應用於大型電信社群網路中核心人物搜尋的雲端運算平行處理演算法

研究生：蔡勝文　　　　　　　　　　　　　　指導教授：高榮鴻

國立交通大學電信工程研究所碩士班

摘　要

在本篇論文中，我們提出一個應用於大型電信社群網路中核心人物搜尋的雲端運算平行處理演算法。首先我們利用分治法 (Divide-and-conquer)，此一平行化的方法來從大型電信社群網路中找出 k-truss 定義下的群體，然後利用特徵向量中心度 (Eigenvector Centrality) 來定義群體中的核心人物 (Alpha user)。除此之外，我們提出了建立在最短路徑演算法之上的核心人物平行排序演算法，來排序、比較不同群體的核心人物重要性。我們也提出了一個具有偵測以及拆解功能的平行演算法來處理位於電信社群網路中的超大群社群。本篇論文中，除了用人工資料來進行測試實驗之外，我們也將此演算法實際應用分析於由電信公司所提供的通聯記錄所建立的真實大型電信社群網路中。

Searching for and Ranking Alpha Users in Massive Telecom Social Networks with MapReduce

Student : Sheng-Wen Tsai Advisor : Rung-Hung Gau

Institute of Communications Engineering
National Chiao Tung University

Abstract

In this thesis, we propose novel MapReduce algorithms to search for and rank alpha users in massive telecom social networks. We first apply the principle of divide-and-conquer to find out trusses in a social graph and then use the eigenvector centrality to identify alpha users in the trusses in parallel. In addition, we propose ranking alpha users in distinct trusses based on their shortest path coverage. Furthermore, we propose novel algorithms to efficiently detect and decompose giant components in a social graph. In addition to synthetic social networks, we have used the proposed algorithms to analyze real smart phone social networks that are created based on call detail records collected by a telecom operator.

誌 謝

時光飛逝，一轉眼碩士生活即將結束，這兩年來付出的努力以及師長們的指導，讓我的研究成果得以順利地在此篇論文呈現。首先，我要感謝擔任口試委員的李程輝、謝宏昀以及趙禧綠教授，在口試時針對我論文提出了許多問題及我從未考慮的方向，讓我了解到有什麼細節需要再改進、以及可能的延伸方向。再來，我要感謝指導教授高榮鴻老師，帶領我從事雲端運算領域上的研究，除了設計有效率的平行演算法外，也實作出實地使用的大型程式。讓我成為一具有研究及實作能力的碩士生。在這段期間，和老師討論問題時，老師總是用輕鬆地方式和我討論問題，並引導我自行尋找答案，而遇到研究上有不順利的時候，也總以鼓勵代替責罵，引領我去克服困難，這裡再次感謝老師兩年來的指導。

加入 WMCN 實驗室的這兩年，是一段難忘的回憶，實驗室認真不失歡樂的氣氛，總是能讓我們能輕鬆且有效率地進行研究。感謝子強當我遇到演算法上的問題時，總是不吝情地分享你獨到的見解與幫助。感謝佩佩處理實驗室大大小小的各種事務，讓我們不必擔心許多瑣碎的事情以專心在研究上，稱呼妳一聲實驗室大姊也不為過。感謝實驗室開心果阜達在我們研究煩悶的時候，總會說一些冷笑話來幫大家醒醒腦，雖然大部分都戳不中我的笑點，但仍非常感謝你的用心。感謝實驗室學弟阿秋、胖胖、仁毅在我們碩二趕論文以及口試的時候給了我們許多幫助，與你們共享的實驗室晚餐歡樂時光，不管多久以後也會是一段難以忘懷的時間，同時也祝福你們之後的研究能順順利利並往自己的目標邁進。

除了實驗室的成員，也感謝所有 716 游泳團的成員們，碩士兩年每天晚上幾乎不間斷的 1000 公尺游泳、SPA 和談天說地的時光，除了讓我擁有強健的體力、舒緩緊張的心情外，也讓我對人生規劃更有想法。除此之外，感謝我的室友們一路上互相支持，最後大家也都順利完成碩士學位，謝謝你們。而最感激的，還是爸爸、媽媽及妹妹，一路走來經歷了許多事，但我們也都一一克服，謝謝你們的支持，這是我努力的最大原動力，除此之外，我想跟你們說聲謝謝和一句我愛你們，希望我能讓你們感到驕傲。最後，此篇論文謹獻給所有幫助過我的家人、朋友以及師長們。

蔡勝文謹誌 于國立交通大學 新竹

中華民國 一〇二 年 七 月

Contents

Chinese Abstract i

English Abstract ii

Acknowledgement iii

Contents iii

List of Tables iv

List of Figures v

1 Introduction 1

2 Related Work and MapReduce 3

2.1 Related work 3

2.2 Truss definition 4

2.3 Eigenvector centrality [7] 4

2.4 The MapReduce platform [9] [10] 6

3 System model and MapReduce algorithm 10

3.1 Pre-processing 12

3.2 K-truss clustering 12

3.3 Decomposing large communities 13

3.3.1 Statistic algorithm 14

3.3.2 Decomposing Algorithm 14

3.4 Influential ranking 19

3.4.1 Eigenvector centrality computing 20
3.4.2 Influence ordering between communities 20

4 Experimental Results 25

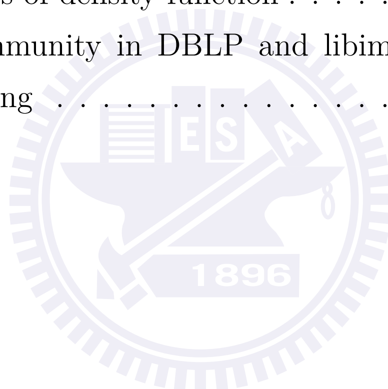
4.1 Datasets and system settings 25
4.1.1 Datasets 25
4.1.2 System settings 26
4.2 K-truss clustering 26
4.2.1 Performance of K-truss clustering 26
4.3 Decompose giant component 28
4.4 Result of influential rankings 31
4.4.1 Another social network 34

5 Conclusion 41

Bibliography 42

List of Tables

4.1 The computers' specification of the cloud 26
4.2 Synthesis data generated from Erdős-Rényi model with $p = 1.5e^{-5}$ 28
4.3 The parameters of density function 29
4.4 The giant community in DBLP and libimseti dataset after 3-truss clustering 38



List of Figures

2.1 K-trusses of graph with randomly assigned color, vertices and edges not in trusses are black. 5
2.2 A figure of HDFS structure from [10] 8
2.3 A simplified view of word count example, illustrating the function of mapper and reducer. 9

3.1 System modules 10
3.2 System workflow 11
3.3 Example graph for illustrating statistic algorithm 17
3.4 Initial records 17
3.5 Statistic algorithm step1 mapper 18
3.6 Statistic algorithm step1 reducer 18
3.7 Statistic algorithm step2 mapper 18
3.8 Statistic algorithm step2 reducer 19
3.9 Statistic algorithm step3 19
3.10 A procedure for calculating shortest path ranking algorithm. . 24

4.1 The result of 3-trusses clustering of a real data, each member of 3-trusses subgraph is randomly assigned color. And the members not in any community is black. 27
4.2 The result of 3-trusses clustering of a real data, non-trusses members have been deleted from the graph. 28
4.3 The experimental results of synthesis data in our own clusters 29

4.4 The experimental results of synthesis data in a Chunghwa Telecom's private server 30
4.5 The probability density function of actual call duration distribution, exponential distribution, and log-normal distribution 31
4.6 The decomposing result of actual distribution with five times the average 32
4.7 The decomposing result of exponential distribution with five times the average 32
4.8 The decomposing result of log-normal distribution with five times the average 33
4.9 3,4-truss community and its corresponding truss alpha user . 35
4.10 Comparison of the sequence of each truss alpha user in different community and the corresponding size of truss-component in synthesis data with five times the average 36
4.11 Comparison of the sequence of each truss alpha user in different community and the corresponding size of truss-component in real world data 36
4.12 the result of 3-trusses clustering in DBLP dataset 37
4.13 The edge weight probability density function comparison of DBLP and Libimseti dataset 38
4.14 The decomposing result of Libimseti dataset 39
4.15 The decomposing result of DBLP dataset 39
4.16 Comparison of the sequence of each truss alpha user in different community and the corresponding size of truss-component in DBLP dataset 40

Chapter 1

Introduction

In this thesis, we study the problem of finding out all alpha users and the corresponding communities in mobile telecommunications social networks. Online social network analysis [1] draws a lot of attention recently and has a variety of applications including social marketing. According to [2], alpha users of a social network tend to be highly connected users with exceptional influence to the other thought-leaders. In addition, modern social marketing campaigns attempt to use alpha users as spokespersons in marketing and advertising [2]. As [3] [4] [6], we study telecommunications social networks extracted from a large amount of Call Detail Records (CDRs). In telecommunication company strategy, in addition to identify the alpha user finding its own corresponding community is also very important. It helps telecom operator to recognize who can be affected by this alpha user, and these users are considered as the potential marketing objective.

In principle, given the adjacency matrix [5] of a social graph, one could identify alpha users by calculating the eigenvector associated with the maximum eigenvalue [7]. However, this approach is not scalable and does not identify the corresponding community for an alpha user. In addition, to the best of our knowledge, for the adjacency matrix of an arbitrary graph, there is no formal proof for the existence of strictly positive eigenvectors in the literature. Let $k \geq 3$ be an integer. A k-truss [8] of a graph is a component

of the graph such that each edge in the component belongs to at least $(k-2)$ triangles in the component. In [17], based on the theory of non-negative matrix, prove that for the adjacency matrix of an arbitrary k-truss, there exists a strictly positive eigenvector associated with the maximum eigenvalue.

To address the scalability issue and to find out the communities, we propose enumerating all trusses in the social graph and then finding out the alpha users in all trusses in parallel. The identified alpha users are called truss alpha users, since trusses are used to efficiently identify the alpha users and the corresponding communities in the social graph. Besides, there is usually a giant community within any large social network, and this kind of giant community is too large to analyze. To deal with the giant community problem, we propose a MapReduce algorithm to decompose a giant component into smaller components. We also address the issue of ranking alpha users in different trusses. In particular, we propose using the shortest path coverage to rank alpha users. Furthermore, we have implemented MapReduce [9] Java programs to analyze massive networks.

The rest of the thesis is organized as follows. In Chapter 2, we briefly introduce related work. In Chapter 3, we propose a MapReduce framework to efficiently search for and rank truss alpha users. The proposed framework exploits graph theory and linear algebra. In Chapter 4, we present some numerical results. Our conclusions are included in Chapter 5.

Chapter 2

Related Work and MapReduce

In this chapter, we introduce related work and the MapReduce platform. In particular, MapReduce is the de facto standard for processing big data in academic and industry [11] [12] [13].

2.1 Related work

Newman [15] proposed a fast community detection algorithm based on the idea of modularity. Cohen [8] introduced some MapReduce graph algorithms for social network analysis. In particular, instead of cliques, he used trusses for community detection. For finding trusses in e-mail social networks, Gau, Hsieh, Tsai, and Cheng [16] compared the performance of cloud computing programs with that of conventional computer programs. Wang and Chen [18] proposed fast algorithms for truss decomposition in massive networks. In this thesis, we focus on finding out alpha users and the corresponding communities in social networks.

