, . . . , vn(p)\}$ is called the p-core of S. Threshold is another argument. If the similarity of two communities is larger than threshold, the argument will consider they have some relationship of evolution and add a link between the communities. Figure 2-2 illustrate the Step2.

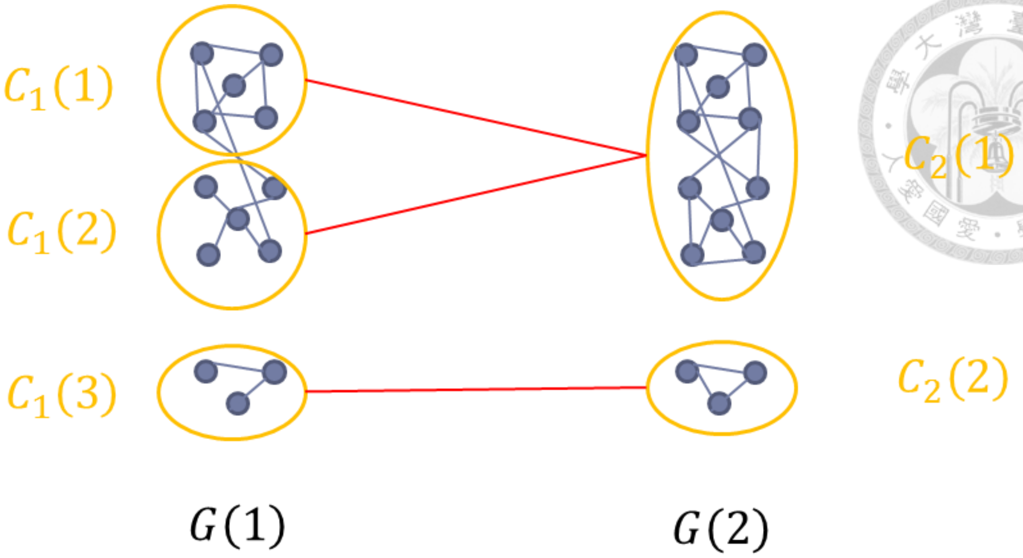

Figure 2-2 Step 2 of Short-term Evolution Method

Step3 Generate a bipartite graph, which each node in the graph is a community. In this step, we will construct a bipartite graph. Each node in this graph is the communities in the original two graph. The edges between nodes in this new bipartite graph are the links set in Step2. The following Figure 2-3 is the illustration of Step3.

doi:10.6342/NTU201601504

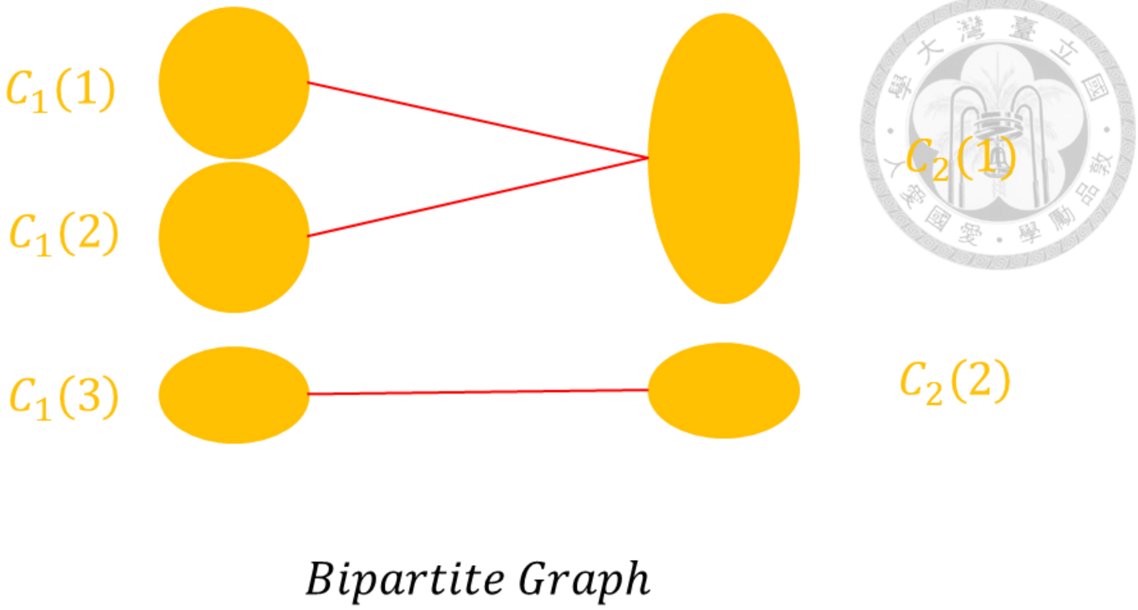

Figure 2-3 Step 3 of Short-term Evolution Method

Step4 Apply the community detection on the bipartite graph. The results are the evolution chains. This step is the most important step in Weiux's algorithm. The result of community detection on this bipartite graph is the community evolution chain because nodes in this bipartite graph is the communities in original two graph. The nodes in same a community in this bipartite graph means these nodes, which were communities in original two graphs, are very close with each other. As a result, the communities in a community evolution chain means they have relationship in evolution. Figure 2-4 illustrate the Step4.

doi:10.6342/NTU201601504

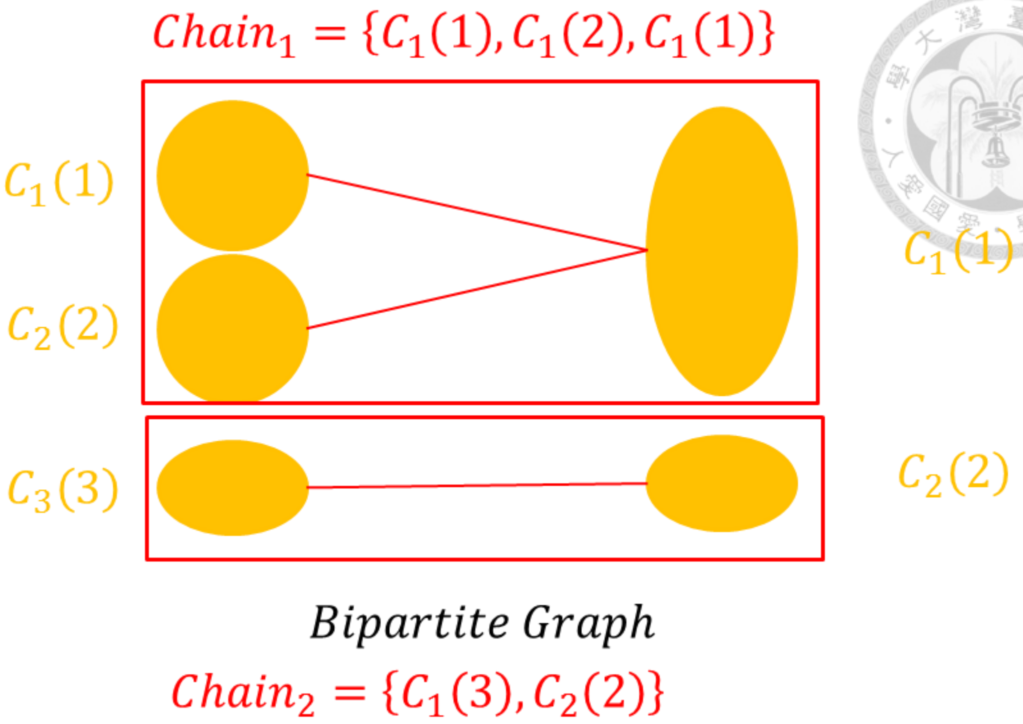
$Chain_1 = \{C_1(1), C_1(2), C_1(1)\}$
$C_1(1)$
$C_2(2)$
$C_1(1)$
$C_3(3)$
$C_2(2)$
$Bipartite\ Graph$
$Chain_2 = \{C_1(3), C_2(2)\}$

Figure 2-4 Step 4 of Short-term Evolution Method

In Weiux's Short-term Evolution Method, he also provide his definition of evolution types. However, we will provide our own definition of evolution types and a method to transfer the evolution chain to evolution types. As a result, we will talk about the evolution types in next chapter.

2.2 Weiux's Long-term Evolution Method

Based on the Short-term Evolution Method, Weiux designed there kinds of Long-term Evolution Method. These three methods can detect community evolution in graphs in several timeslots. The first one, named LM1, is similar to Short-term Evolution, but it apply the community detection in many graphs in many timeslot, and construct the multipartite graph instead of bipartite graph in Short-term Evolution Method. The second one, named LM2, does community evolution detection in multiple timeslots by using Short-term Evolution Method many times in every two adjacent timeslots, and combining all the results. The third one, named LM3, is very similar to LM1. However, LM3 adds links in communities which are not only in adjacent timeslots but across timeslots. The LM3 build a new graph but not a new multipartite graph. For choosing a simple but useful algorithm, we decide to use LM1 as our main algorithm for community evolution detection in this work.

2.3 SGCI Algorithm

SGCI algorithm is a community evolution detection proposed by B. Gliwa, S. Saganowski, A. Zygmunt, P. Bro?dka, P. Kazienko, and J. Kozlak. SGCI stands for Stable Group Change Identification. The algorithm can be split in the following steps:

Step 1. Identification of fugitive groups in the separate time periods

Step 2. Identification of group continuation – assigning transitions between groups in adjacent time steps.

Step 3. Separation of the stable groups (lasting for at least three subsequent time steps).

Step 4. Identification of types of group changes. Assigning events from the list describing the change of the state of the group to the transitions

In Step2, it use a measure called modify Jaccard to judge is there a transitions between groups. The definition of modify Jaccard is $MJ(A,B) = \max\left(\frac{A\cap B}{A}, \frac{A\cap B}{B}\right)$, where A and B are two groups. Because of this simple but useful measurement, the computation time of SGCI algorithm is very short compare to other algorithm. In Step4, there are 8 events in SGCI's definition. They are "Split", "Deletion", "Merge", "Addition", "Split_Merge", "Decay", "Constancy", and "Chang Size".

doi:10.6342/NTU201601504

Chapter 3
Our Method

In this chapter, we talk about how we do the community evolution detection and prediction, including our definition of evolution types. We also introduce how we transfer evolution chains to evolution types, and how we predict community evolution types. Before we do the community detection, we examine our evolution detection algorithm with our own synthetic dataset. In order to address these issues, in this chapter we as well introduce how we generate our synthetic data, and how we measure the correctness of Long-term Evolution Method.

3.1 Synthetic Data Generator

We use our own synthetic data to test Weiux's algorithm, so in this section we talk about how we generate the synthetic data. As we want to verify Weiux's algorithm is

doi:10.6342/NTU201601504

robust and can correctly detect the evolution, we are supposed to have datasets with evolution ground truth to judge performance. To our best knowledge, there is no real world dataset containing the ground truth of community evolution, so it is required to generate a synthetic dataset on our own and assign the community evolution manually. If Long-term Evolution Method can precisely detect the community evolution in the synthetic dataset, then it indicates that this algorithm is useful and correct.

Our synthetic data generator is based on the Stochastic Block Model [4], which is a widely used model for generating synthetic dataset with community ground truth. For a simple Stochastic Block Model, each of nodes will be assigned to one of K communities. Assume node i is in community C_i and node j is in community C_j. An edge is placed between node i and node j with probability $\psi_{C_iC_j}$. We can define a $K \times K$ matrix ψ to construct all the edges [5]. Here we define $\psi_{C_iC_j} = p_{in}$ if $C_i = C_j$, otherwise $\psi_{C_iC_j} = p_{out}$.

To build a simple but representative synthetic data, we only consider two kinds of community evolution – "merge" and "split" in our synthetic data. We first generate a graph with community by the Stochastic Block Model. After generating one graph, we assign which communities should merge and which communities should split. Finally,

doi:10.6342/NTU201601504

we generate a new graph followed by our evolution assignment. The following content is the detail of the synthetic data generator.

Step1 Generate one graph by Stochastic Block Model. In this step, we generate graph by Stochastic Block Model with two probabilities, p_{in} and p_{out}. If two nodes are in the same community, there will be an edge connecting two nodes with probability equal to p_{in}. On the other hand, if two nodes are in different communities, there will be an edge connecting two nodes with probability equal to p_{out}. To generate graphs with community structure, the constraint is that p_{in} should be larger than p_{out}, because the definition of community is the connection between nodes in same community is larger than connection between nodes in different community. Throughout this step, we will set number of communities, and number of nodes in each community. With p_{in} and p_{out}, we can generate a graph with community ground truth based on Stochastic Block Model.

Step2 Assign community evolution. In this step, we decide which community should merge or split in next timeslot. We first separate communities into two groups. The community in the first group are going to merge with each other in next time slot, and the community in the second group are going to split into two communities. For

doi:10.6342/NTU201601504

example, we can assign community 1.1 and community 1.2 to merge into community 2.1, and assign community 1.3 to split into community 2.2 and community 2.3, where community a.b means the b-th community in timeslot a. In this example, community 1.1, community 1.2, and community 2.1 are in the same evolution chain. For easy computation and analysis, there will be only two communities becoming one or one community becoming two. There are no three or more communities merging into one or one community split into three or more communities. In this step, we also decide the evolution in the timeslot after next timeslot. If two communities are merging into one community in next timeslot, the merged community will split into two communities in the timeslot after next timeslot. For example, if community 1.1 and community 1.2 merge into community 2.1, then community 2.1 will, as our design, split into community 3.1 and community 3.2. The reason why we choose to specify the merged group should split in next timeslot is to make this synthetic data simple and reduce the time for generating community evolution.