PageRank [19] is a well-known link analysis algorithm for the World Wide Web. While PageRank could be used to identify alpha users in principle, it is not designed for finding out the corresponding communities. Weng, Lim, Jiang, and He [20] proposed TwitterRank for identifying influential users of micro-blogging services. In particular, TwitterRank takes into consideration

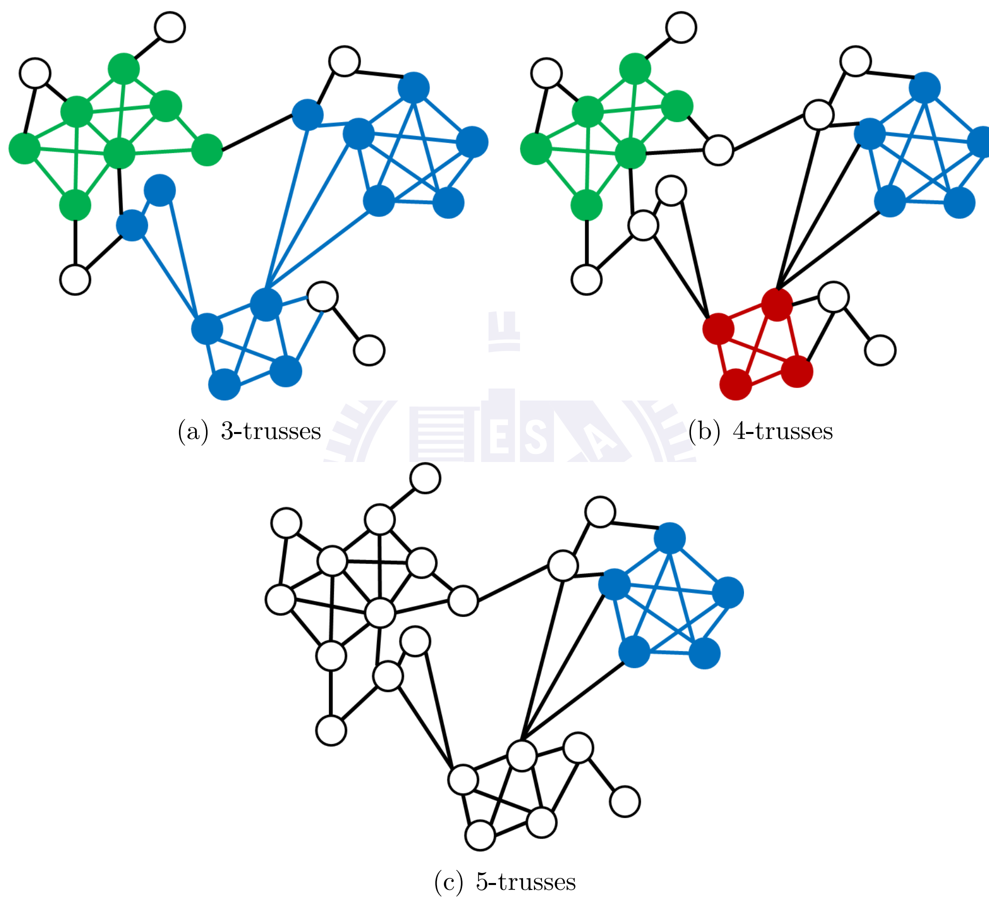
both topic similarity between users and link structure. Instead of micro-blogging services, we study mobile telecommunications social networks in this thesis. In addition, the proposed approach in the thesis could be easily modified to use PageRank to identify alpha users in each truss.

2.2 Truss definition

As we mention in Chapter 1, a well-defined community structure has the advantage of target marketing in business strategy. There are three desired features for a community structure. First, the structure has to be well-defined in mathematics. Second, the link between two users in the same community should be *stronger* than the link between users in different communities. Third, this structure can be efficiently discovered by parallel algorithms. As the data size increases, It is necessary to utilize parallel computing to increase the efficiency. Based on the above arguments and previous works we studied, K*-truss* is one of the most appropriate subgraphs for community detection. K-truss is a relaxation of clique definition. In particular, a K-truss is a maximal connected subgraph such that each edge in the subgraph belongs to at least $(K-2)$ triangles in the subgraph. [8]. For example, in a 4-truss every edge is contained in at least 2 triangles. Figure 2.1 shows 3-trusses, 4-trusses,and 5-trusses. Note that the problem of finding the maximum clique in a graph is NP-hard. Namely, it can't be solved in polynomial time.

2.3 Eigenvector centrality [7]

Eigenvector centrality is a measure of the influence of a vertex in a graph. It starts from a concept that a vertex can be considered as *important* if and only if it also has lots of *important* neighbors. Google's PageRank is a variant of the eigenvector centrality measure. Now we give an explanation of using adjacency matrix to get eigenvector centrality. Consider a graph $G := (V, E)$.



(a) 3-trusses (b) 4-trusses

(c) 5-trusses

Figure 2.1: K-trusses of graph with randomly assigned color, vertices and edges not in trusses are black.

Let A be the adjacency matrix. Note that $a_{ij} = 1$ if vertex x_i connects to vertex x_j, and $a_{ij} = 0$ otherwise. Let λ be the Perron-Frobenius eigenvalue of A. Let $M(v)$ be a set that is composed of the indexes of vertices adjacent to vertex v. Let x_i be the eigenvector centrality of vertex i. In particular,

$$x_i = \frac{1}{\lambda} \sum_{j \in M(i)} x_j$$

$$= \frac{1}{\lambda} \sum_{j=1}^{N} a_{ij} x_j \tag{2.1}$$

Define a vector $x = (x_1, x_2, \ldots, x_n)$, where $n = |v|$. Then, based on the above equation, we have

$$\mathbf{x} = \frac{1}{\lambda} \mathbf{A} \mathbf{x} \tag{2.2}$$

$$\mathbf{A} \mathbf{x} = \lambda \mathbf{x} \tag{2.3}$$

In [17], it has been proved that for an arbitrary K-truss G, the adjacency matrix A is primitive and therefore there exists a strictly positive eigenvector associated with λ. As the result, the k_{th} element of the eigenvector we get represents the centrality value of the vertex k in the graph. A common algorithm to find the largest eigenvalue is the power method which is also used in this thesis.

2.4 The MapReduce platform [9] [10]

In order to process a large amount of call data records from a telecom company, we need a platform that could efficiently store big data, and run parallel algorithms to analyze big data. One of the well-known methods

is MapReduce which is a programming model proposed by Google for processing large data sets with a parallel, distributed algorithm on a cluster of computers. A popular free implementation is Apache Hadoop which is an open-source software framework. It supports the running of applications on large clusters of commodity hardware with proper scheduling mechanism and fault tolerance techniques. Since large clusters are built by commodity hardware, Hadoop is very cost efficient on storing and processing big data.

The Hadoop Distributed File System (HDFS) is an master/slave architecture of distributed data storing in Hadoop framework. There are two important components in HDFS, *NameNode* and *DataNode*. Figure 2.2 illustrates the HDFS.

- **NameNode** is a master server responsible for storing meta-data about HDFS and managing all the requests to files by the clients. For example, It executes file system operations : opening, closing and renaming files and directories. Meta-data contains several file information like the file namespace, number of replications, and the address of files. The information controls and organizes the process on HDFS.

- **DataNode** In HDFS data storing, a file is divided into one or more blocks and these blocks are stored in DataNode. A DataNode is also responsible for serving requests from file system's client, and executing instructions from the NameNode.

In the **MapReduce** framework, a key-value pair is an essential data structure. Each input data is assigned a key and a value, which could be primitives (integers, strings, floating values …) or complex structures like lists, tuples, etc. For the famous *word count* example, keys are words, and values corresponded to the number of occurrences. In MapReduce programming, developers have to define a *mapper* (map function) and a *reducer* (reduce function) with following characteristics :

map : $\langle key_1, value_1\rangle \rightarrow [\langle key_2, value_2\rangle]$

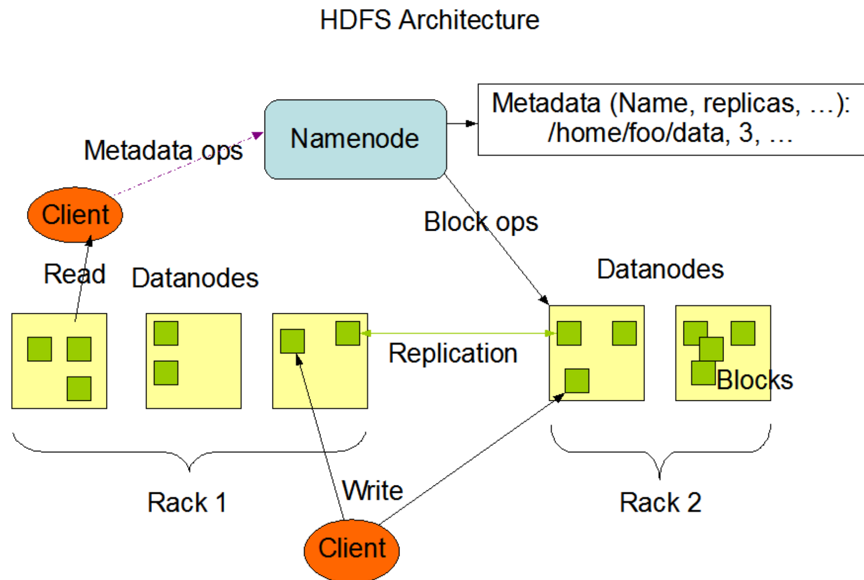


Figure 2.2: A figure of HDFS structure from [10]

reduce: $\langle key_2, [value_2] \rangle \rightarrow [\langle key_3, value_3 \rangle]$

The mapper is applied to every key-value pair in the input which is originally stored on the underlying distributed file system. The result of mapper is an arbitrary number of intermediate key-value pairs, and then these pairs will be sorted and grouped by the same key, finally be passed to reducer(reduce function) as input. This step is called *shuffle* which can strongly affects the efficiency of MapReduce tasks. After shuffle, the reducer starts to apply user-defined function to every intermediate key and its related values. In the end, the output of reducer will be written back to HDFS. For each MapReduce process, there are M map tasks and R reduce tasks where M is the number of divided parts of input and R is the number of reducers.

A task which can be divided and conquered is suitable for processing on MapReduce platform. There is a famous example of word count in Figure 2.3. It counts the number of occurrences of words. In mapper, for each input it emits the word as key and the integer one as value. After shuffle step, all

values with the same key are collected in the reducer. The only task which reducer has to do is summing up all the counts of words. In the end, reducer emits the final result as output.

It is worth to mention that a *combine* task (combiner) is widely use in MapReduce to increase the efficiency and reduce network loading. The combiner can be considered as the helper of reducer, its main objective is to decrease the size of output from mapper. More specifically, It is operated locally after the processing of a map task, its main objective is to merge lots of results from each mapper into one message. This would be a great difference while processing vast amounts of data. We also use this method to increase the efficiency and speed up the implementation of algorithm in this thesis.