Step3 Move nodes to new communities and reconstruct the graph. After we assign the community evolution, we move nodes from original communities to new communities one by one and reconstruct a new graph. We move nodes from the

doi:10.6342/NTU201601504

communities in present timeslot to the communities we assigned in next timeslot. Every node object will have a variable about which community the node is in. By change value of this variable, we move the node to new community. When a node is moved to a new community, some of its edges should be reconstructed with the same probability used in constructing Stochastic Block Model in Step1. The edges which will be reconstructed will be mentioned later. There are two possible evolution of communities: merge and split. For two communities which will merge into a new community, nodes will break their edges with the community and the community they are going to merged with. The edge connecting in the new community will be generated with probability equal to p_{in}. For example, if a node is originally in community 1.1 and it is assigned to merge with community 1.2 into community 2.1, the node will break all the edges connecting to nodes in community 1.1 and community 1.2 but remain the edges connecting to other communities. The edges in community 2.1 will reconstruct with the probability equal to p_{in}. For a community that will split into two communities, nodes in the community will break their edges with each other, and the edges connecting between the two new communities will be assigned with the probability p_{out}. For example, if a node is originally in community 1.3 and community 1.3 is going to split

doi:10.6342/NTU201601504

into community 2.2 and community 2.3, the node will break all the edges connecting to nodes in community 1.3. The edges connecting between nodes in community 2.2 and nodes in community 2.3 will be generated with probability p_{out}. It is notable that edges should be unchanged if they connect two nodes from different communities which won't merge together. For example, if community 1.1 merges with community 1.2, the edges connecting nodes in community 1.1 and nodes in communities other than community 1.2 should keep unchanged. After all nodes are moved to new communities respectively and all edges are reconstructed with the probability set in Step 1, we get a new graph with new communities, new edges, and community evolution ground truth. By applying these procedures, the probability of the connection between two nodes in same communities and the probability of connection between two nodes in different communities remain the same with the probability we used in Step1. Besides, the connections between communities that do not evolve together will remain the same, which is more reasonable and more realistic for community evolution.

Step4 Adding core nodes and noise. In this step, we modify the synthetic data by setting some nodes as core nodes in communities and adding some noise when moving nodes to new communities. To make the synthetic data more similar to the real world

doi:10.6342/NTU201601504

case, we add noise. The noise we add in synthetic is that nodes may randomly move to communities other than we assigned. We use a probability called "correctly migration probability" to control the noise. The lager the correctly migration probability is, the less the noise is, and the more stable the evolution is. Core nodes is to simulate the core structure in communities. There are two key point about the core nodes in new synthetic data: (1) core nodes will connect to all the nodes in the same community, and (2) core nodes will always go to the correct communities we assigned in evolution. Because the core nodes will move to new communities correctly, the noise we added in new synthetic data will not affect to the core nodes.

Step5 Repeat Step2 to Step4 to get required number of graphs. After Step4, we will get a new graph for 1st evolution, and we also assign the 2nd evolution in step2. Repeat Step2 to Step4 will give us one more graphs and the pattern of 3rd evolution.

3.2 Measure Correctness of Evolution Detection

To judge how well Weiux's algorithm is, we use normalized mutual information to evaluate the community evolution detection result. Because the algorithm we used (LM1) generate the community evolution chain instead of community evolution type, we cannot just compare the evolution type of the detection result and the ground truth.

doi:10.6342/NTU201601504

As a result, we need a method to calculate the precision of the evolution chain we detected. In the algorithm LM1, LM2, and LM3, the last step of these algorithms are the same. In the final step of these algorithms, they apply a community detection on the bipartite graph or multipartite graph to get community evolution chain, so the evolution chain we detected is actually the communities in these graphs. Finding communities in graphs is a famous problem in online social network analysis. Many works manage to find community detection algorithm and they need a method to evaluate their algorithm. Normalized mutual information [12] is one of the common used algorithm to compare the community detection result and the community ground truth. In Chapter 3, we know the evolution chains are the communities detected from the bipartite or multipartite graphs. We can therefore use normalized mutual information as the measurement of our result.

3.3 Definition of Evolution Types

We propose our own defined evolution types in this work. Most community evolution detection algorithms try to detect the evolution type, which is very different from the way in Weiux's algorithm. To compare the result with other algorithm, one way is to generate the evolution types rather than the evolution chains. Evolution type

doi:10.6342/NTU201601504

is much easier to read and analysis than evolution chain. The output of Long-term Evolution Method is evolution chain so we propose a method to transform evolution chain into evolution type. First, we define 7 types of evolution types. They are “Birth”, “Merge”, “Split”, “Growth”, “Shrink”, “Continue”, and “Death”. The following is the concept of each evolution type.

Birth “Birth” is the evolution type which means the community is just appear in the present timeslot. No community is in its history. When we concentrate in what happened to a community before, this evolution type is one of the possible case. However, if we focus on what happens to a community next timeslot, this evolution type will never be the answer because that will break our concept: no community is in its history.

Merge This evolution type indicates that a community is merged form other communities or it will merge with other communities. It depends on which timeslot we are talking about. If we are talking about what happened to a community before, “Merge” means this community is merge from communities in previous timeslot. If we are talking about what will happen to a community, “Merge” means this community is going to merge with other communities in the present timeslot and become new

doi:10.6342/NTU201601504

communities in next timeslot.

Split This evolution means a community is split form communities in previous timeslot or a community will split into multiple communities in next timeslot. Again, it depends on which timeslot we are focus on.

Growth The type “Growth” is quite different from “Merge” and “Split”. “Merge” and “Split” involve the number of communities changed from previous timeslot to present timeslot or will change from present timeslot to next timeslot. “Growth” means that number of nodes in communities increased when communities evolved to present timeslot, or number of nodes in communities will increase when communities evolve to next timeslot.

Shrink This evolution type is a contrary case of “Growth”. The number of communities have never changed. However, number of nodes in communities decreased when communities evolved to present timeslot, or number of nodes in communities will decrease when communities evolve to next timeslot.

Continue This evolution type means that the number of communities have never changed, like “Growth” and “Shrink”, but number of nodes in communities remains the same or no significant change when communities evolved to present timeslot or when communities evolve to next timeslot. We talk about how to distinguish significant change or not in Section 3.4.

Death This evolution type is a contrary case of “Birth”. A community is given the evolution type “Death” as it will disappear in next timeslot and the evolution of this community will end in the present timeslot. No other community in next timeslot is evolved from this community. Different from the type “Birth”, this evolution type can only happen when we talk about a community’s future. If we focus on what happens to a community before, this evolution type will never be the possible type of this community. No community is evolved from a “Death” community.

3.4 Method to Transfer Evolution Chain to Evolution Types

In this section, we give a method to transfer the detected evolution chain into evolution type we defined in Section 3.3. Because most works focus on what happens next to a community, we focus on the future of communities as well. As a result, “Birth” will not appear in this method. In this method, we first decide whether communities are “Death” or not. If the communities are not “Death”, then next step we see if communities are “Merge” or “Split”. If communities are neither “Merge” nor “Split”, we will calculate the total number of nodes in communities to judge the communities

doi:10.6342/NTU201601504

belonging to “Growth”, “Shrink”, or “Continue”. The following is the detail of the procedure.

Step1 Decide evolution type is “Death” or not. Given a community evolution chain, there are many communities in the chain. If a community is in the last timeslot of the community evolution chain, this community will be given a type “Death” since no community is in next timeslot. For example, if there is an evolution chain “12:3, 22:4, 35:4”, the evolution type of community indexed 22 and community indexed 35 will be “Death” because there is no other community in this chain which time index is 5. And, for the community indexed 12, the evolution type of it is not “Death” because in this chain there are two communities (community 22 and community 35) which are in the following time slot (time index 4). If a community is not “Death”, then we go to the next step.

Step2 Decide evolution type is “Merge” or “Split”. In this step, we will calculate the number of communities in each timeslot in an evolution chain, and compare the numbers between each timeslot. If the number of communities in a given timeslot is less than the number of communities in the next timeslot, the evolution types of communities in given timeslot will be “Split”. Otherwise, the evolution type will be

doi:10.6342/NTU201601504

“Merge”. For example, if there is an evolution chain “12:3, 22:4, 35:4”, we can calculate the number of communities in time index 3 and in time index 4. There is one community whose time index is 3 in this chain, and there are two communities whose time index is 4 in this chain. Then, we can compare the number of communities in time index 3 and the number of communities in time index 4. The number of communities in time index 3 is less than that in time index 4, so the community in time index 3 will be given a type “Split”. “Merge” and “Split” are sometimes having different definition. One may consider that only multiple communities merge into one community is called “Merge”. However, others may think that large number of communities merging into small number of communities can be called “Merge”. To make our transferring method more flexible, we used a factor α to control the rule we used to decide “Merge” and “Split”. We used an example to show how the factor controls the decision. Given x is the number of communities in the present timeslot, and y is the number of communities in next timeslot, the possible evolution type of communities in the present timeslot is in the Table 3-1.

Table 3-1 Rule of deciding “Merge” or “Split”

Compare x and y	Evolution Type

$x > \alpha y$	Merge
$x < \frac{1}{\alpha} y$	Split
$otherwise$	Growth, Shrink, or Continue

If $\frac{1}{\alpha} y < x < \alpha y$, then we go to next step to decide the evolution type is “Growth”, “Shrink”, or “Continue”.

Step3 Decide evolution type is “Growth”, “Shrink”, or “Continue”. In this step, we compare the number of nodes in communities. We will calculate the total number of nodes in communities in each timeslot, and compare the numbers between each timeslot. If the total number of nodes in a timeslot is larger than the total number of nodes in its next timeslot, the communities in this timeslot will be given the type “Shrink.” In contrast, if the total number of nodes in a timeslot is smaller than the total number of nodes in its next timeslot, the communities in this timeslot will be given the type “Growth”. Other cases besides these two will be the type “Continue”. For example, there is an evolution chain “35:3, 36:3, 50:4, 51:4”. Given the number of communities in time index 3 is two and the number of communities in time index 4 is also two, the evolution type of communities in time index 3 is neither “Merge” nor “Split”. We have to decide the evolution type of communities in time index 3. Assume the number of nodes in community 35, 36, 50, 51 is 20, 20, 10, 10. The total number of nodes in communities in time index 3 is 20 + 20 = 40, and the total number of nodes in communities in time index 4 is 10 + 10 = 20. Because the total nodes in time index 3 is larger than that in time index 4 (40 > 20), the evolution type of community 35 and community 36 should be "Shrink." There is a problem that one may think the community with 40 nodes which become a new community with 39 nodes should not be given the type "Shrink" since there is only one node difference. Again, to make our method more flexible, we use a factor β to control the decision of "Growth" and "Shrink". The following Table 3-2 shows how the factor β control the decision. Assume the total number of nodes in a timeslot is x and the total number of nodes in its next timeslot is y.

Table 3-2 Rule of deciding "Shrink", "Growth", or "Continue"

Compare x and y	Evolution Type
$x > \beta y$	Shrink
$x < \frac{1}{\beta} y$	Growth
$otherwise$	Continue

Given a community evolution chain, we can apply our method Step1 to Step 3 in

doi:10.6342/NTU201601504

each timeslot and give every community in each timeslot a proper evolution type. All we need are evolution chains and the number of nodes in each community. By adjust the factor α and β, we can affect the decision of evolution types.

3.5 Method for Community Evolution Prediction

For predicting community evolution, we propose a method to predict our evolution types. Although Weiux's algorithm generates evolution chains, it is difficult to predict which communities will be in next timeslot in a chain. As a result, we switch to predict the evolution types. To predict the evolution type, we decide to use Libsvm [9] to build a classification model. Chih-Chung Chang and Chih-Jen Lin propose the Libsvm, a library for support vector machines. It is a tool which can perform support vector classification, regression, and distribution estimation. This tool makes the usage of SVM much simpler. There are many interfaces and extensions to Libsvm. We use the Libsvm interfaces in R because R is a convenient and powerful tool to analyze the data. As we want to build a prediction model with SVM, we have to select features from the data. In next section, we show the details of how we select the features and which features we pick.

doi:10.6342/NTU201601504

3.6 Features Selection for Community Evolution Prediction

We select many kinds of features for the classification. In our prediction model, the evolution types of communities are the predicting target. To build a classification model, we select 310 features for each community. The 310 features can be classified into 4 different groups. There are features about communities, features about the difference between previous communities, features about the evolution chains, and features about the previous evolution types. The following content is the detail of these 4 groups of features.

3.6.1 Features about communities

The features about communities are communities' index, time index, size, density, cohesion, leadership, closeness of nodes, degree of nodes, and betweenness of nodes. There are 51 features in this group, which are feature 1 to feature 51. The explanations for the features are presented below.

Index – Each community is given an index. This index never repeats for communities in all timeslot. The community index is not a random number but a number start from 0 and always increase. As a result, the communities in present timeslot will have their index smaller than the index of communities in next timeslot.

doi:10.6342/NTU201601504

Time index – This feature represents which time slot a community is in. The time index is start from 0.

Size – The number of nodes in the group.

Density – This feature represents how many connections between nodes in a community verse the maximum possible connections between nodes in this community. The equation is below:

$$\text{Density} = \frac{\sum_i \sum_j a(i,j)}{n(n-1)}$$

Cohesion – A measurement of the strength of connections inside a community versus the strength of connections outside this community. The equation is as below, where $w(i,j)$ is weight of the connection between node i and node j and N is the total number of nodes in graph.

$$\text{Cohesion} = \frac{\frac{\sum_{i \in C} \sum_{j \in C} w(i,j)}{n(n-1)}}{\frac{\sum_{i \in C} \sum_{j \notin C} w(i,j)}{N(N-n)}}$$

Leadership – A measurement of the centralization of a community. The equation is as below, where d_{max} is the maximum degree in the community and d_i is the degree of the i-th node in the community.

$$\text{Leadership} = \sum_{i=1}^{n} \frac{\mathrm{d}_{max} - \mathrm{d}_i}{(n-2)(n-1)}$$

Closeness of nodes – The closeness is a node measure in inverse of how far a node

doi:10.6342/NTU201601504

is from other nodes. Here the “other nodes” means the other nodes is the same community. A node’s closeness toward its community is define as:

$$\text{Closeness}(i) = \frac{1}{\sum_{j \in C} d(i,j)}$$

where $d(i,j)$ is the shortest distance between node i and node j. Given a community, we can calculate the sum, the maximum, the minimum, the average, and the standard deviation of the closeness of all nodes. In our feature selection, we calculate more closeness measure of a community. In addition to calculate closeness with all nodes in a community, we pick the leaders, the outermost nodes, and the core nodes of this community. The leaders of a community are defined by the closeness of nodes. If a node’s closeness is larger than the mean of the closeness plus two times of standard deviation of closeness of all nodes, it is the leader of its community. The outermost nodes are, in contrary, the nodes which closeness are smaller than the mean of the closeness minus two times of standard deviation of closeness of all nodes. The definition of core nodes is the same as our detection algorithm. So, in selecting communities’ closeness measure, we consider 4 groups of nodes in communities and each group we calculate 5 different measure of closeness. We get $4 \times 5 = 20$ features in calculating closeness of nodes.

doi:10.6342/NTU201601504

Degree of nodes – Degree is a node measure. It represents how many links a node has between the node and other nodes. Here the “other nodes” means the other nodes belong to the same community. In calculating the degree of nodes, we do the same thing in calculating closeness of nodes. We pick 4 groups of nodes in a community, which are leaders, outermost nodes, core nodes, and all nodes. The definition of leaders and outermost nodes are the same as in calculating closeness of nodes. We calculate the sum, the maximum, the minimum, the mean, and the standard deviation of degree of these 4 groups. We get 20 features in calculating degree of nodes.

Betweenness of nodes – Betweenness is a node measure. The betweenness of a node represents the number of shortest path from nodes in a community pass through this node. The equation is below, which σ_{jk} is number of shortest path from node j and node k and $\sigma_{jk}(i)$ is number of shortest path from node j and node k through node i.

$$\text{Betweenness}(i)= \sum_{j\neq k\neq i} \frac{\sigma_{jk}(i)}{\sigma_{jk}}$$

In calculating the betweenness of nodes, we consider all nodes in a community. We calculate the sum, the maximum, the minimum, the mean, and the standard deviation of betweenness of all nodes.

3.6.2 Features about difference between previous communities

In an evolution chain, there are many communities in different timeslots. In Section 3.6.1 we select the features from communities. In this section we calculate the differences of features between communities in different timeslot. Here we use an example to show how to calculate the difference of features. Suppose there is an evolution chain “35:3, 36:3, 50:4”. From Section 3.6.1, we get the size of communities are 15, 20, and 40. The differences of the feature “size” between community 50 and the communities in its previous timeslot (which is community 35 and 36) are 25 and 20. For each feature, except for community index and time index, in Section 3.6.1, we can get a list of differences of feature because the communities in previous timeslot may be more than one, just like the example. For every lists of differences of feature selected in Section 3.6.1, we can calculate their sum, maximum, minimum, mean, and standard deviation. So, there are $(51-2)\times 5=245$ features which is about the differences of features in Section 3.6.1. We add three more features. The first added feature represents the percentage of the nodes in a community are new nodes which are not in the previous timeslot. The second feature is the percentage of the nodes which are in previous timeslot but not in this community in present timeslot. The third one is the

doi:10.6342/NTU201601504

percentage of nodes in a community that has already appeared in previous timeslot. The third feature is equal to 1.0 minus the first feature. By adding these three features, we get $245 + 3 = 248$ features in this section, which are feature 52 to feature 299.

3.6.3 Features about evolution chains

As our community evolution detection algorithm detect the evolution chains, features of evolution chains may play an important role in prediction. So, we consider to add the feature of the evolution chain to our feature selection. To calculate the features of an evolution chain, we first construct a list in which each element is the number of communities in each timeslot. For example, given an evolution chain “35:3, 36:3, 50:4, 63:5”, we will construct the list [2, 1, 1] which represents there are two communities in time index 3, one community in time index 4, and one community in time index 5. Then, we calculate the sum, the maximum, the minimum, the mean, and the standard deviation of the list. Besides, we also calculate the size of the list and the number of communities in the same timeslot. When we select the features, it is important that we do not select the feature which contains the answer of the evolution. So, when we construct the list from an evolution chain, we consider which timeslot of a community we are calculate now. For example, in the previous example, we get the

doi:10.6342/NTU201601504

list [2, 1, 1] from the evolution chain “35:3, 36:3, 50:4, 63:5”. If we are selecting the features of community 36, the list will be [2]. However, if we are selecting the features of community 50, the list will be [2, 1] because we cannot tell the answer that whether there will be a community 63 in next timeslot. There are 7 features about the evolution chain, which are feature 300 to feature 306.

Here we further talk about each feature in this group. Feature 300 is the sum of the list we mentioned before. The large value means this chain is very big and contains many communities. Feature 301 and Feature 302 are the maximum and the minimum of the list. Feature 303 is the mean of the list, which means in average how many communities in each timeslot. Feature 304 is the standard deviation of the list. If the value is high, it means that the changes in number of communities is big, and this evolution chain is very unstable. Feature 305 is the number of communities in the same timeslot. Feature 306 is the size of the list. This value means how long an evolution chain is. The larger the value is, the longer the evolution chain lives, which implies that the evolution chain may be still alive in next timeslot.

3.6.4 Features about the previous evolution type

The final group of features is the features about the previous evolution type. Every

doi:10.6342/NTU201601504

community will have two evolution types. One is what the community is from, the other is what will happen to the community. If a community is the result of two communities merging with each other in previous timeslot, this community's previous evolution type is "Merge". The previous evolution type will be one of "Birth", "Merge", "Split", "Growth", "Shrink", and "Continue". It is notable that the type "Death" is not a possible type in previous evolution type. This is mentioned in Section 3.3. We also add three different features in this group. The first one represents whether a community is survived from previous timeslot. The second one represents whether a community's members increased compared to the previous timeslot. The third one represents if a community merged or not from previous timeslot. These three features are recorded with the value type of "true" or "false". In Chapter 6, we not only predict the evolution types, but also predict whether a community will live, whether a community will growth, and whether a community will merge. As a result, we add these three features as the input features. This will be discussed in Section 6.1. Because it is impossible for a community evolves form a "Death" community and, as a result, the feature of whether a community will live is always true. In the prediction model, we will remove this feature for it is helpless. Before we remove this feature, there are 4 features in this group,

doi:10.6342/NTU201601504

which are feature 307 to feature 310.

doi:10.6342/NTU201601504

Chapter 4
Synthetic Dataset Experiment

In this chapter, we use the synthetic data we generate to test the evolution detection algorithm. In Chapter 2, we talk about the Long-term Evolution Method designed by Weixu Lin, and we decide to use this method as our main algorithm to do more evolution analysis and evolution prediction. However, his algorithm is not fully verified. In this chapter, we try to verify our long term evolution algorithm (LM1) by testing it with some datasets which contain evolution ground truth. We are also interesting in the two main parameters in the algorithm, p-core and threshold, so we conduct the experiment to show the impact of these two parameters and the detection result.

4.1 Experiment Result

In section 3.1 and section 3.2, we generate the synthetic data with ground truth

doi:10.6342/NTU201601504

and the measurement for judging Long-term Evolution Method. In our synthetic data, we set 4 communities, and each community contains 20 nodes. The nodes connect with each other in the same communities with probability 0.9, which is p_{in}. The nodes in different communities connect with each other with probability 0.1, which is p_{out} in Stochastic Block Model. We generate 6 graphs which have 5 times evolution. We set the probability of correct migration to 1.0, which means there is no noise. We do not add core node in this experiment. We would like to see how Long-term Evolution will perform if communities contain no core node. For the arguments of Long-term Evolution Method, we set p-core as 0.2 and threshold as 0.05. In this algorithm, the first step is applying community detection on each graphs. Here we use community ground truth instead of apply community detection in the first step of this algorithm because we want to evaluate our evolution detection algorithm. The error of community detection will impact the result of evolution detection. As we just want to judge how well our evolution detection algorithm is, we control the community detection to 100% correct by using community ground truth. The result is shown below.

Table 4-1 Evolution chains detected by Long-term Evolution Method

Chain Index	Community Evolution Chain

0	0:0, 1:0, 4:1, 10:2
1	3:0, 7:1, 12:2
2	8:1
3	2:0, 5:1, 6:1, 11:2, 14:3
4	9:2, 13:3, 18:4, 19:4, 22:5
5	16:3, 17:3, 21:4, 25:5
6	15:3, 20:4, 23:5, 24:5
7	26:5

Table 4-1 is the result of Long-term Evolution Method. There are 8 chains detected by the algorithm. In each chains, there are several communities in each time index. The syntax of each element in a evolution chain is <Community Index> : <Time Index>. For example, in the first evolution chain, 1:0 means there is a community which index is 1 and time index is 0 in this chain. The evolution ground truth is shown in Table 4-2.

Table 4-2 Ground truth of evolution chain in synthetic data

Chain Index	Community Evolution Chain
0	0:0, 1:0, 4:1, 9:2, 10:2, 13:3, 18:4, 19:4, 22:5
1	2:0, 5:1, 6:1, 11:2, 14:3, 15:3, 20:4, 23:5, 24:5

doi:10.6342/NTU201601504

2	3:0, 7:1, 8:1, 12:2, 16:3, 17:3, 21:4, 25:5, 26:5

As we can see, the community evolution chains between Long-term Evolution Method and the ground truth are different. However, we still can show some correctness of this algorithm.

Because the result is not so well, we try to modify some argument in this algorithm. We modify p-core to 1.0, which means we consider no core nodes in communities (or every node is core node) and the algorithm will calculate similarity of communities considering all nodes in communities. The result is in Table 4-3.

Table 4-3 Evolution chains detected by LM1 with new arguments

Chain Index	Community Evolution Chain
0	0:0, 1:0, 4:1, 9:2, 10:2, 13:3, 18:4, 19:4, 22:5
1	2:0, 5:1, 6:1, 11:2, 14:3, 15:3, 20:4, 23:5, 24:5
2	3:0, 7:1, 8:1, 12:2, 16:3, 17:3, 21:4, 25:5, 26:5

The result is just the same as the ground truth. The normalized mutual information of the result is 100%, which means the result is 100% correct. The normalized mutual information shows that the evolution chains we detected are totally correct, and verifies that Long-term Evolution Method is correct. The reason why we get perfect result when

doi:10.6342/NTU201601504

we choose p-core as 1.0 is our synthetic data have no core structure in communities. Choosing p-core to 0.2 is not a good choice in this synthetic data. In most of the real world dataset, communities will contain some core nodes. As a result, we need a new synthetic dataset which contains core structure in communities. Besides, this synthetic data has almost no noise. The only noise in this synthetic is from the Stochastic Block Model. We set p_{in} to 0.9 so the nodes in the same communities may have no connection with probability 0.1. In the real world data, the community evolution is more complicated and has many noise. As a result, we do a new experiment to test a more complicated evolution and analyze the p-core's impact.

4.2 Experiment Result with Cores and Noise

In this experiment, we want to investigate the impact of p-core and threshold. We add core nodes for analyze the impact of p-core. Also, we add noise in the synthetic dataset in this experiment. We start from analysis the p-core's impact. We set p_{in} to 0.5, p_{out} to 0.1, and the correctly migration probability from 1.0 to 0.1. There are 8 communities in the new synthetic data, and each of them contains 40 nodes. For the 40 nodes in each community, 4 of them are set to be core nodes. We test Long-term Evolution Method with p-core from 1.0 to 0.1 and threshold equal to 0.05. We plot the normalized mutual information in Figure 4-1.

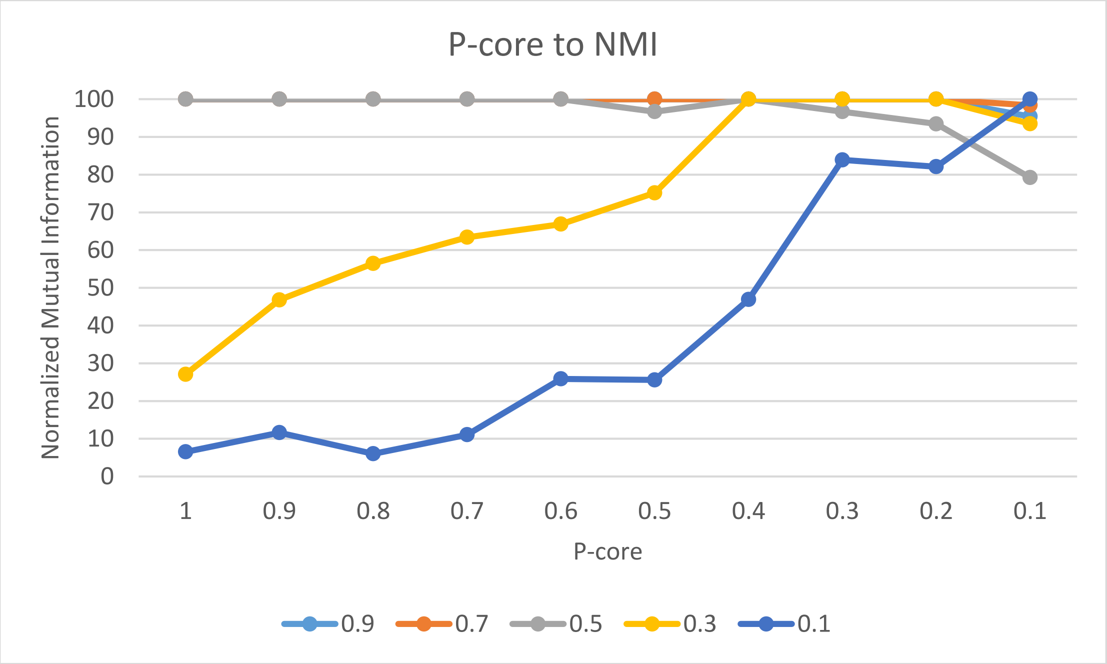

Figure 4-1 NMI to p-core and correctly migration probability

In Figure 4-1, we can see that when the correctly migration probability is large, p-core do little impact to the normalized mutual information. However, when the correctly migration probability is getting smaller, selecting smaller p-core can improve the result. The reason is that the nodes will have larger probability to randomly migrate to wrong communities. The better way for Long-term Evolution Method is to calculate the similarity of communities by considering only the core nodes because core nodes will not migrate to wrong communities. When we choose smaller p-core, we have greater chance to select the core nodes in communities so as to do a correct evolution detection. The figure shows that choosing p-core to 0.3 or 0.2 give us better result in

doi:10.6342/NTU201601504

most of the cases. So, in testing the real world dataset in Chapter 5, we select p-core to be 0.2. Selecting smaller value of p-core also means that the algorithm will compute less nodes when calculating the similarity of communities, and reduce the running time. The following Figure 4-2 is the computing time when choosing different value of p-core. We can see that the computing time decreases as we choose lower p-core.