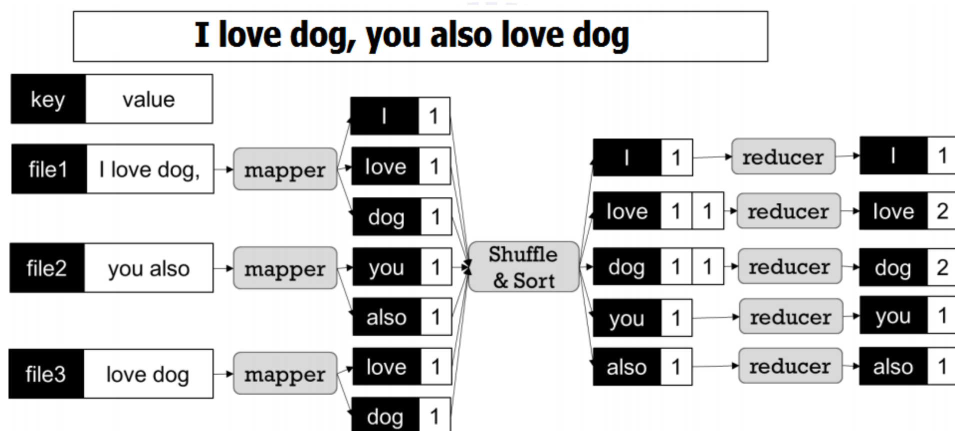


Figure 2.3: A simplified view of word count example, illustrating the function of mapper and reducer.

Chapter 3

System model and MapReduce algorithm

In order to find influential users in giant telecom networks, we propose and implement a MapReduce based solution to deal with large amount of call data records. As shown in Figure 3.1, this model consists of four modules, which are **Pre-processing**, **K-truss clustering**, **Decomposing component**, and **Influential ranking**. The overall flow chart for the proposed approach is shown in Figure 3.2. Furthermore, we elaborate on the four modules in this chapter.

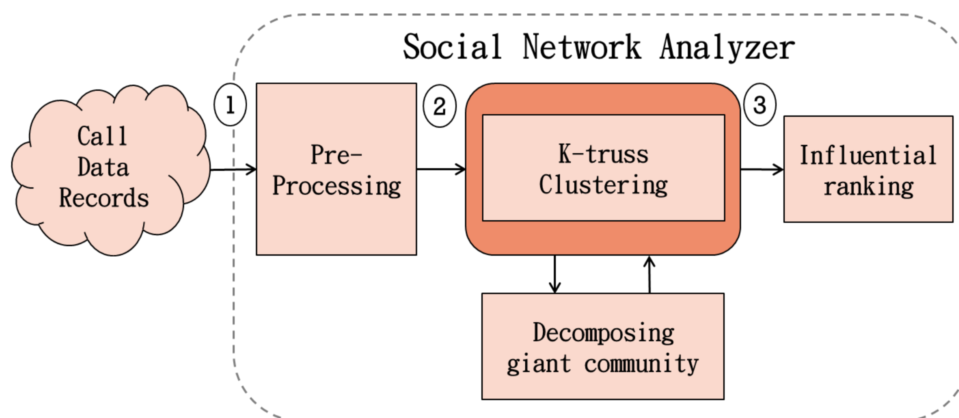


Figure 3.1: System modules

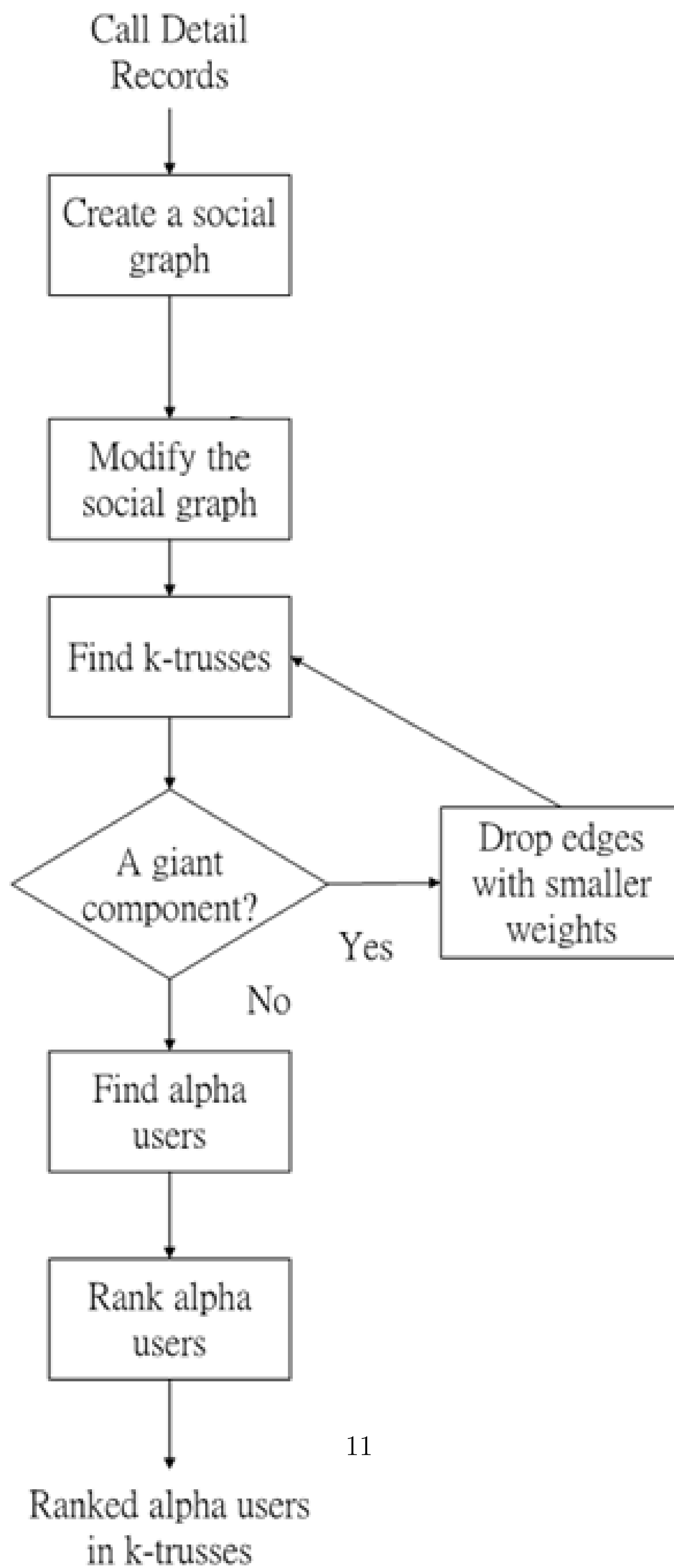


Figure 3.2: System workflow

3.1 Pre-processing

The main objective of the pre-processing module is to prepare graph input data with proper format for the K-truss clustering module. Basically, call d records (CDRs) contain lots of information. To efficiently test our algorithm, irrelevant information has to be filtered out. Each input record is represented by (s, r, w), where s and r correspond to two users/smart phones, while w is the call duration from user s to user r. Based on the input records, we create a weighted graph in which a vertex corresponds to a user. In addition, the edge (u, v) exists if the accumulated call duration between user u and user v exceeds a predetermined threshold. Further more, the weight of the edge (u, v) is the accumulated call duration between user u and user v. The MapReduce algorithm for creating the social graph based on CDRs is very similar to the WordCount tutorial. Therefore, we omit the details.

3.2 K-truss clustering

In order to run K-truss clustering algorithm on Hadoop platform, we implement seven MapReduce algorithms according to Cohen's work [8]. The goal of the first four algorithms is to find triangles in the graph. To achieve that, an iteration is used. The input is a list of edge records and an edge record is represented by (v_1, v_2, w), where v_1 and v_2 are users, while w is the call duration between the two users. Each output record of the K-truss clustering module is represented by $((v_1, v_2), (v_2, v_3), (v_3, v1))$, where v_1, v_2, and v_3 form a triangle. The fifth MapReduce algorithm is responsible for checking if every edge in the list of triangles has sufficient support. Let $k \geq 3$ be an integer. For a graph G, a maximal component H is said to be a K-truss if each edge in H exists in at least $(k-2)$ triangles in H. After the step of checking supports, two MapReduce algorithms are used to find components in the social graph. Initially, each component consists of a single vertex. In each iteration, two adjacent components are merged. The leader

of the merged component is the leader with the smallest index among the two leaders in adjacent components. In the end, every edge belongs to a specific leader. If two vertices have the same leader, they belong to the same component. Namely, they are in the same "community". The output of this module is a list of edges with format $((v_1, v_2), l)$, where l is denoted as the common leader between v_1 and v_2. The procedure of finding K-truss can be found in Algorithm 3.1. and details can be found in Cohen's work [8].

Algorithm 3.1 K-truss clustering according to [8]

Input: The undirected and unweighted graph G of list of edge records with format : "vertex1 vertex2 call-duration"

Output: The undirected and unweighed K-truss graph G_k of list of edge records with format : "vertex1 vertex2 leader"

1: **repeat**
2: Augment the edges with vertex valences (MapReduce Job 1&2);
3: Enumerate triangles (MapReduce job 3&4);
4: For each edge, records the number of triangles containing that edge. Then keep only the edges with sufficient support $K-2$ (MapReduce 5);
5: **until** If step 4 didn't drop any edges
6: Find the remaining graphs components G_k with two MapReduce jobs repeating until finishing recognized components (MapReduce 6&7)
7: **return** G_k;

3.3 Decomposing large communities

One issue in social network analysis is the existence of giant communities. In our own experiment, it happens while the graph is extremely large i.e., in giga-byte scale. From the marketing aspect in real world, the giant community is too big to be useful in social marketing. The appropriate size of community is under 300. So in this section, we will describe the algorithm we use to decompose a giant community into several smaller components.

3.3.1 Statistic algorithm

Before running the decomposing algorithm, it is necessary to collect some statistic information on the results of K-truss clustering. For example, the community size is required to check if a giant community exists or not. The mean of the edge weights is also an important information for the decomposing algorithm. In this subsection, we present a two steps MapReduce algorithm for calculating the community size, the mean of edge weights, and the largest edge weight in a community. For each input edge (v_1, v_2) to the Map phase in the first step, mapper emits v_1 as the key of an output record and v_2 as another one. They both pass an integer count 1 and the half input edge weight as value. For the pairs with key (vertex) v_1, reducer sums all counts as *sumN* and all weights as *sumW*. Practically, sumN is the degree value of v_1. In addition, reducer also finds the larges edge weight. Algorithm 3.2 shows the details of algorithm. In the second step of algorithm, mapper merely pass leader as key and add an integer count 1 to the output value. After shuffle phase, all pairs with same key (leader) are collected to the same reducer. Reducer sums all counts as the size of community, all sumWs as the total number of edge weights, all sumNs as the total number of edges, and finds the largest edge weight. Algorithm 3.3 shows the pseudo codes of this algorithm, and Figure 3.3 to Figure 3.9 illustrate whole algorithm in an example.