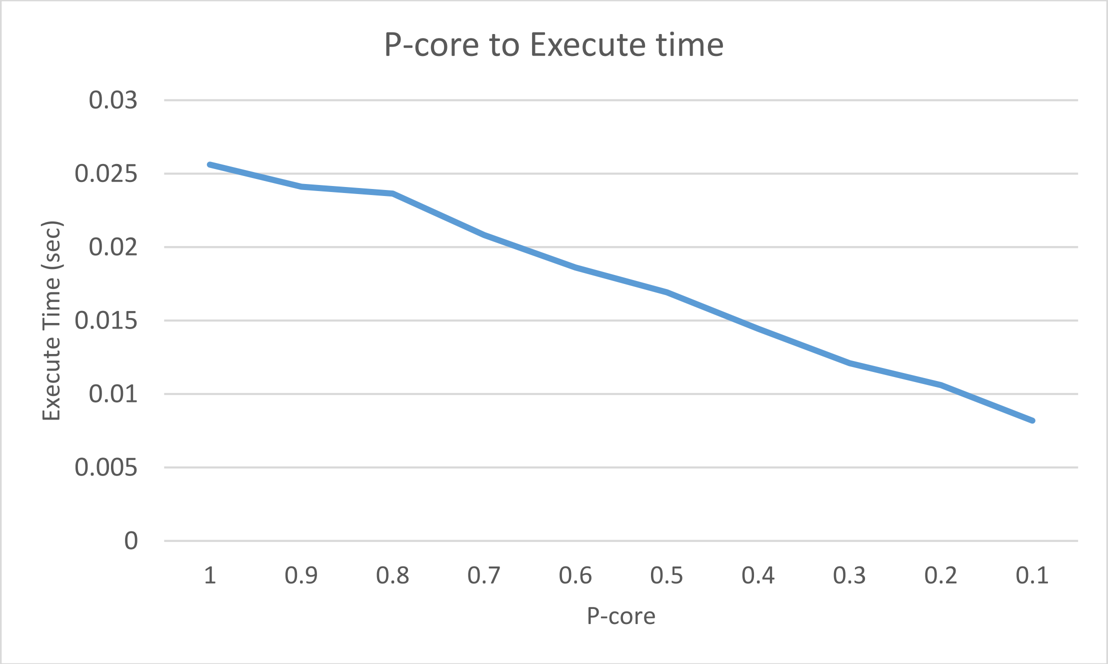

Figure 4-2 Execution time to the p-core

Now we switch to analysis the impact of threshold. We set p-core to 1.0 and correctly migration probability to 0.5, threshold from 1.0 to 0.1, and other arguments are the same with what we set in analyzing p-core. The Figure 4-3 below shows the plot of normalized mutual information to the threshold.

doi:10.6342/NTU201601504

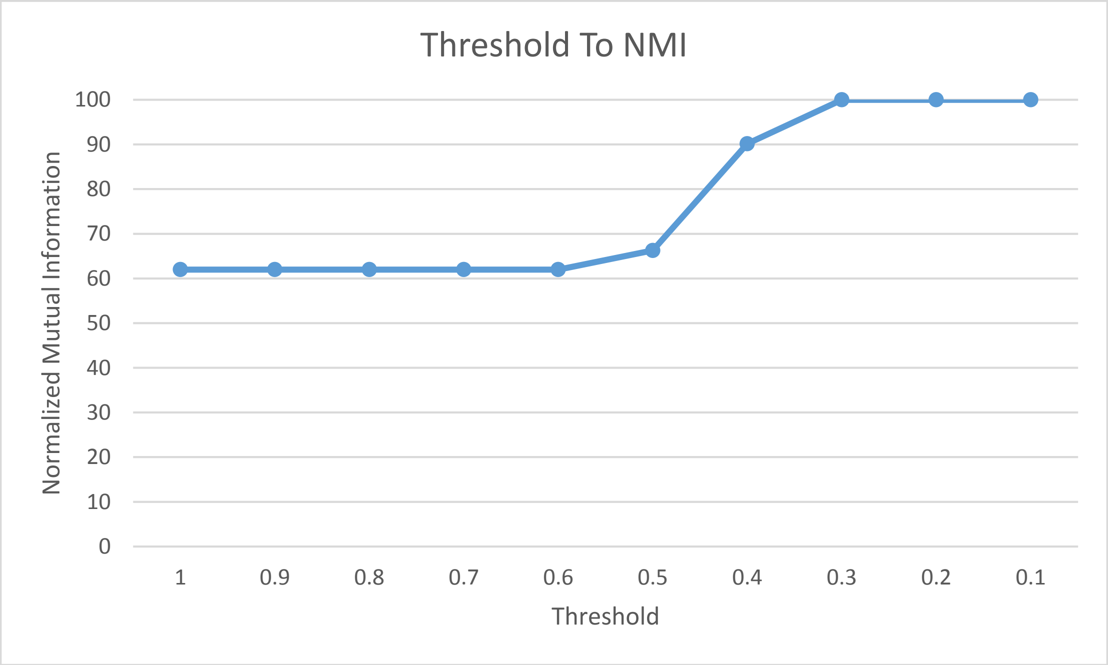

Figure 4-3 Normalized mutual information to the threshold

As Figure 4-3 shows, when the threshold is less than 0.5, the values of normalized mutual information are stocked at 60%. But, as soon as we set the threshold larger than 0.5, the values of normalized mutual information jump to 100%. The threshold acts like a barrier. The result shows that we should not set threshold to large so in the following real world dataset experiment we set threshold to 0.05. The threshold value 0.05 has also been used in [3] when testing real world dataset.

doi:10.6342/NTU201601504

Chapter 5
Real World Dataset Experiment

This chapter is about the real world datasets on Long-term Evolution Method. In Chapter 4, we verify this algorithm is correct, and knowing how the p-core and threshold impact the detection result. In this chapter, we test the algorithm with the real world datasets including Facebook dataset [6] and DBLP dataset [7]. After testing the real world datasets, we should know how well Long-term Evolution Method is when it comes to the real case. Because of no ground truth in real world datasets, we need to do some case study and compare our result with other algorithm to evaluate this algorithm. In this chapter, we use our method to change evolution chains into evolution types: Continue, Merge, Split, Growth, Shrink, Death, and Birth. These types will be used to compare to other algorithm and do prediction in Chapter 6.

5.1 Experiment on DBLP

We test Long-term Evolution Method with DBLP dataset first. The DBLP dataset is a co-author network. We clean up the DBLP dataset by remove the nodes who have published number of papers less than five. After the cleaning up, we separate the DBLP dataset by years and change it to graphs. Each graph contains data for two years. There is a 50% overlap, which is a year overlap, between each graphs. For example, the first graph contains the data in the first and second year, and the second graph contains the data in second and third year. We get 11 graphs which cover the data from D.C. 1995 to 2005. There are 148334 nodes and 1210663 edges in graphs. The reason making the graphs with some overlap is that it will make us much more easily to catch the communities' evolution. This is more important when we test the algorithm with Facebook dataset because the Facebook dataset is much smaller than DBLP dataset. The nodes and communities in each graph in Facebook dataset is smaller than that in DBLP dataset so that it makes the Facebook dataset more difficult to detect the evolution.

Based on the knowledge we learn from Chapter 4, we can set a proper value of p-core and threshold for Long-term Evolution Method. From the experiment in Chapter

doi:10.6342/NTU201601504

4, we set p-core to 0.2 and threshold to 0.05. We run this algorithm on DBLP dataset and it takes 174999 second (without the time of reading graphs) to finish the computation. There are 83301 communities, 32038 evolution chains in the result. To realize the result, we transfer the evolution chains to evolution type we defined. By setting the $\alpha = 1$ and $\beta = 1$, we list the evolution types' distribution in Table 5-1, and plot the distribution in Figure 5-1. We also remove the evolution types of communities in last timeslot because the evolution types of communities in last timeslot is always "Death". We have only 11 graphs so the evolution types of communities in 11th graphs are all "Death"

Table 5-1 Number of each types in Long-term Evolution Method in DBLP

	# of Event
Continue	21874

Death	22958
Merge	[redacted]
Split	6739
Shrink	5384
Grow	8438

doi:10.6342/NTU201601504

th	

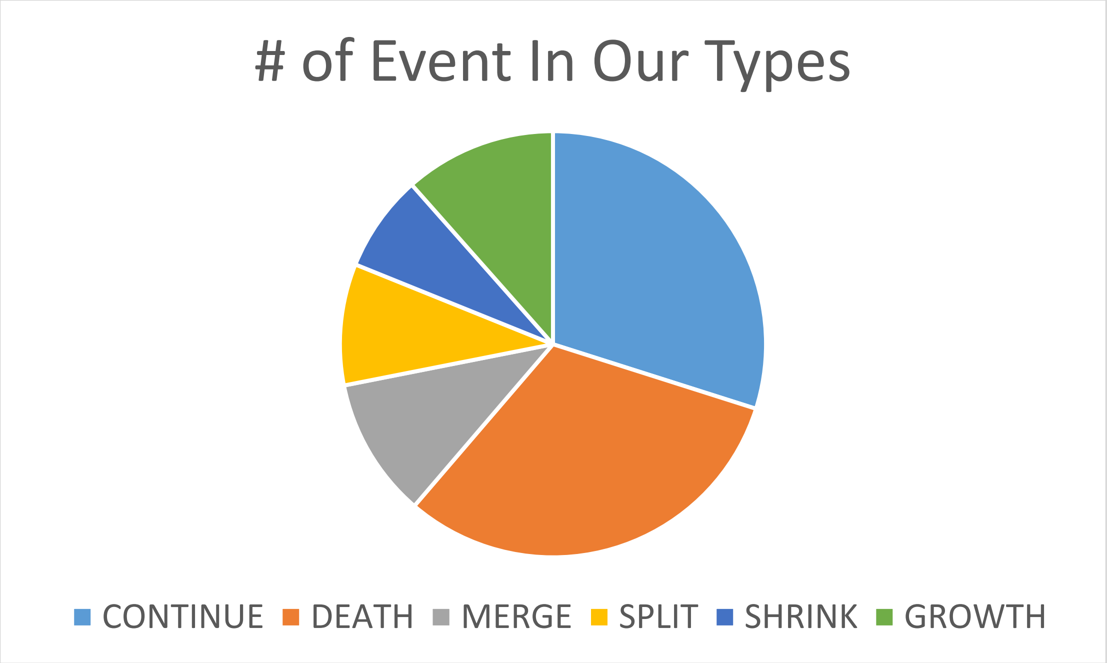

Figure 5-1 Pie chart of types in Long-term Evolution Method in DBLP

As we can see, the “Death” and “Continue” are the most of the evolution types. Other types are distributed very evenly. Because of there is no ground truth in community detection and community evolution in DBLP dataset, the only way we can do is to do some case study. In [3], it shows many case about DBLP dataset, Enron dataset, and U.S. Senate dataset, and proves his algorithm perform well on real world dataset, so we will not go further case studying on DBLP dataset. Instead, we compare our result with SGCI, which is one of the famous evolution detection algorithm. The evolution types in SGCI are “Addition”, “Change Size”, “Constancy”, “Decay”, “Deletion”, “Merge”, “Split”, and “Split Merge”. The type “Merge” and “Split” is the same as our definition of “Merge” and “Split”. Our types “Growth” and “Shrink” are similar to types “Addition”, “Deletion”, and “Change Size” in SGCI. The type “Decay” is the same as the type “Death” in our definition. The distribution of SGCI is shown in Table 5-2 and Figure 5-2.

Table 5-2 Number of each types in SGCI in DBLP

	# of Event in SGCI
Addition	3014
Change Size	16329
Const	

ancy	
Decay	22977
Deletion	798
Merge	4088
Split	1484
Split_	95

Merge	

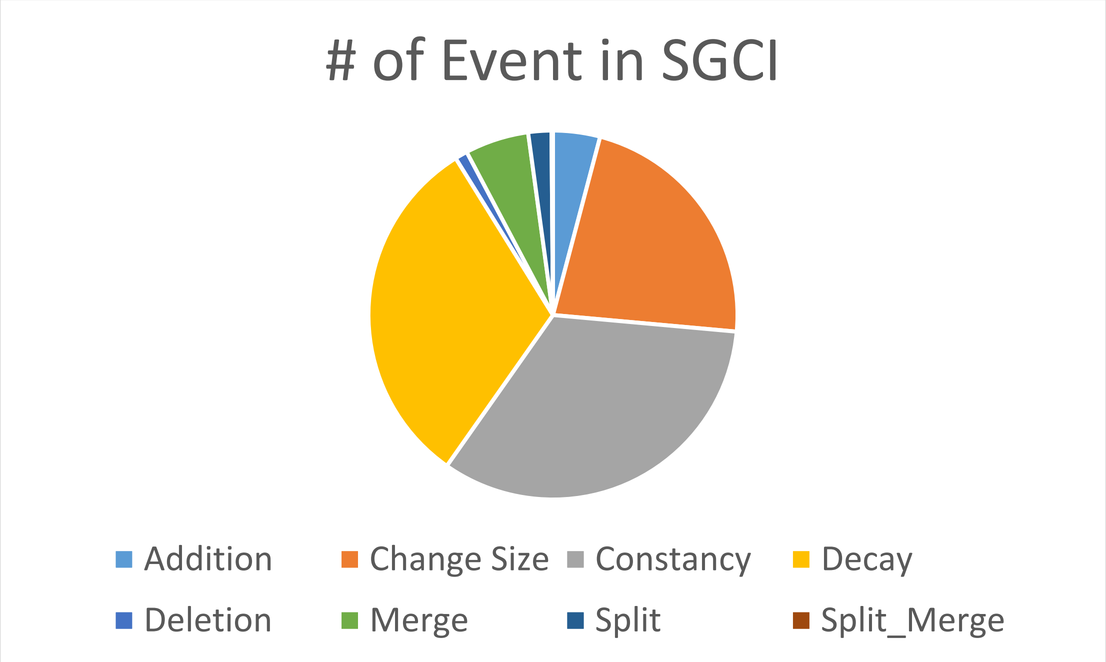

Figure 5-2 Pie chart of types in SGCI in DBLP

As the Figure 5-2 shows, the distribution of evolution types of SGCI is very uneven. The possible reason is that it has 8 different types but we only have 6, and the definition of types is different from us. In Table 5-3 we list the comparison of types in SGCI and types in our definition.

Table 5-3 Comparison of our types and SGCI types in DBLP

	Our Type

doi:10.6342/NTU201601504

		Continue	Death	Growth	Shrink	Merge	Split
SGCI Type	Constancy	21606	132	196	148	1306	990
	Decay	145	18737	282	121	981	2711
	Change Size	70	1124	6455	4530	2290	1860
	Addition	5	1386	86	44	581	912
	Deletion	0	2	41	17	344	392
	Merge	39	1456	670	48	555	1320
	Split	9	113	38	449	647	228
	Split_Merge	0	8	2	27	35	23

The result shows that, except for types “Decay” and “Constancy” are almost match our types “Death” and “Continue”, other types does not match very well. It is because the other types’ definition is different from us. We cannot tell which algorithm is better for each algorithm has their own definition of evolution types. We will have a further discussion of each factor in Chapter 6 when it comes to predicting evolution types.

5.2 Experiment on Facebook

After testing Long-term Evolution Method with DBLP dataset, we do it again with

Facebook dataset. Facebook dataset is a posting dataset. When a user posts something on the other user's wall, there is a link between them. Same as the experiment on DBLP dataset, we separate Facebook Dataset into 17 graphs by every four months starting from Feb. 2006. There is a two-month overlap between each graphs. There are total 45613 nodes and 1559167 edges in these graphs.

We perform Long-term Evolution Method on Facebook dataset and get 9985 communities and 5672 evolution chains in total. Again, we transfer the evolution chain to evolution type with $\alpha = 1$ and $\beta = 1$, and remove the evolution types of communities in last timeslot. The following Table 5-4 and Figure 5-3 is the distribution of the evolution types of Facebook dataset.

Table 5-4 Number of each types in Long-term Evolution Method in Facebook

	# of Event
Continue	3036
Death	4610
Merge	533
Split	655
Shrink	313

doi:10.6342/NTU201601504

Growth	838

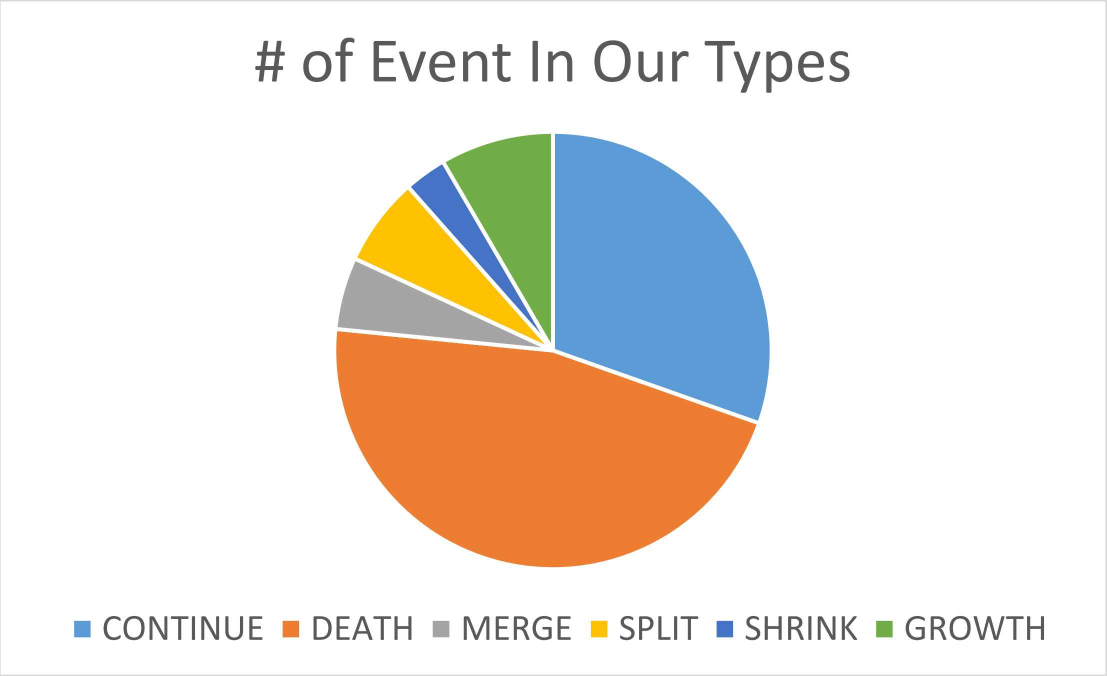

Figure 5-3 Pie chart of types in Long-term Evolution Method in Facebook

The two major evolution types in the result are still “Death” and “Continue”, and “Death” is almost the half of the total types. We also perform SGCI on Facebook dataset. Table 5-5 and Figure 5-4 is the result of SGCI.

Table 5-5 Number of each types in SGCI in Facebook

	# of Event in SGCI
A dd iti	562

doi:10.6342/NTU201601504

on	
ChangeSize	1042
Constancy	
Decay	4908
Deletion	220

doi:10.6342/NTU201601504

Merge	93
Split	36
Split – Merge	0

doi:10.6342/NTU201601504

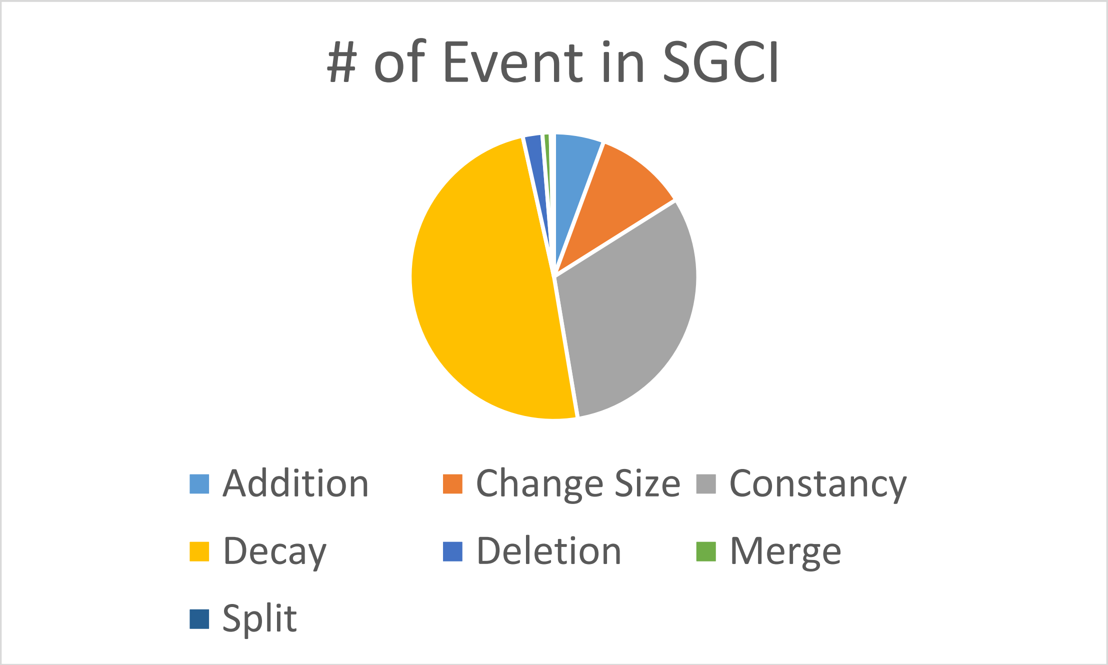

Figure 5-4 Pie chart of types in SGCI in Facebook

In Table 5-5 and Figure 5-4, we can find that the distribution of the types is more uneven than us, which is the same phenomenon we see in the experiment of DBLP dataset. As what we did in testing DBLP dataset, we list the comparison of types in SGCI and types we defined in Table 5-6.

Table 5-6 Comparison of our types and SGCI types in Facebook

		Our Type					
		Continue	Death	Growth	Shrink	Merge	Split
SGCI Type	Constancy	3028	2	4	2	53	35
	Decay	3	4101	4	2	310	488

doi:10.6342/NTU201601504

	Change Size	2	58	507	297	97	81
	Addition	1	425	4	2	46	84
	Deletion	0	0	2	1	119	98
	Merge	2	23	12	1	6	49
	Split	0	1	0	8	24	3

The communities with types “Constancy” is almost the same as communities with types “Continue” in our definition. The type “Decay” is detected as “Death” in our types. The types “Growth” and “Shrink” in our result are mostly detected as “Change Size” in SGCI. It is matched our knowledge about “Change Size” in SGCI as we mention that our types “Growth” and “Shrink” are similar to types “Addition”, “Deletion”, and “Change Size” in SGCI. However, other types are not matched.

5.3 Case Studying in Facebook Dataset

We do the case studying on Facebook dataset by analysis the distribution of evolution types in different timeslots, and matching the events of Facebook with the detection result. Although [3] contains many case studying to support Long-term Evolution Method, it didn’t apply the algorithm on Facebook dataset. As a result, in this section, we try to analysis the evolution in Facebook dataset and support this algorithm and our method of transferring evolution chain to evolution type. Facebook is a great company and is still growing bigger. We can find the history of the Facebook on the Internet. We may able to link the evolution of Facebook dataset we detected to what actually happened to Facebook in its history. To further understand the evolution of Facebook dataset, we do case study by plotting the distribution of evolution types in each timeslot. Figure 5-5 is the distribution of types in each timeslot. It shows how many percentages of each types in each timeslot.

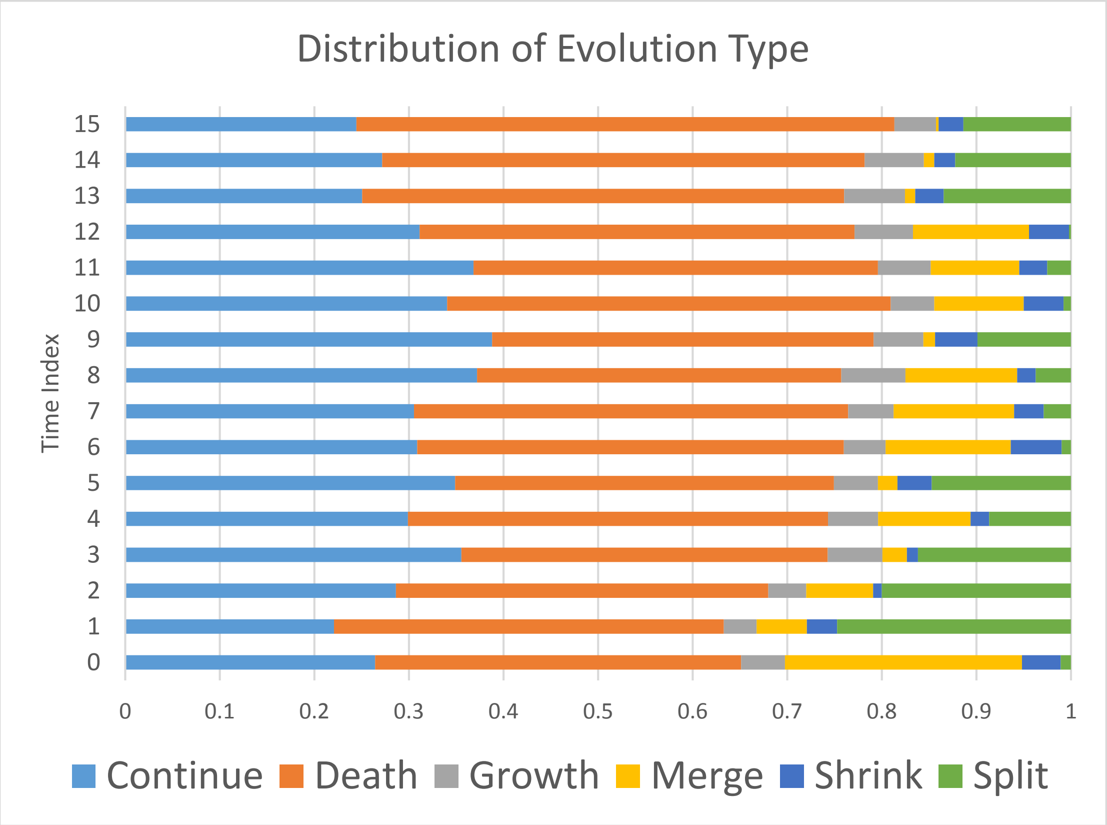

Figure 5-5 Percentages of evolution types in each timeslot in Facebook

doi:10.6342/NTU201601504

There are two interesting points in this plot. First, in the second timeslot which time index is 1, we can see the percentages of type “Split” is much larger than its previous timeslot. Second, in the 14th timeslot which time index is 13, the percentages of type “Split” is again much larger than its previous timeslot. In the second timeslot, it contains the data from 2006/4 to 2006/7. In 2006/9, Facebook was opened to everyone at least 13 years old with a valid email address. The possible reason why the type “Split” become larger is the nodes in 2006/4 ~ 2006/7 will break their own communities because more friends are join the Facebook and they want to form new communities with new join friends. In the 14th timeslot, it contains data from 2008/4 to 2008/7. In 2008/7, Facebook was fully opened to China users due to 2008 Summer Olympic Games at Beijing. So, again, the structure of Facebook communities changed a lot at this period because of the same reason: new friends, in this case Chinese users, were join to Facebook. As we know that China finally block the Facebook in the late 2008, we may be able to see another big changes happened in community evolution types in Facebook if we have more data.