3.3.2 Decomposing Algorithm

In this subsection, we introduce an algorithm to decompose a giant community. A giant community can be considered as a composition of several sub-communities. The connections between sub-communities are much weaker than the connections inside each sub-community. As a result, if we can find the relatively stronger connected structure, then the giant community is very likely to be decomposed. We first analyze the probability

Algorithm 3.2 Statistic algorithm setp1

The mapper emits an intermediate key-value pair for each vertex and its connected edges information. The reducer sums the number and the total weight of the vertex connected edges, and finds the maximum edge weight.

```
1: class Mapper
2:     method Map(string [vertex_1, vertex_2], string [leader, weight])
3:         v_1 ← vertex_1
4:         v_2 ← vertex_2
5:         n ← count 1
6:         w ← weight/2
7:         Emit(vertex v1, set [leader, n, w])
8:         Emit(vertex v2, set [leader, n, w])

1: class Reducer
2:     method Reduce(vertex v, sets [[leader, n_1, w_1], [leader, n_2, w_2], ...])
3:         sumN ← 0
4:         sumW ← 0
5:         maxWeight ← 0
6:         for all set [leader, n, w] ∈ sets [[leader, n_1, w_1], ...] do
7:             sumN ← sumN + n
8:             sumW ← sumW + w
9:             if w > maxWeight then
10:                maxWeight ← w
11:        Emit(vertex v, sets [leader, sumW, sumN, maxWeight])
```

Algorithm 3.3 Statistic algorithm step2

The mapper emits an intermediate key-value pair for each community and its connected vertices information. The reducer computes the sum of edge weights, number of edges, mean of edge weights, and the largest edge weight.

1: **class** MAPPER
2: **method** MAP(string $[vertex]$, string $[leader, sumW, sumN, maxWeight]$)
3: $s \leftarrow count\ 1$
4: $w \leftarrow sumW$
5: $n \leftarrow sumN$
6: $mw \leftarrow maxWeight$
7: EMIT($leader$, set $[s, w, n, mw]$)

1: **class** REDUCER
2: **method** REDUCE($leader$, sets $[[s_1, w_1, n_1, mw_1], [s_2, w_2, n_2, mw_2], \ldots]$)
3: $sumOfWeight \leftarrow 0$
4: $numOfEdges \leftarrow 0$
5: $maxWeight \leftarrow 0$
6: $size \leftarrow 0$
7: **for all** set $[s, w, n, mw] \in$ sets $[[s_1, w_1, n_1, mw_1], \ldots]$ **do**
8: $size \leftarrow size + s$
9: $sumOfWeight \leftarrow sumOfWeight + w$
10: $numOfEdges \leftarrow numOfEdges + n$
11: **if** $mw > maxWeight$ **then**
12: $maxWeight \leftarrow mw$
13: $maxWeight \leftarrow maxWeight * 2$
14: $meanOfWeight \leftarrow sumOfWeight/numOfEdges$
15: $stat \leftarrow [size, sumOfWeight, numOfEdges, meanOfWeight, maxWeight]$
16: EMIT($leader$, $stat$)

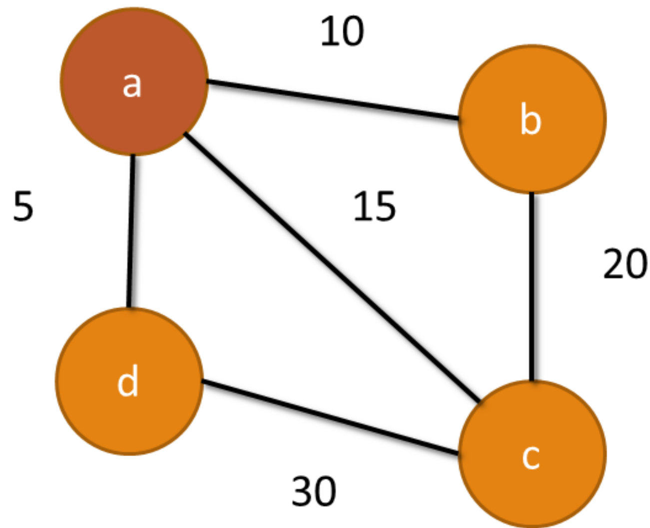


Figure 3.3: Example graph for illustrating statistic algorithm

edge	leader	weight
(a, b)	a	10
(a, c)	a	15
(a, d)	a	5
(b, c)	a	20
(c, d)	a	30

Figure 3.4: Initial records

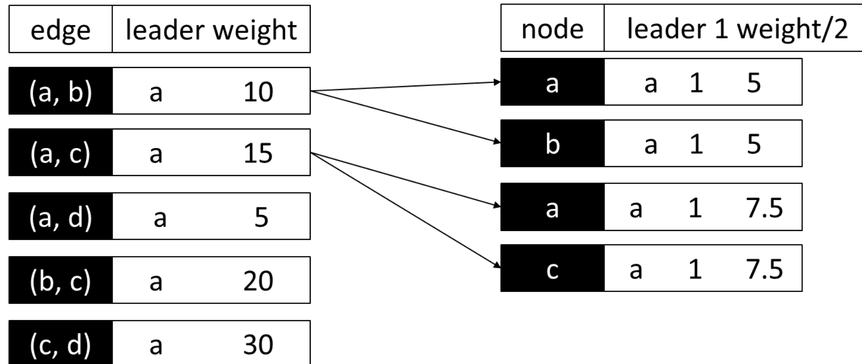


Figure 3.5: Statistic algorithm step1 mapper

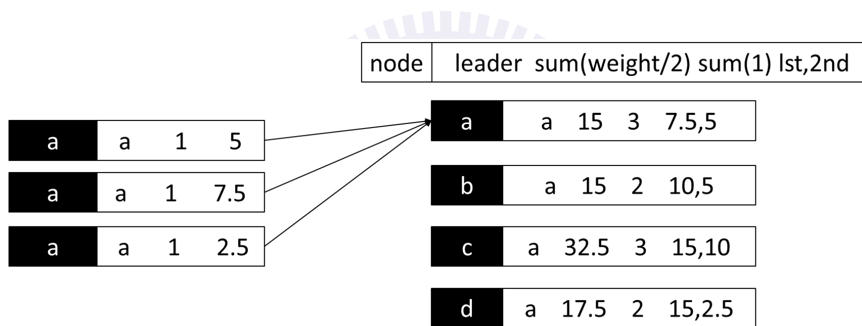


Figure 3.6: Statistic algorithm step1 reducer

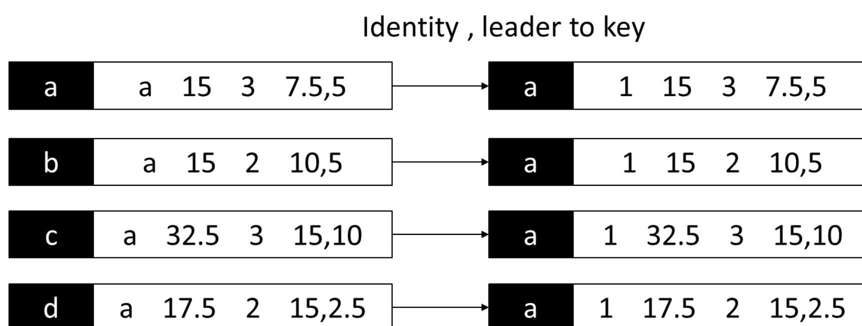


Figure 3.7: Statistic algorithm step2 mapper

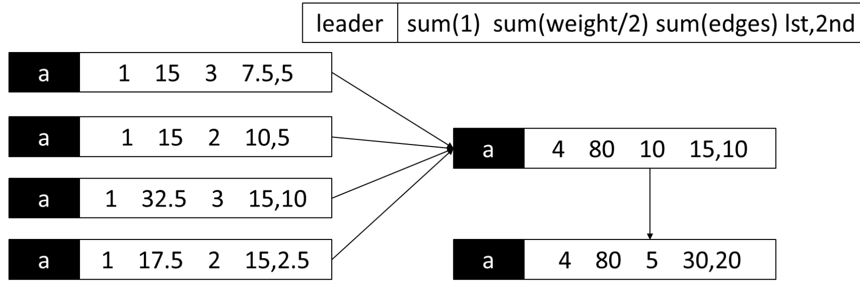


Figure 3.8: Statistic algorithm step2 reducer

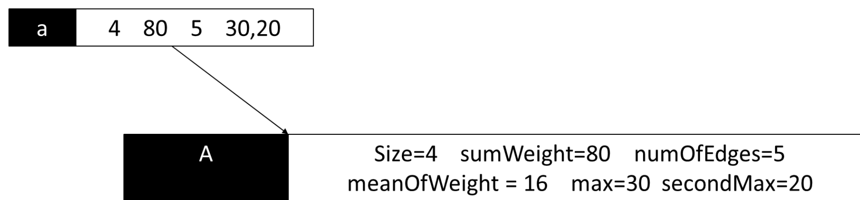


Figure 3.9: Statistic algorithm step3

distribution of the call duration. Next, we create a new graph by removing each edge with weight below a threshold from the original graph. Whenever, a new graph is generated, the K-truss clustering module is used again to cluster the new graph. We check if the giant component still exists. We could repeat the above procedure until either there are no giant components or there is no way to decompose the giant components.

3.4 Influential ranking

In this section, we propose two MapReduce algorithms to find alpha users and rank alpha users. We first append the edge weights to the output of the K-truss clustering module. As a result, the format of records is represented by $((v_1, v_2), l, w)$, where l is the leader of the edge (v_1, v_2) and w is the weight of the edge (v_1, v_2). We use the well-known eigenvector centrality to find alpha users. In addition, we propose ranking alpha users based on the shortest path coverage. In the end, the influential ranking module creates a list of

the most influential users in a telecom social network.

3.4.1 Eigenvector centrality computing

The influential ranking module calculates the eigenvector centrality of each member in each community. It takes the output of k-truss clustering module as input. To reduce computational complexity, we cluster call data records into several communities first and then calculate eigenvector centrality of each community instead of calculating centrality directly. If we want to calculate the eigenvector centrality directly, the only choice is to use the iteration method. However, from our own experience in Hadoop programming, an iteration method is always quite slow, especially running many iterations usually needs longer period of time. After clustering data into communities, we only need one more MapReduce round which a power method of eigenvector centrality computing is embedded in the Reducer. Thus, this algorithm not only saves the running time but also allows us to easily observe the most influential user in each community. The details of algorithm are shown in Algorithm 3.4.

3.4.2 Influence ordering between communities

In practical social marketing, the resource of marketing is usually limited. However, it's easily to extract lots of small communities users from the original graph which is created by call data records. And each community has its own influential user. So several issues arise : First, how can we compare the importance of different influential users in different communities ? In addition, if we are the marketing operators in a telecom company, who should we invest the marketing resources first ? To address these issues, we propose a new algorithm to rank alpha users in different trusses. The key idea of the algorithm is that the intimacy between two users usually relates to their accumulated phone call duration. Namely, the longer phone call duration

Algorithm 3.4 Finding influential users

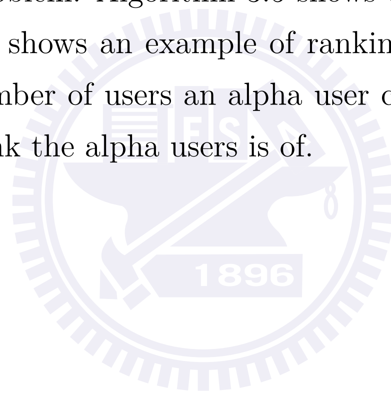
The mapper emits an intermediate key-value pair for each edge and its leader. The reducer collects all the edges it has, and then creates adjacency list to compute the largest eigenvalue and corresponding eigenvector by power method.

```
1: class Mapper
2:     method Map(string [v_1, v_2], string leader)
3:         edge ← string [v_1, v_2]
4:         Emit(string leader, edge)        ▷ Pass leader as key, edge as value

1: class Reducer
2:     method Reduce(string leader, edges [edge_1, edge_2, ...])
3:         L ← string leader
4:         L.AdjacencyList ← edges [edge_1, edge_2, ...]
5:         A ← L.AdjacencyList
6:         start with vector y = z, the initial guess
7:         for k = 1, 2, ... do        ▷ Power method to calculate eigevalue
8:             v = y/‖y‖_2
9:             y = Av
10:            κ = v * y
11:            if isConverged(‖y − κv‖_2) then
12:                break
13:        accept λ = κand x = v
14:        Emit(string leader, pair [λ, x])
```

they have, the closer their relationship is. Starting from this idea, we define the abstract distance between two users as follows :

$$\textsc{AbstracDistance} \propto 1/\textsc{AccumulatedCallDuration} \tag{3.1}$$

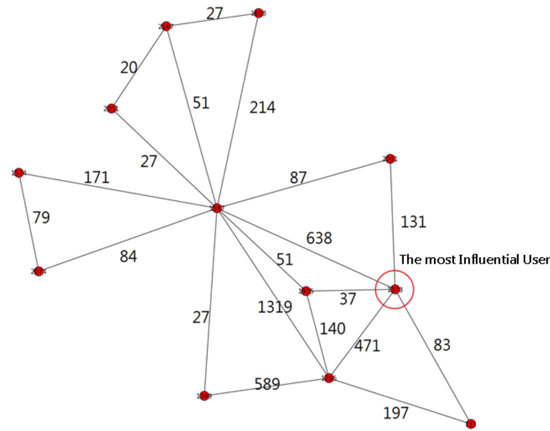
Once we have the abstract distance between users, we can use a well-known shortest path algorithm such as the *Dijkstra* algorithm to determine the shortest distance between two users. Back to the original problem, the limited marketing resource now corresponds to limited distance. So the ordering problem of influential users in different communities can be transformed into a shortest path problem. Algorithm 3.5 shows the pseudo codes of this algorithm. Figure 3.10 shows an example of ranking procedure. In particular, the larger the number of users an alpha user can reach within limited distance, the higher rank the alpha users is of.



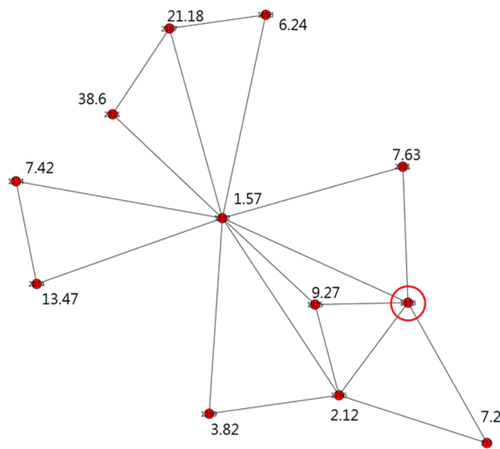
Algorithm 3.5 Shortest path

The mapper emits an intermediate key-value pair for each edge and its leader. The reducer collects all the edges it has, and then creates adjacency list to compute the shortest distance to truss alpha user based on Dijkstra algorithm.

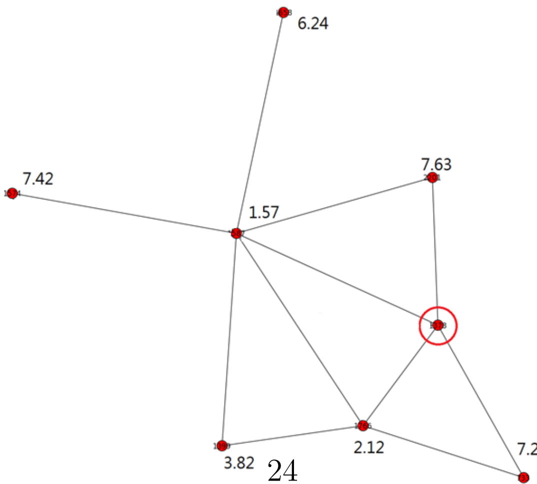
```
1: class Mapper
2:     method Map(string [v_1, v_2, w], string leader)
3:         edge ← string [v_1, v_2, w]
4:         Emit(string leader, edge)        ▷ Pass leader as key, edge as value

1: class Reducer
2:     method Reduce(string leader, edges [edge_1, edge_2, ...])
3:         L ← string leader
4:         L.AdjacencyList ← edges [edge_1, edge_2, ...]
5:         A ← L.AdjacencyList
6:         LD ← LimitedDistance
7:         S ← InlfuentialUser
8:         N = runDijkstraAlgorithm(A, LD, S)        ▷ Start from
   S vertex, apply Dijkstra to find vertices set N with limited distance LD
   requirement
9:         Sort(N)
10:        Emit(string leader, vertices N)
```



(a) A 3-truss community with weighted edge. An alpha user (most influential user) is determined with red circle.



(b) A shortest path algorithm is applied, new weight of each edge is the reciprocal of original weight multiplied by 1000



(c) Given limit distance 8, edges whose weight exceed 8 have been deleted from the graph. As a result, this alpha user can reach 7 users in this condition.

Figure 3.10: A procedure for calculating shortest path ranking algorithm.

Chapter 4

Experimental Results

In this chapter, we include experimental results to show the scalability of the proposed MapReduce framework.

4.1 Datasets and system settings

4.1.1 Datasets

We use the Erdős-Rényi model (ER model) citeNewman2001 to generate synthetic data. A random graph is constructed by connecting vertices randomly. The ER model has two parameters n and p, and therefore it is also called $G(n, p)$ model. The number n represents the total number of vertices, and each edge is included in the graph with probability p independent from every other edge. Right after an edge is created, to simulated CDRs in real world, an actual call duration distribution is applied to determine the weight of each edge. In the end, a random graph is prepared for testing. In addition to synthetic data, real world CDRs from a telecom are used, However, to protect privacy, experiments based on the real world CDRs are executed in a private cluster of the telecom operator.

4.1.2 System settings

To implement the proposed algorithms, we wrote a MapReduce Java program which can be executed on Hadoop platform. Our MapReduce Java program had been executed in a private cluster of 10 computers in our lab. The detail specification of each computer is shown in Table 4.1.

Table 4.1: The computers' specification of the cloud

ID	CPU	RAM	HDD
1	Intel(R) Core(TM)2 Quad CPU Q9500 @ 2.83GHz	4.0GB	290GB
2	Intel(R) Core(TM)2 Quad CPU Q8200 @ 2.33GHz	2.0GB	460GB
3	Intel(R) Core(TM)2 i5 CPU 650 @ 3.20GHz	2.0GB	220GB
4	Intel(R) Core(TM)2 Quad CPU Q8300 @ 2.50GHz	2.0GB	460GB
5	Intel(R) Core(TM) i5-2300 CPU @ 2.80GHz	4.0GB	460GB
6	Intel(R) Core(TM)2 Quad CPU Q8300 @ 2.50GHz	2.0GB	460GB
7	Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz	4.0GB	460GB
8	Intel(R) Core(TM) i5 CPU 760 @ 2.80GHz	2.0GB	460GB
9	Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz	4.0GB	460GB
10	Intel(R) Core(TM)2 Quad CPU Q8400 @ 2.66GHz	2.0GB	290GB

4.2 K-truss clustering

In Figure 4.1, we show a 3-trusses clustering result of a real data, each member of 3-trusses subgraph is randomly assigned a color, and the members not in any community is marked in black. Figure 4.2 is obtained by deleting non-trusses members in Figure 4.1. These figures show that the k-trusses components represent central part of each component.

4.2.1 Performance of K-truss clustering

We create social graphs based on the $G(n,p)$ model, where $p = 1.5 \times 10^{-5}$. For a specific collection of telecom CDRs, the corresponding value of p is about 1.5×10^{-6}. In Table 4.2 and Figure 4.3 we show the number of vertices,

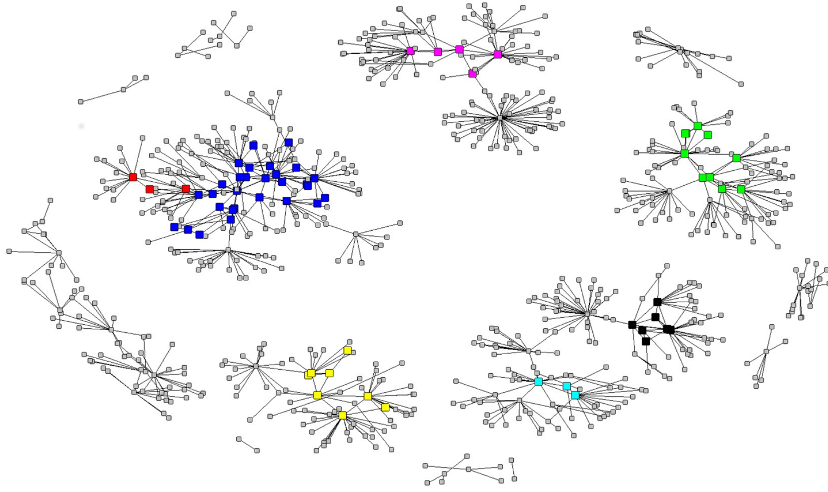


Figure 4.1: The result of 3-trusses clustering of a real data, each member of 3-trusses subgraph is randomly assigned color. And the members not in any community is black.

the total number of edges, the size of each input file, and the execution time for processing each input file in our own cluster. When the number of vertices is set to 1 million, there are 7,242,803 edges, the size of input file is 0.125GB, and it takes around 9.95 minutes for our MapReduce program to finish. When the number of vertices is set to 1million, there are 754,531,223 edges, the size of the input file is 14.578GB, and it takes around 1743 minutes for out MapReduce program to finish.

In addition to the synthetic data, our MapReduce program had been executed in a private cluster of 20 computers that is owned by a telecom operator. In each computer in this cluster, there is an Intel Xeon E5520 2.27GHz CPU and 16GB RAM. In addition, the input file is composed of real call detail records (CDRs). In Figure 4.4, we show the size of each input file, the total number of CDRs in each input file, and the execution time for processing each input file. It only takes 147.5 minutes for our MapReduce program to analyze the 13.5GB input file which has 558,537,632 CDRs. The performance difference between these two experiments is quite large. There are three reasons. First, the size of the input file becomes smaller after

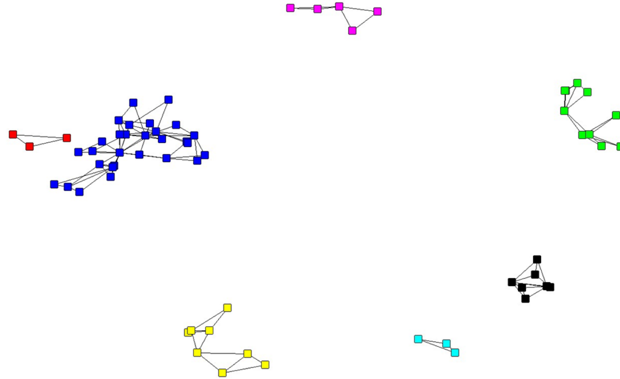


Figure 4.2: The result of 3-trusses clustering of a real data, non-trusses members have been deleted from the graph.

the preprocessing step. Second, there are more and faster computers in the telecom operator's private cluster. Third, the configuration of the telecom operator's cluster had been optimized for large data processing. These results show that our MapReduce program has the ability to process large data in real world telecommunication networks.