doi:10.6342/NTU201601504

Chapter 6
Community Evolution Prediction

We use the detection result in Chapter 5, and build a prediction model for community evolution in this chapter. In Chapter 5, we test Long-term Evolution Method with real world datasets, and give a method to transfer community evolution chains to community evolution types. Long-term Evolution Method can do perfect detection. In this chapter, we want to do more – predict the evolution types. We want to show that evolution is predictable and our method help us to reach the goal.

6.1 Experiment on Facebook Dataset

We first predict the communities' evolution in Facebook dataset. In Chapter 5, we apply Long-term Evolution Method on Facebook dataset and get 9985 communities and 5672 evolution chains. To build a more precise prediction, we remove the

doi:10.6342/NTU201601504

communities which survive for no more than 2 timeslots. The communities which survive for very short timeslots may be unpredictable. After we remove the short-life communities, we have 4235 communities. Our predicting target is the evolution types. Before we start our model training, we should notice the distribution of the evolution types in training and testing set. Table 6-1 is the distribution of types.

Table 6-1 Number of evolution types in training/testing set in Facebook

	# of Event
Continue	1388
Death	760
Merge	652
Split	828
Shrink	257
Growth	350

As we can see, the distribution of 4235 communities is uneven. As a result, we should first evenly and randomly sample each evolution types to get an unbiased prediction model. We sample 257 data for every types, pick 80% of data as the training set, and use the feature we select in Section 3.6 to train the prediction model. Then, we

doi:10.6342/NTU201601504

use the remaining 20% data as the testing set to test our model. The prediction result is below in Table 6-2.

Table 6-2 Predicting 6 evolution types in Facebook

		Ground Truth					
		Continue	Death	Growth	Merge	Shrink	Split
Prediction Result	Continue	0	0	0	0	0	0
	Death	30	38	13	1	0	2
	Growth	18	2	15	0	2	0
	Merge	0	0	0	38	0	10
	Shrink	4	8	12	0	59	1
	Split	0	0	0	7	0	49

The accuracy of the prediction is 64.40%. It seems like the performance is not very well. However, if we consider the accuracy of randomly prediction which is only 16.67%, it is still far better than random.

The accuracy of predicting 6 evolution types is only 64.40%. The result may because of the difficulty of prediction. Predicting 6 evolution types may be too difficult. So, we try to build the prediction models to the different but much easy targets: whether a community will live, whether a community will grow, and whether a community will merge. Same as the prediction model target to 6 evolution types, we first evenly sample the training/testing set, pick 80% of data as training set, and use the feature we select in Section 3.6 to train the prediction model. The following three tables, which are Table 6-3, Table 6-4, and Table 6-5, are the testing result of three prediction models.

Table 6-3 Predicting live or not in Facebook

		Ground Truth		Precision
		False	True	
Prediction	False	108	49	68.79 %
Result	True	0	78	100 %
Recall		100 %	61.42 %	

Table 6-4 Predicting grow or not in Facebook

		Ground Truth		Precision
		False	True	
Prediction	False	346	80	81.22 %
Result	True	12	224	94.92 %

Recall		96.65 %	73.68 %	

Table 6-5 Predicting merge or not in Facebook

		Ground Truth		Precision
		False	True	
Prediction	False	120	9	93.02 %
Result	True	17	115	87.12 %
Recall		87.59 %	92.74 %	

In the Table 6-4, the accuracy of predicting live or not in Facebook dataset is 79.15%, which is pretty well. When we are predicting whether a community grow or not, we only consider the communities which will survive in next timeslot. As a result, the training set and testing set is different from those in predicting 6 evolution types and in predicting live or not. The accuracy of predicting a community growth or not is 86.10%, which is better than the accuracy of predicting live or not, and much better than predicting 6 evolution types. When we predicting a community merge or not, we only consider the communities that will survive as well. The accuracy of predicting merge or not is 90.04%. Only 10 of 100 will predict incorrectly. It infers that predicting these answers is easier than predicting 6 types. Additionally, the communities which will survive is more predictable since the results of predicting growth or not and merge or not are better than the result of predicting live or not.

To see if there is overfitting problem in our model, we calculate the training accuracy of each prediction. We show the training accuracy in the below Table 6-6. We can see that the training accuracy is almost equal to the testing accuracy in each prediction. As a result, it shows that there is no overfitting problem in our models.

Table 6-6 Testing accuracy and training accuracy of prediction in Facebook

	Testing Accuracy	Training Accuracy
Predict 6 evolution types	64.40 %	63.67 %
Predict live or not	79.15 %	79.55 %
Predict grow or not	86.10 %	86.58 %
Predict merge or not	90.04 %	90.22 %

It is interesting to know which features are important in our prediction model, for we select more than 300 features. The support vector machine algorithm will apply the kernel method, mapping the input features into high-dimensional feature spaces by kernel function, so it is difficult for us to know which algorithm is the most important.

doi:10.6342/NTU201601504

Consequently, we use random forest to train a new model and show the importance of each features. The following Figure 6-1 is the features importance of the random forest model of 6 evolution types in Facebook.

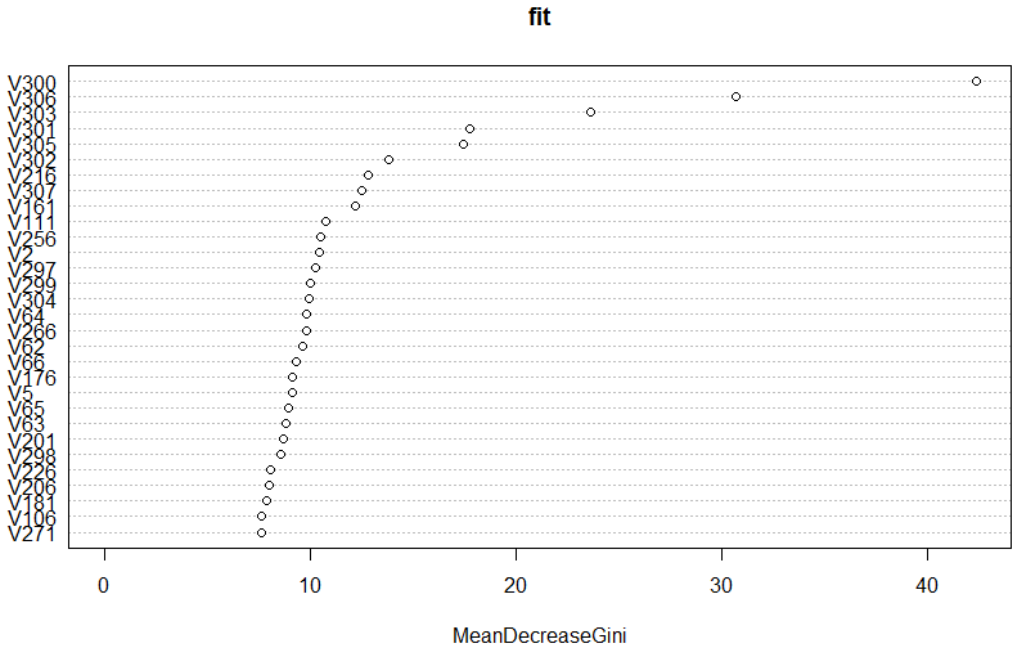

Figure 6-1 Features importance in predicting 6 evolution types in Facebook

As Figure 6-1 shows, the top 7 most important features are feature 300, feature 306, feature 303, feature 301, feature 305, feature 302, and feature 216. Recall that in Section 3.6.3, features from feature 300 to feature 306 are the features about evolution chains. It means that the evolution chains which are detected by Long-term Evolution Method do help a lot in the prediction model of predicting 6 evolution types. We keep move on to plot the features importance of other prediction models in Figure 6-2, Figure

doi:10.6342/NTU201601504

6-3, and Figure 6-4.