Table 4.2: Synthesis data generated from Erdős-Rényi model with $p = 1.5e^{-5}$

vertices (millions)	file size (GB)	total edges
1	0.125	7,242,803
2.5	0.868	46,537,077
5	3.587	187,814,176
7.5	8.156	423,754,868
10	14.578	754,531,223

4.3 Decompose giant component

In this section, three synthetic data with 100 thousand vertices is prepared as input to test the decomposing function. As [24], each synthesis

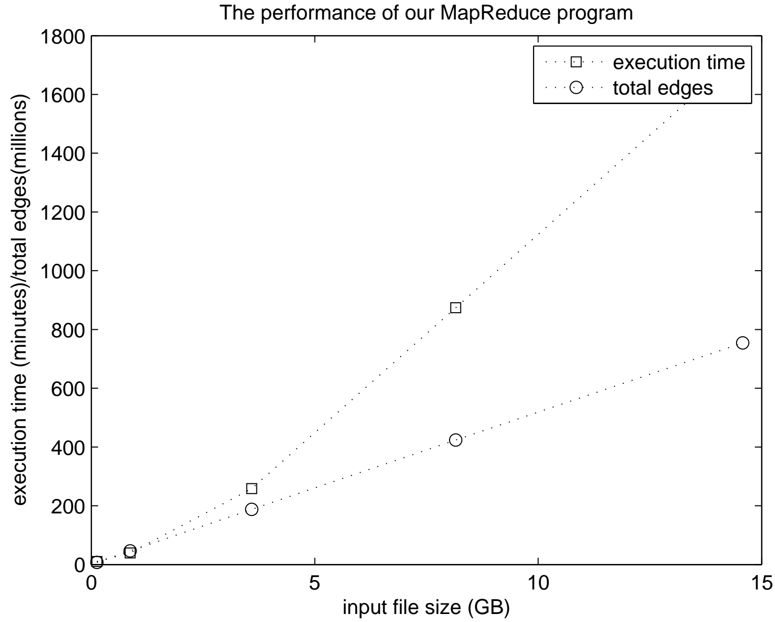


Figure 4.3: The experimental results of synthesis data in our own clusters

data is generated with different weight distribution which is real world call duration distribution, exponential distribution, or log-normal distribution. In Table 4.3 we show the optimal parameters of the log-normal and the exponential distribution to approximate the actual density function. Figure 4.5 shows the three probability density functions for call duration. In addition, we set the acceptable size of each generated component after decomposing to 300.

Table 4.3: The parameters of density function

Density function	**mean** μ	**std** σ
log-normal	100	1
exponential	150	

In Figure 4.6, 4.7, and 4.8, the trends are quite similar. In particular, as the normalized threshold increases from 0.1 to 0.6, the total number of acceptable components increases too, but the maximum component size and survived edges ratio decreases. The decreasing of survived edges ratio shows

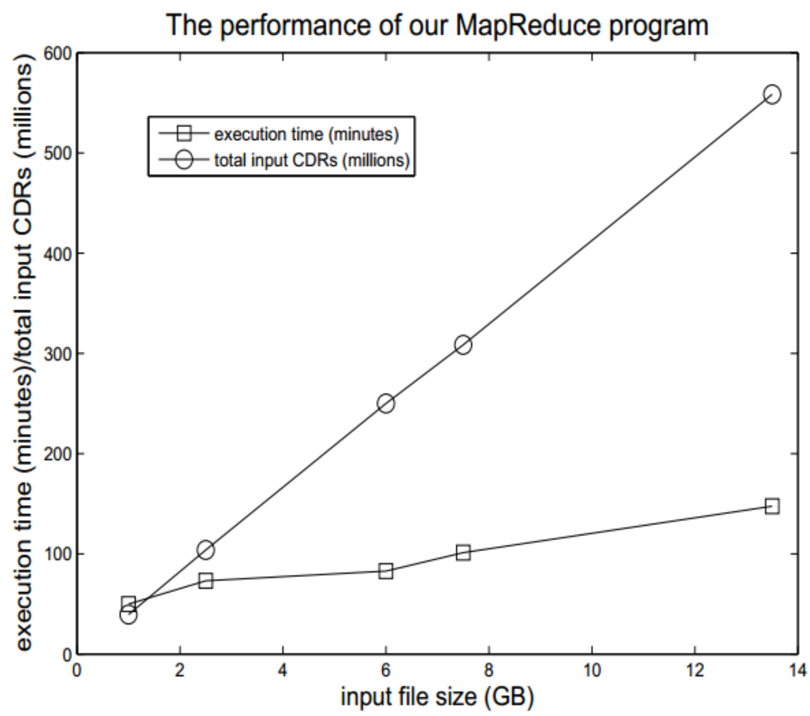


Figure 4.4: The experimental results of synthesis data in a Chunghwa Telecom's private server

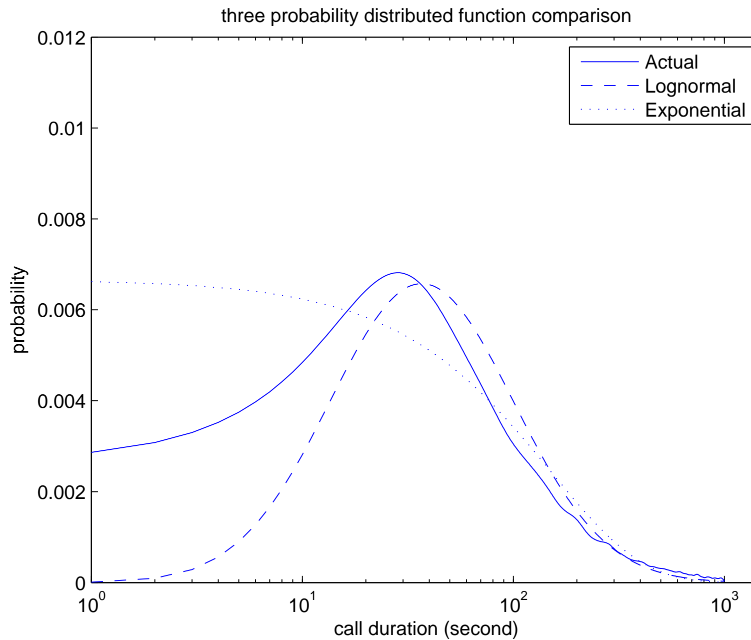


Figure 4.5: The probability density function of actual call duration distribution, exponential distribution, and log-normal distribution

that many edges with smaller weight had been deleted. As a result, the giant component seems no longer inseparable. There are more and more smaller components had been separated from the giant component. It explains the trends of the decreasing of maximum components size and the increasing of total number of acceptable components. When the normalized threshold increases from 0.6 to 0.8, all the three lines decreases. Because there are too much edges had been deleted, many acceptable components were deleted as well.

4.4 Result of influential rankings

In Figure 4.9, we show two results of 3-truss clustering and 4-truss clustering to the same real world community and the corresponding truss alpha user. For a rectangle, the number next to the rectangle is the eigenvector

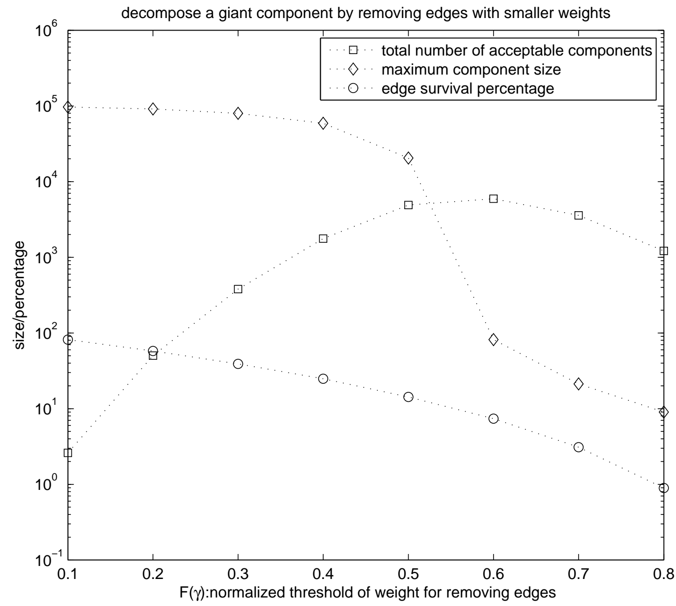


Figure 4.6: The decomposing result of actual distribution with five times the average

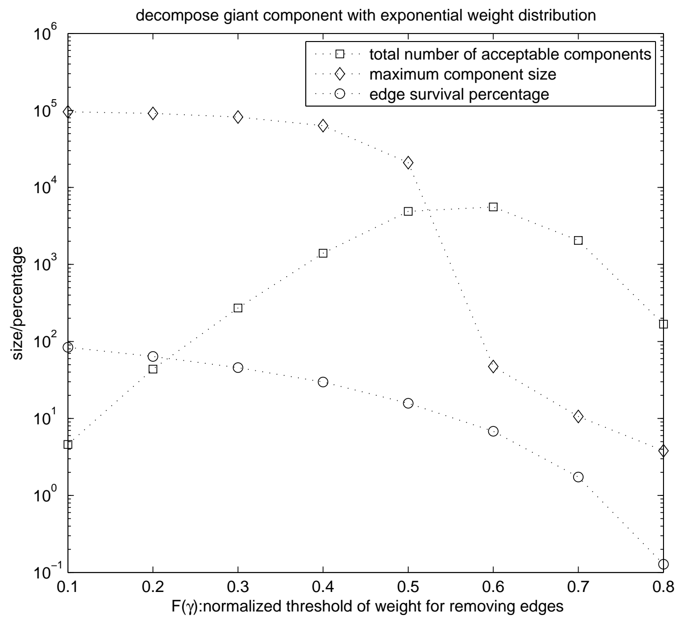


Figure 4.7: The decomposing result of exponential distribution with five times the average

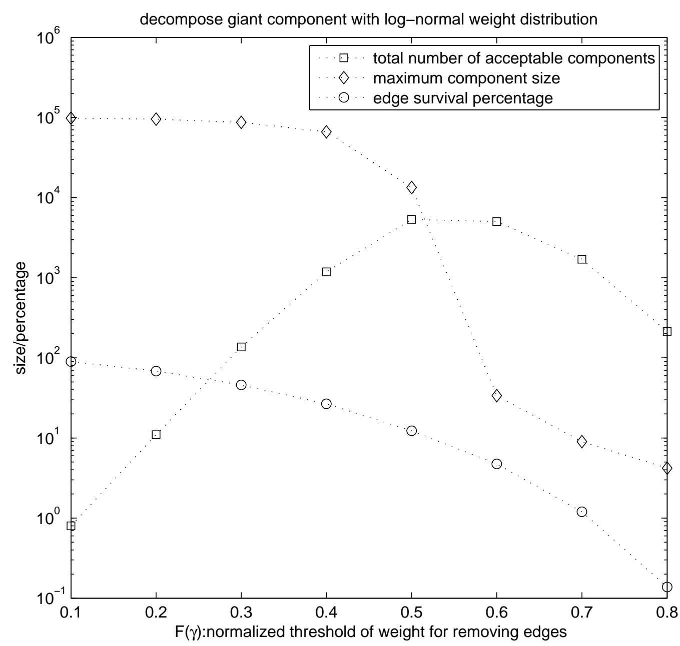


Figure 4.8: The decomposing result of log-normal distribution with five times the average

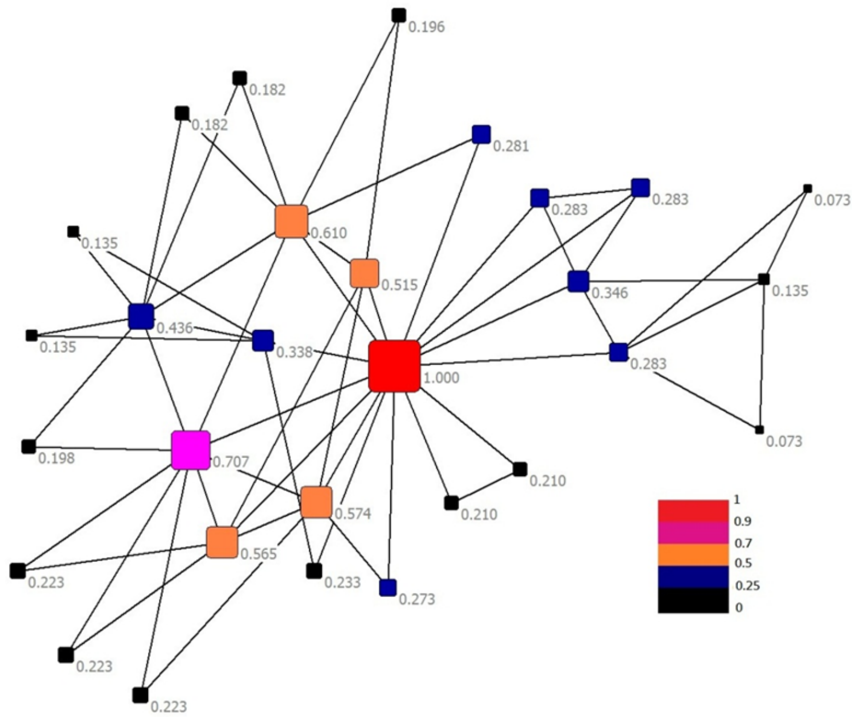
centrality of the corresponding user. The figure is created by the NetDraw network visualization tool [23]. We can find a 4-truss rather than a 3-truss represents the more central part of this community.

In Figure 4.10, we show the relationship between the result of shortest path ranking algorithm and the size of the corresponding community with five different limit distance in the synthetic dataset with real edge weight distribution. The variable in the X axis represents the rank number of each truss alpha user. The smaller the rank number is, the larger the total number of vertices the associated alpha user can reach. The variable in the Y axis is the size of the community. In general, top truss alpha users can reach more vertices in the same limit distance condition, and it usually belongs to the community with larger size. The trend in Figure 4.11 is similar to that in Figure4.10.

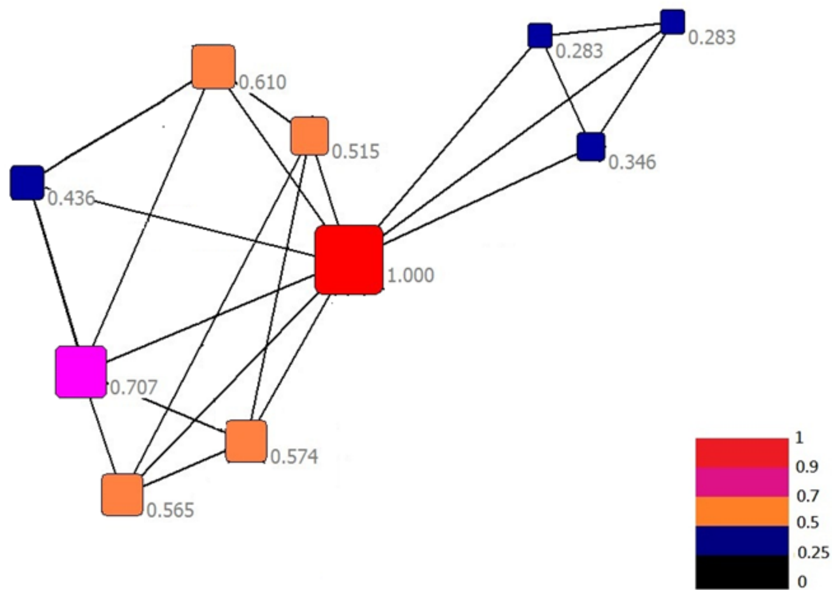
4.4.1 Another social network

We have applied the proposed MapReduce algorithm to two non-telecom social networks. The first one [25] is the collaboration graph of authors of scientific papers from DBLP computer science bibliography. In the DBLP social graph, an edge between two authors represents a common publication. In addition, edges are annotated with the date of the publication. There may be multiple edges between two vertices, when the two authors have written multiple publications together. The second one Libimseti.cz [26], is a Czech dating site. This is the network of ratings given by users of Libimseti.cz to other users. The network is unipartite, directed, and edges represent ratings on a scale from 1 to 10.

In Figure 4.12, we show the 3-trusses clustering result of DBLP dataset. The variable in the X axis represents the size of community, the giant community with size 839,636 is excluded. The variable in the Y axis represents the total number of communities. In Figure 4.13, we show the edge weight probability density functions of DBLP and Libimseti. These two distribu-



(a) An Influential rankings result of 3-trusses community.



(b) An Influential rankings result of 4-trusses community.

Figure 4.9: 3,4-truss community and its corresponding truss alpha user

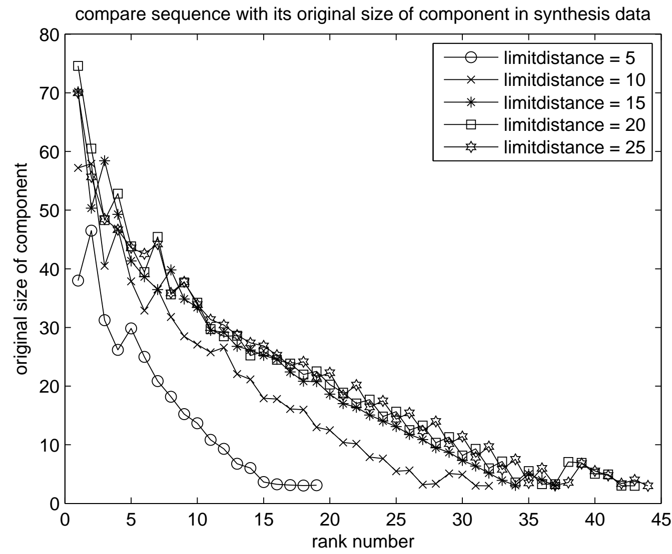


Figure 4.10: Comparison of the sequence of each truss alpha user in different community and the corresponding size of truss-component in synthesis data with five times the average

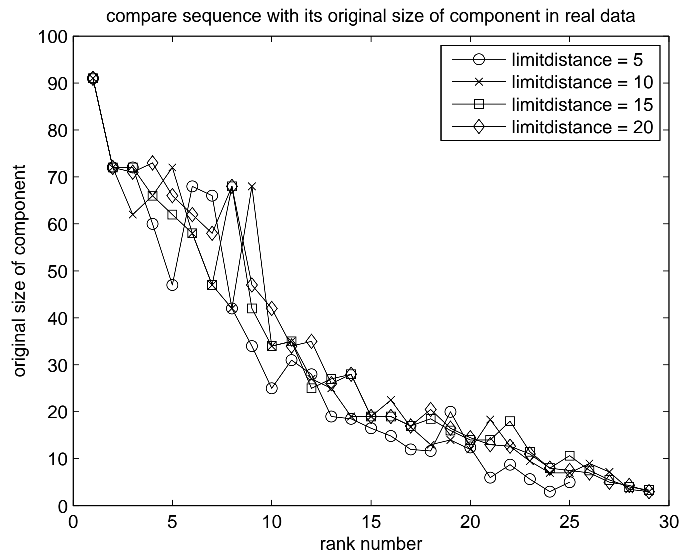


Figure 4.11: Comparison of the sequence of each truss alpha user in different community and the corresponding size of truss-component in real world data

tions are quite different from the call duration distribution. For the DBLP or Libimseti, the weight of an edge is between 0 and 10. Table 4.4 shows the features of the giant community after the 3-trusses clustering in each dataset. In Figure 4.14 and Figure 4.15 we show the decomposing results of these two data. The giant component/community is not decomposed to many smaller/ acceptable components. The results are very different from those of telecom social networks. Since the two giant components constitute of edges with large weight, it is very difficult to decompose the giant components by filtering out edges with smaller weight. In Figure 4.16, we show the relations between the rank of an alpha user and the size of the corresponding truss, when the shortest path ranking algorithm is applied to the DBLP dataset. The trend in the DBLP dataset is very similar to that in the telecom dataset.