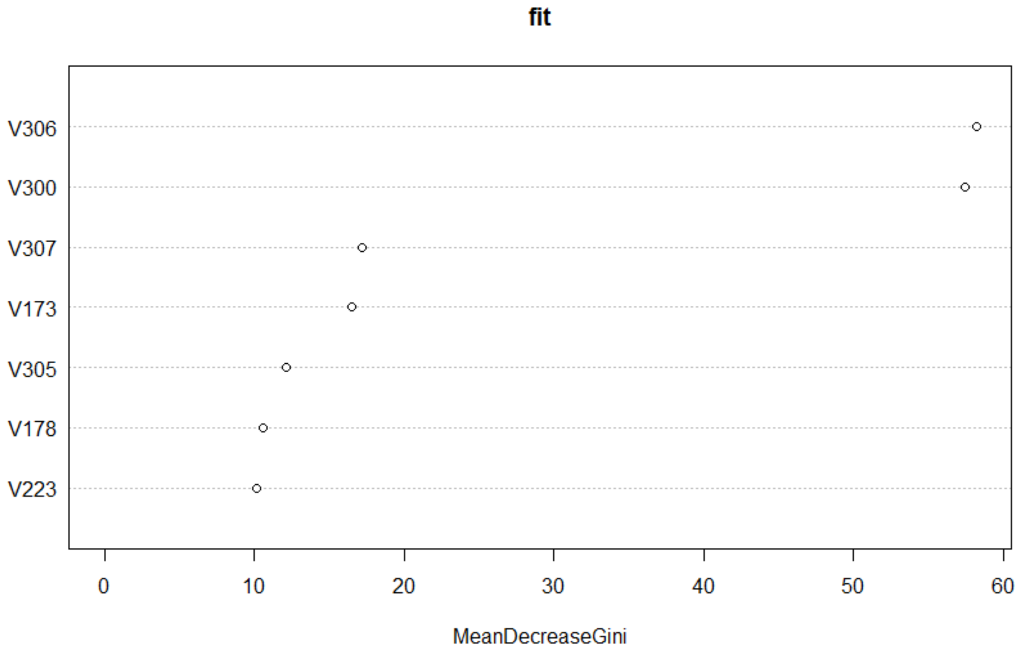

Figure 6-2 Features importance in predicting live or not in Facebook

doi:10.6342/NTU201601504

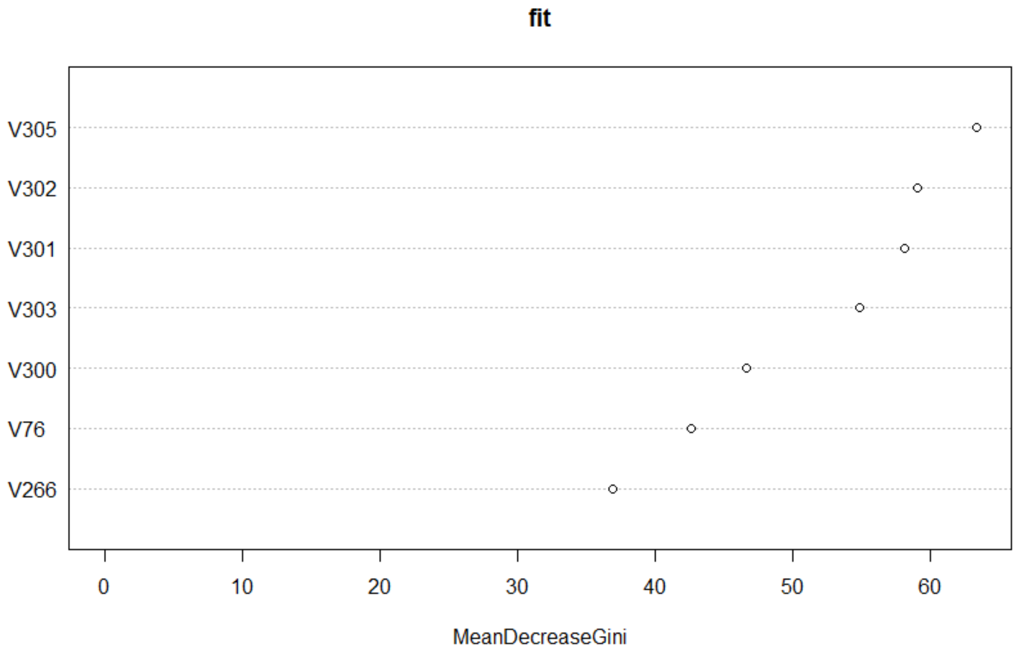

Figure 6-3 Features importance in predicting grow or not in Facebook

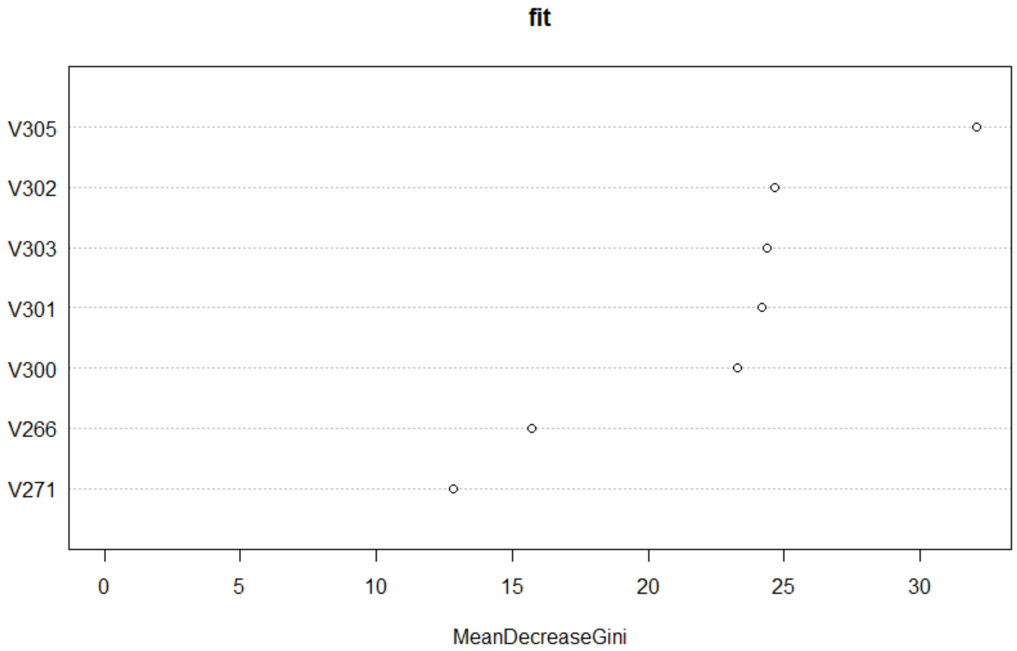

Figure 6-4 Features importance in predicting merge or not in Facebook

We only plot the top 7 most important features in each model. Although in each prediction model, the order of the most important features is not the same with each other, we still can see most of the features in the top 7 important features are features about evolution chain. It shows that our detection algorithm is useful. We can use the features of evolution chain detected by Long-term Evolution Method to build a good prediction model.

6.2 Experiment on DBLP Dataset

In this section, we show the result of prediction on DBLP dataset. We get 83301 communities and 32038 evolution chains in DBLP dataset when we apply our detection algorithm on the dataset in Chapter 5. Similarly, we remove the communities which survive no more than two timeslots. Before we build the predicting models, we sample the training and testing set evenly so as to get an unbiased model. By selecting 80% of data as training set, we train SVM models and test them with the remaining 20% of data. We first show the result of predicting 6 evolution types in Table 6-7.

Table 6-7 Predicting 6 evolution types in DBLP

	Ground Truth					
	Continue	Death	Growth	Merge	Shrink	Split

Prediction Result	Continue	577	135	447	49	18	23
	Death	147	498	115	4	56	7
	Growth	51	5	82	16	37	2
	Merge	0	1	30	673	21	157
	Shrink	146	263	201	98	747	63
	Split	20	34	61	133	33	662

The accuracy of prediction is 57.36%, which is worse than the result of predicting 6 evolution types in Facebook dataset. The DBLP dataset contains more nodes and more edges than Facebook dataset, so the DBLP dataset may be more complicate than Facebook and make the prediction more difficult. However, the result 57.36% is still much better than randomly predicting. We move on to shows the results of predicting live or not, grow or not, and merge or not in following table.

Table 6-8 Predicting live or not in DBLP

		Ground Truth		Precision
		False	True	
Prediction	False	959	335	74.11 %
Result	True	50	722	93.52 %

doi:10.6342/NTU201601504

Recall		95.04 %	68.30 %	

Table 6-9 Predicting grow or not in DBLP

		Ground Truth		Precision
		False	True	
Prediction	False	2880	1397	67.24 %
Result	True	275	1692	86.02 %
Recall		91.28 %	54.78 %	

Table 6-10 Predicting merge or not in DBLP

		Ground Truth		Precision
		False	True	
Prediction	False	1125	215	83.96 %
Result	True	223	1120	83.40 %
Recall		83.46 %	83.90 %	

The accuracy of predicting live or not in DBLP is 81.36%, which is slightly better than the result in Facebook. The accuracy of predicting grow or not and merge or not are 73.22% and 83.67%. The results of predicting grow or not and merge or not are both worse than that in Facebook. It is the same phenomenon with the result of predicting 6 evolution types. The reason is that the DBLP dataset is more complicate and more difficult to predict because DBLP dataset has more nodes and more edges.

It is noted that the result of predicting merge or not is always the best no matter in Facebook or DBLP. “Merge” and “Split” are the types that are very obvious and very easy to realize and to detect. Given a community evolution chain, we can easily tell whether a community type is “Merge” or not. We think that is the answer to the question why the result of predicting a community merge or not is so well.

Again, we list the training accuracy and testing accuracy of prediction in DBLP to see if there is overfitting problem in our model. Table 6-11 shows that the training accuracy is very similar to the testing accuracy so there is no overfitting problem in our models.

Table 6-11 Testing accuracy and training accuracy of prediction in DBLP

	Testing Accuracy	Training Accuracy
Predict 6 evolution types	57.36%	60.26 %
Predict live or not	81.36%	74.91 %

doi:10.6342/NTU201601504

Predict growth or not	73.22%	74.95 %
Predict merge or not	83.67%	85.97 %

Next step, we want to see the importance of features. Because from Facebook dataset we know that the features of evolution chains in very important in models, we want to know that if this is the same with DBLP dataset. We use random forest to build a new model, which is the same process in experiment on Facebook dataset, and plot the importance of features. Figure 6-5 is the features importance in predicting 6 evolution types in DBLP.

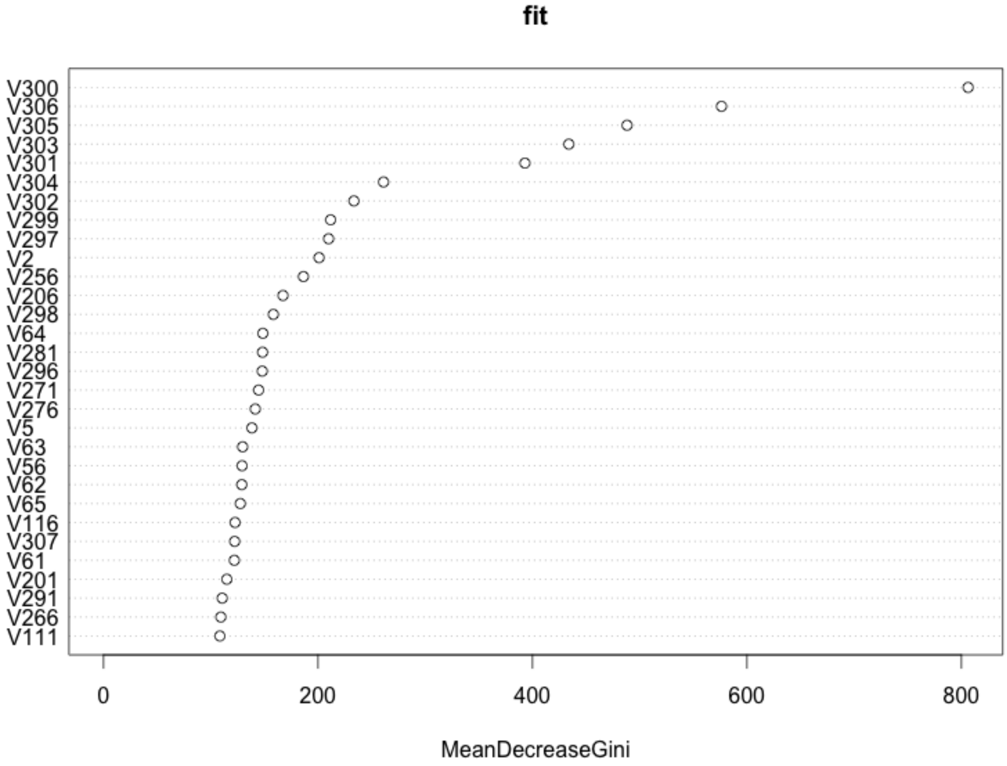

Figure 6-5 Features importance in predicting 6 evolution types in DBLP

The top 7 important features are feature 300, feature 306, feature 305, feature 303, feature 301, feature 304, and feature 302. All of them are the features of evolution chains. It proves that the features of evolution chain do help the prediction. The features importance of other prediction models is plot in the following Figure 6-6, Figure 6-7, and Figure 6-8.