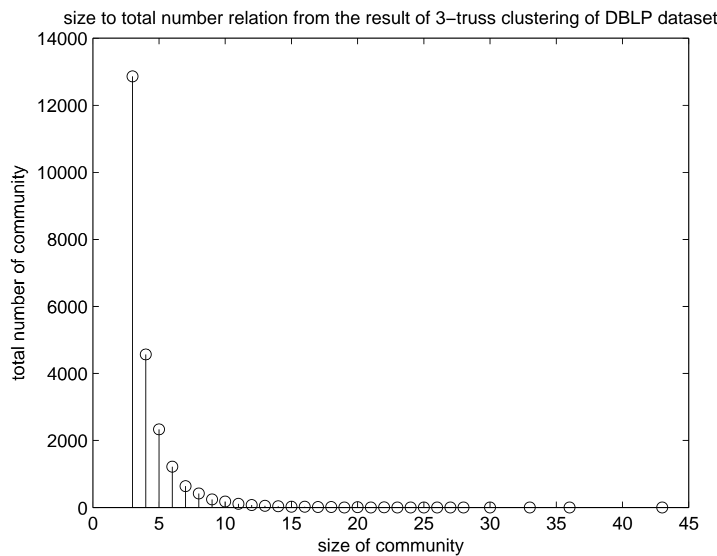


Figure 4.12: the result of 3-trusses clustering in DBLP dataset

Table 4.4: The giant community in DBLP and libimseti dataset after 3-truss clustering

Dataset	**size**	**total number of edges**
DBLP	839,636	3,866,921
libimseti	181,563	13,138,033

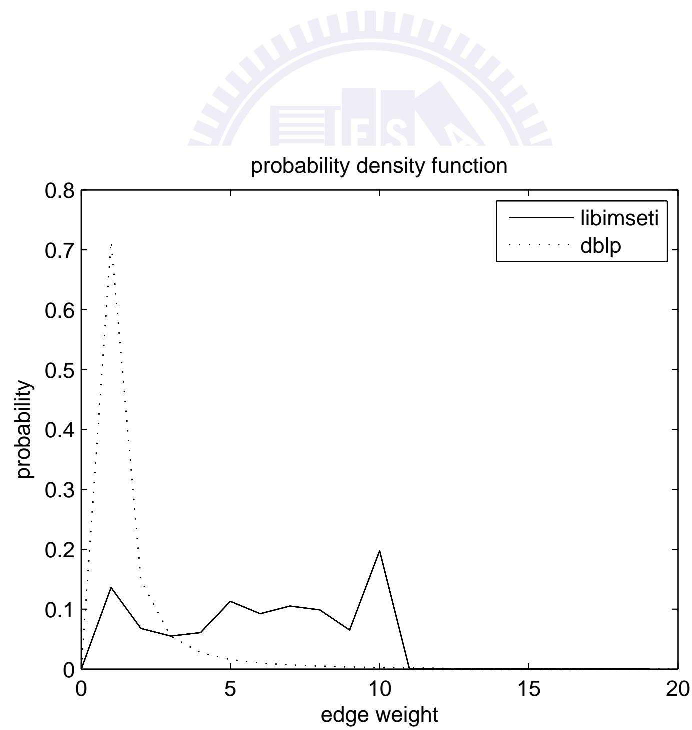


Figure 4.13: The edge weight probability density function comparison of DBLP and Libimseti dataset

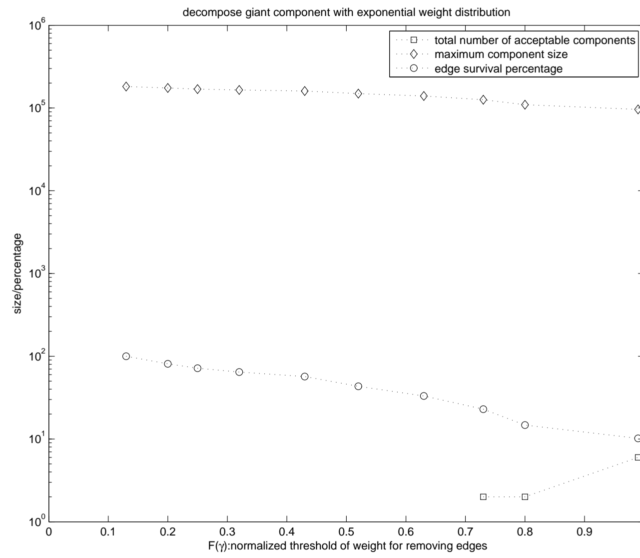


Figure 4.14: The decomposing result of Libimseti dataset

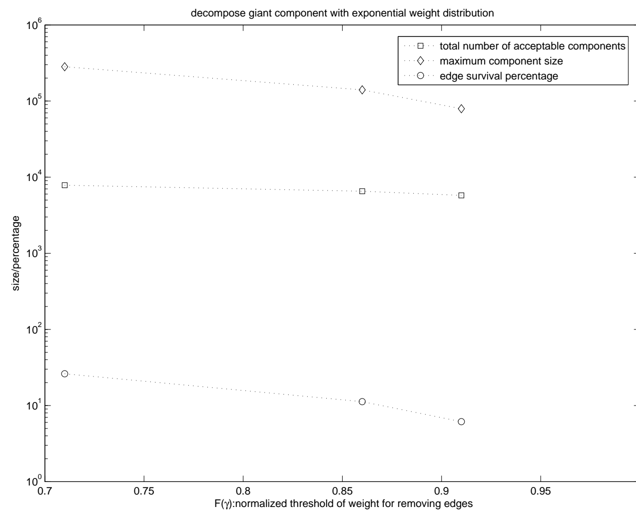


Figure 4.15: The decomposing result of DBLP dataset

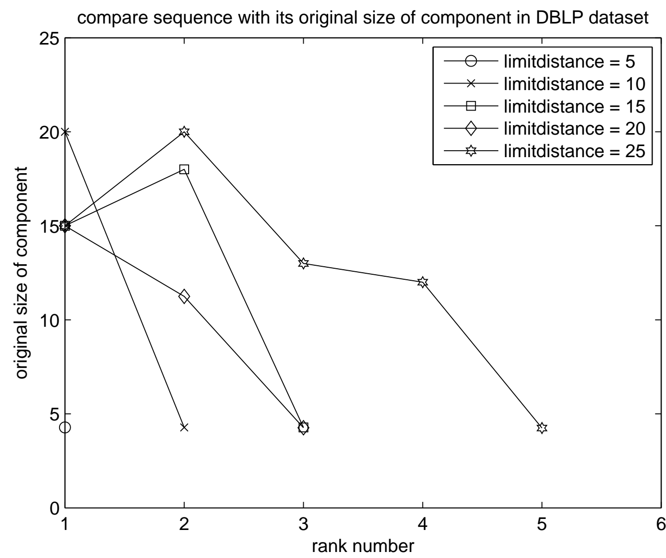


Figure 4.16: Comparison of the sequence of each truss alpha user in different community and the corresponding size of truss-component in DBLP dataset

Chapter 5

Conclusion

In this thesis, we have proposed novel MapReduce algorithms to search for and rank alpha users in massive smart phone social networks. We have applied the principle of divide-and-conquer to find out all trussed in a social graph and then use the eigenvector centrality to identify alpha users in the trusses in a distributed manner. In addition, we have proposed ranking alpha users in different trusses based on the corresponding shortest path coverage. Furthermore, we have proposed novel algorithms to detect and decompose giant components in a social graph. We have implemented and verified the proposed MapReduce algorithms in a Hadoop platform. In addition to synthetic social graphs, we have used the proposed approach to efficiently analyze large-scale smart phone social networks that are created based on call detail records collected by a telecom operator. Future work includes analyzing the computational complexity of the proposed MapReduce framework. Another direction of future research is to exploit additional communication records such as HTTP connection records and geographical data.

Bibliography

[1] M. Tavakolifard and K. C. Almeroth, ”Social computing: an intersection of recommender systems, trust/reputation systems, and social networks,” *IEEE Network*, vol. 26, no. 4, pp. 53-58, July 2012.

[2] http://en.wikipedia.org/wiki/Social_marketing_intelligence.

[3] M. Lidstrom, M. Shahan, and M. Svensson, ”A Method for Providing Content and Service Recommendations Using Social Information from Telecommunications Networks,” in *Proc. 2011 IEEE MDM*, pp. 321-328.

[4] Y. Dong, Q. Ke, Y. Cai, B. Wu, B. Wang, ”TeleDatA: data mining, social network analysis and statistics analysis system based on cloud computing in telecommunication industry,” in *Proc. 2011 ACM CloudDB*.

[5] G. Chartrand, L. Lesniak, P. Zhang, *Graphs and Digraphs*, 5th edition. Chapman and Hall/CRC, 2010.

[6] J. Magnusson and T. Kvernvik, ”Subscriber classification within telecom networks utilizing big data technologies and machine learning,” in *BigMine’12, Beijing, China*, pp.77-84, Aug. 2012.

[7] M. E. J. Newman, ”Analysis of weighted networks,” *Physical Review E*, vol. 70, no. 5, Nov. 2004.

[8] J. Cohen, ”Graph twiddling in a MapReduce world,” *IEEE Computing in Science and Engineering*, pp. 29-41, July 2009.

[9] J. Dean and S. Ghemawat, ”MapReduce: Simplified data processing on large clusters,” in *Proc. 2004 USENIX OSDI*, pp. 137-149.

[10] http://hadoop.apache.org.

[11] I. Palit and C. K. Reddy, ”Scalable and Parallel Boosting with MapReduce,” *IEEE Trans. Data and Knowledge Engineering*, vol. 24, no. 10, pp. 1904-1916, Oct. 2012.

[12] A. Bahga and V. K. Madisetti, ”Analyzing Massive Machine Maintenance Data in a Computing Cloud,” *IEEE Trans. Parallel and Distributed Systems*, vol. 23, no. 10, pp. 1831-1843, Oct. 2012.

[13] M. Cardosa, A. Singh, H. Pucha, and A. Chandra, ”Exploiting Spatio-Temporal Tradeoffs for Energy-Aware MapReduce in the cloud,” *IEEE Trans. Computers*, vol. 61, no. 12, pp. 1737-1751, Dec. 2012.

[14] M. E. J. Newman, S. H. Strogatz, D. J. Watts, ”Random graphs with arbitrary degree distribution and their applications,” *Physical Review E*, vol. 64, 2001.

[15] M. E. J. Newman, ”Fast algorithms for detecting community structure in networks,” *Physical Review E*, vol. 69, 2004.

[16] R.-H. Gau, T.-C. Hsieh, S.-W. Tsai, and C.-P. Cheng, ”An Implementation Framework of MapReduce Email Social Network Analysis,” The 7th ACM Workshop on Wireless Multimedia Networking and Computing, October 2011, pp. 67-69.

[17] R.-H. Gau, S.-W. Tsai, and T.-T Tseng, ”Searching for Truss Alpha Users in Mobile Telecommunications Social Networks,” *IEEE GLOBECOM 2013, Atlanta, GA USA*, Dec. 2013.

[18] J. Wang and J. Cheng, ”Truss Decomposition in Massive Networks,” in *Proc. VLDB Endowment*, vol. 5, no. 9, pp. 812-823, Aug. 2012.

[19] M. Bianchini, M. Gori, and F. Scarselli, ”Inside PageRank,” *ACM Trans. Internet Technology*, vol. 5, no. 1, pp. 92-128, Feb. 2005.

[20] J. Weng, E.-P. Lim, J. Jiang, and Q. He, ”TwitterRank: Finding Topic-Sensitive Influential Twitterers,” in *Proc. 2010 ACM WSDM.*

[21] A. Kumar, D. Manjunath, and J. Kuri, *Wireless Networking.* 2005, Morgan Kaufmann.

[22] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd edition. The Johns Hopkins University Press, 1996.

[23] Borgatti, S.P., 2002. NetDraw Software for Network Visualization. Analytic Technologies: Lexington, KY. https://sites.google.com/site/ucinetsoftware/home.

[24] J. Guo, F. Liu, and Z. Zhu, ”Estimate the Call Duration Distribution Parameters in GSM System Based on K-L Divergence Method,” in *International Conference on Wireless Communications, Networking and Mobile Computing, Shanghai, China*, pp. 2988-2991, Sep 2007.

[25] M. Ley. The DBLP computer science bibliography: Evolution, researchissues, perspectives. In *Proc. Int. Symp. on String Processing and Information Retrieval,* pages 1-10, 2002.

[26] Libimseti.cz, http://konect.uni-koblenz.de/networks/libimseti