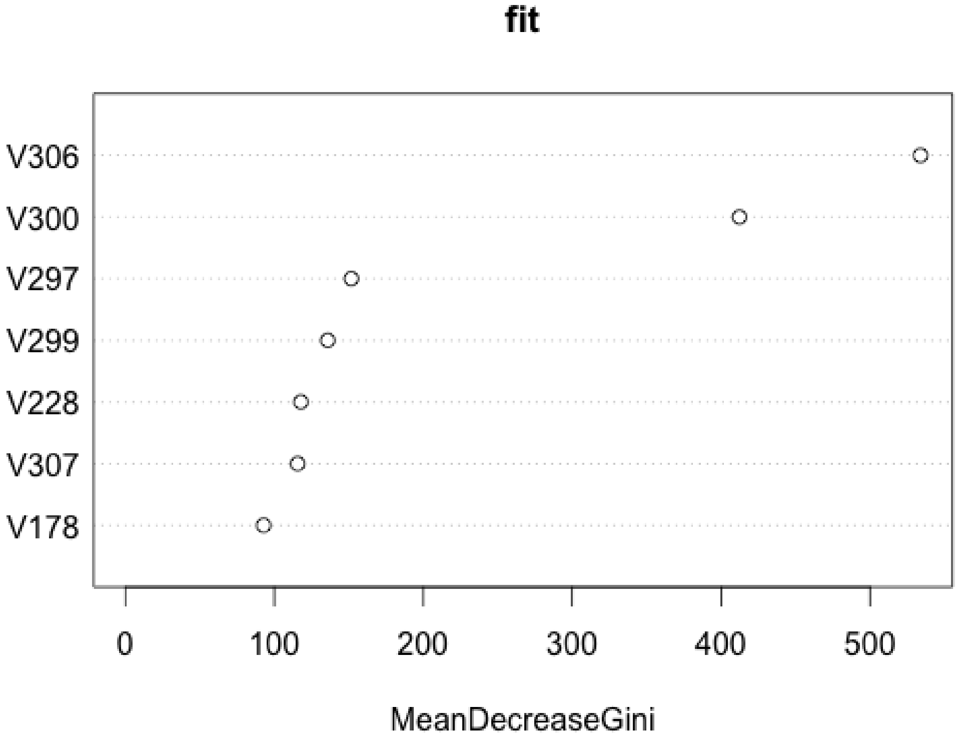

Figure 6-6 Features importance in predicting live or not in DBLP

doi:10.6342/NTU201601504

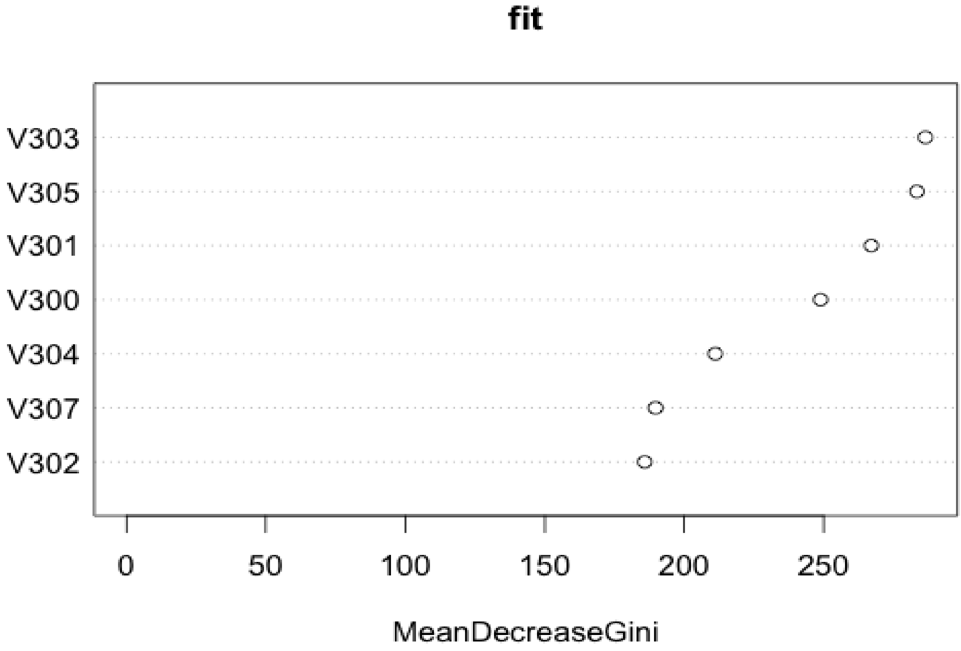

Figure 6-7 Features importance in predicting grow or not in DBLP

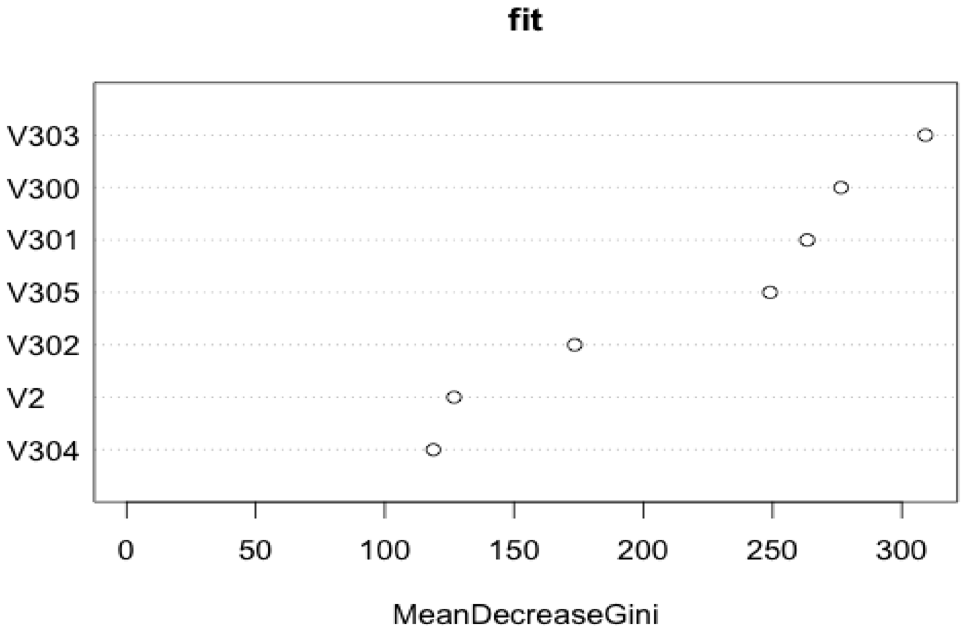

Figure 6-8 Features importance in predicting merge or not in DBLP

doi:10.6342/NTU201601504

In the plots in Figure 6-6, Figure 6-7, and Figure 6-8, we can see that most of the top 7 important features are still features of evolution chains. There is an interesting thing. If we take a look at the top two most important features in predicting 6 evolution types in DBLP and that in Facebook, we can find that they are the same. The top two most important features in predicting 6 types in DBLP and Facebook are feature 300 and feature 306, and the order of these two feature are also the same. The same situation happens in predicting live or not in DBLP and Facebook.

Feature 306 is the most important feature in predicting live or not, which is match to the definition of feature 306. We have mentioned that feature 306 is about how long an evolution chain have lived. The results in Figure 6-2 and Figure 6-6 tell that how long an evolution chain have lived is highly correlated with whether a community in this chain will be still alive in next timeslot. This observation really make sense. If we see an evolution chain which lives many timeslot, we can guess that it will keep alive.

In predicting grow or not, the feature 305 is the most important feature in Facebook and the second important feature in DBLP. However, in predicting grow or not in DBLP dataset, the feature 305 and feature 303 are almost the same importance so we can even say that the 305 is also the most important feature in predicting grow

doi:10.6342/NTU201601504

or not in DBLP dataset. As a result, feature 305 is the most important feature in both DBLP and Facebook dataset when predicting a community growth or not. From these phenomena, it shows that the prediction models are the same is some aspect no matter which dataset we are going to predict. What makes a community live, growth, or other possible evolution may be independent from datasets in some aspect.

To further realize how the features of evolution chains help us predict the community evolution, we compare the original result in DBLP dataset with the result of training model without these features and the result of training model with only these features. The comparison is in Table 6-12.

Table 6-12 Compare the accuracy of prediction with or without chains' feature

	With chains' features	Without chains' features	With only chains' features
Predict 6 evolution types	57.36%	53.03%	51.08 %
Predict live or not	81.36%	80.78%	87.46 %
Predict growth or not	73.22%	71.44%	71. 67 %

doi:10.6342/NTU201601504

Predict merge or not	83.67%	81.74%	85.39 %

It is obvious that the accuracy of prediction model without chains' features are all worse than model with chains' features. The results have no significant differences because in features we mention in Section 3.6.2, it contains some information about the evolution chain though not very much. The prediction result of the model with only evolution chains' features is only slightly worse than original result. In some case, the model with only evolution chains' features is even better. The reason may be we do not pay lots of tension on tuning the argument of SVM. But, from the result, we still can confirm that features of evolution chains do help us predict the community evolution.

From all the result above, we can prove that the evolution of community is predictable since all the predictions is much better than random, no matter in Facebook dataset or in DBLP dataset. Additionally, the evolution chain, which is detected by Long-term Evolution Method, help us to do the prediction, and we get good performance in prediction. We keep doing more prediction to show that our evolution detection algorithm and our definition of evolution types can do better prediction than some other algorithm.

doi:10.6342/NTU201601504

6.3 Compare Prediction Result with SGCI

We compare our evolution detection result with the SGCI algorithm in Chapter 5, so we are also interesting about the evolution prediction of the SGCI, and if Long-term Evolution Method and our defined types are better than SGCI. Considering the DBLP dataset, we replace the evolution types we detected with the SGCI types, and make it as the new prediction target. We use the same features selected in Section 3.6. but eliminate the evolution chains' features. Because the features of evolution chain may not fit SGCI algorithm, we should select some general features like size or density of a community. To do a fairly comparison, we will compare the SGCI result with our prediction result which is trained without evolution chains' features. Other steps are the same as that in Section 5.1. The result is shown in Table 6-13.

Table 6-13 Predicting SGCI types in DBLP

	Ground Truth							
Predict Result		Addition	Chang size	Constancy	Decay	Deletion	Merge	Split
	Addition	71	23	7	19	0	32	18
	Change	5	21	12	7	0	17	10

	size							
	Constancy	10	42	120	38	0	22	0
	Decay	15	38	21	69	0	24	4
	Deletion	4	2	0	5	144	2	4
	Merge	6	9	10	13	0	31	1
	Split	37	44	5	11	0	40	94

The accuracy of predicting 7 SGCI types in DBLP is 49.33%, which is worse than our result of predicting 6 types in DBLP. It is not very clear which detection is better because predicting 7 types should be more difficult than predicting 6 types. So, we go further to compare other prediction: predicting whether a community will live and whether a community will merge. With the SGCI types, a community will live if its evolution type is not “Decay”, and a community will merge if its evolution type is “Merge”. The result is in Table 6-14 and Table 6-15.

Table 6-14 Predicting live or not according to SGCI in DBLP

		Ground Truth		Precision
		False	True	
Prediction	False	1388	719	65.88 %

Result	True	273	980	78.21 %
Recall		83.56 %	57.68 %	

Table 6-15 Predicting merge or not according to SGCI in DBLP

		Ground Truth		Precision
		False	True	
Prediction	False	365	267	57.75 %
Result	True	146	301	67.34 %
Recall		71.43 %	52.99 %	

The accuracy of predicting live or not and predicting merge of not according to SGCI types are 70.48 % and 61.72 %. We use a table to show the comparison between our prediction result and the predicting result of SGCI.

Table 6-16 Compare the accuracy of prediction according to different types

	According to our types	According to SGCI types
Predicting multiple types	53.03 %	49.33 %
Predicting live or not	80.78 %	70.48 %
Predicting grow or not	71.44 %	-

Predicting merge or not	81.74 %	61.72 %

Note that we do not predict a community grow or not because if a community will grow, the evolution type of this community according to SGCI may be “Change Size” or “Addition”. However, a community with evolution type “Change Size” according to SGCI may be “Growth” or “Shrink” in our types’ definition, so we skip this prediction. It is obvious that our accuracy is higher, especially in predicting merge or not. We may attribute the worse prediction in SGCI types to the difficulty of prediction more types than us. However, the prediction of live or not and merge or not is very fair. We get much better accuracy in the two prediction. We can say that our evolution types, which come from the evolution chain we detected and transfer by our method, is more preventative since we get better performance in evolution prediction.

Chapter 7
Conclusion

In this work, we use the algorithm proposed by Weixu Lin to do community evolution detection and community evolution prediction. We first build a synthetic data generator based on Stochastic Block Model to test Weiux's Long-term Evolution Method. Our synthetic data generate can simulate the core structure in a community and noise in community evolution. To give a good measure to judge the algorithm, we suggest to use normalize mutual information as a measure of community evolution chain detection. Based on the synthetic data and the measure, we officially verify Long-term Evolution Method is correct, and we analysis the two arguments of the algorithm so that we get the knowledge of how to set up p-core and threshold to get better performance. When there is core structure in communities, selecting p-core to catch the

doi:10.6342/NTU201601504

core nodes and lower threshold can improve the detection. After we get some knowledge of Long-term Evolution Method, we perform the algorithm on DBLP and Facebook dataset. To make the detection result more comprehensive and compare with other detection algorithm, we define our evolution types and propose a method to transfer evolution chains detected by Long-term Evolution Method to the evolution types we defined. This method can be adjusted by two factors which make the method flexible. We provide a case study on Facebook dataset according to our evolution types, and shows that we can match some events happened to Facebook with the changes in evolution types we defined. By analyzing the evolution types we detected, we can get more information in datasets. After we have done the detection on Facebook and DBLP dataset, we move on to do community prediction on these datasets. We use Libsvm with R to build a classification model. We select 4 kinds of features including features about communities, features about the difference between previous communities, features about the evolution chains, and features about the previous evolution types, and we get totally 307 features as our input features. The prediction result is pretty well. From the result of the prediction, we show that the community evolution is predictable and the evolution chain detected by Long-term Evolution Method help a lot in prediction. We

doi:10.6342/NTU201601504

also compare the prediction result with the SGCI algorithm, by replacing our evolution types with SGCI types. The prediction according to our types is much better than that according to SGCI types, which means that our evolution types is more representative and so that more predictable than SGCI. Our work's contributions are we propose a synthetic data generator which have evolution ground truth and core structure, provide a measure for accuracy of evolution chain, define our own evolution types, give a method to transfer evolution chain to our types, provide features we selected for community evolution prediction, and finally prove the evolution is predictable with the help of evolution chains' features.

Community evolution prediction is still a growing topic, so there are still lots of works can be done in the future. The main future work of community evolution prediction is finding solutions to get better prediction result. For example, Recurrent Neural Network (RNN) can be a useful algorithm that help us get better prediction result. RNN can learn the evolution of features so it may help us predict evolution. Deep learning is a useful skill. It is start to be used in many places. For example, Google train a Go AI with deep learning and win 4 of 5 games in the contest with Lee Sedol. Deep learning may also help the community evolution prediction. It can be a great

doi:10.6342/NTU201601504

future work to predict evolution with deep learning skill. We select lots of feature about a community to predict the communities' types. However, there might be some features which are not as general as features like size or density of a community. We can call these features as "latent vector". The concept of latent vector has been used in recommendation system. Users may contain latent vector which impact their rating so we can use latent vectors of users to predict their rating. Maybe communities also contain their own latent vector which impact their evolution. Finding communities' latent vectors and using them to predict evolution are interesting future works.

doi:10.6342/NTU201601504

Reference

[1] Girvan, Michelle, and Mark EJ Newman. "Community structure in social and biological networks." Proceedings of the national academy of sciences 99.12 (2002): 7821-7826.

[2] Fortunato, Santo. "Community detection in graphs." Physics reports 486.3 (2010): 75-174.

[3] Weiux Lin. "On the evolution of communities in large social networks," MS thesis, National Taiwan University, 2015.

[4] Holland, Paul W., Kathryn Blackmond Laskey, and Samuel Leinhardt. "Stochastic blockmodels: First steps." Social networks 5.2 (1983): 109-137.

[5] Karrer, Brian, and Mark EJ Newman. "Stochastic blockmodels and community structure in networks." Physical Review E 83.1 (2011): 016107.

doi:10.6342/NTU201601504

[6] Viswanath, B., Mislove, A., Cha, M., & Gummadi, K. P. (2009, August). On the evolution of user interaction in facebook. In Proceedings of the 2nd ACM workshop on Online social networks (pp. 37-42). ACM.

[7] DBLP dataset, http://dblp.uni-trier.de/

[8] Gliwa, B., Saganowski, S., Zygmunt, A., Bródka, P., Kazienko, P., & Kozak, J. (2012, August). Identification of group changes in blogosphere. In Proceedings of the 2012 International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2012) (pp. 1201-1206). IEEE Computer Society.

[9] Chang, Chih-Chung, and Chih-Jen Lin. "LIBSVM: a library for support vector machines." ACM Transactions on Intelligent Systems and Technology (TIST) 2.3 (2011): 27.

[10] Takaffoli, Mansoureh, Reihaneh Rabbany, and Osmar R. Zaiane. "Community evolution prediction in dynamic social networks." Advances in Social Networks Analysis and Mining (ASONAM), 2014 IEEE/ACM International Conference on. IEEE, 2014.

[11] Saganowski, S., Gliwa, B., Bródka, P., Zygmunt, A., Kazienko, P., & Koźlak, J. (2015). Predicting community evolution in social networks. Entropy, 17(5), 3053-3096.

doi:10.6342/NTU201601504

[12] Strehl, Alexander, and Joydeep Ghosh. "Cluster ensembles---a knowledge reuse framework for combining multiple partitions." Journal of machine learning research 3.Dec (2002): 583-617.

doi:10.6342/NTU201601504