Design of a Crawler for Online Social Networks Analysis

Chi-In Wong[1], Kin-Yeung Wong[2], Kuong-Wai Ng[3], Wei Fan[4], Kai-Hau Yeung[5]
Computing Program, Macao Polytechnic Institute,
R. de Luis Gonzaga Gomes, Macao[1,2,3]
Engineering Science, the Open University of Hong Kong
30 Good Shepherd Street, Homantin, Kowloon, Hong Kong[2]
Department of Electronic Engineering, City University of Hong Kong,
83 Tat Chee Avenue, Kowloon, Hong Kong[4,5]
kazec.y.wong@gmail.com[1], wood@ieee.org[2], tenjoukouka@hotmail.com[3], fanwei.fw@gmail.com[4],
eeayeung@cityu.edu.hk[5]

Abstract: - The growths of online Social Networks in scale and amount of information are immense in recent years. The analyses of the structure of online social networks have thus drawn much research interests. Before the analyses, the information and the characteristics of the structure have to be obtained. However, the complexity of today's web technologies imposes challenges for collecting the data. In this paper, we discuss the design and implementation of a crawler for online social networks, and propose countermeasures to the technical challenges when crawling the networks in practice. Then, the crawled data is visualized and analyzed in graphs.

Key-Words: - Social Networks, Network Analysis, Crawler, Software Design, Automated Agents, Network Visualization.

1 Introduction

The increasing popularity of online social networks (OSNs) has gathered hundreds of millions of users. OSNs have become a platform for people to easily communicate and share information, particularly with the sophisticate smartphones [1]. Since the structures of OSNs will be able to reflect the real-life society in certain extend, the structure and the information shared in OSNs are of interests for different communities [2,3,4,5]. For instance, sociologists regard OSNs as a venue for collecting relationship data and study online human behaviors. Marketers, in contrast, seek to exploit information about how messages spread so as to design viral marketing strategies. For network engineers, understanding OSNs improves the design of interconnected systems so as to provide better user experience.

In order to analyze the structure of an OSN, information regarding the network structure is needed. Since OSN operators will not disclose their network structures (completely or partially) to the public, writing a crawler is a common way to obtain the data. A crawler in a social network is a program that starts with a list of user's page to visit. As the crawler visits these pages, it identifies all the friend pages embedded in the current page, and then follows those just identified pages to discover more new pages. This process continues until some criteria are met.

However, there are many challenges when crawling OSNs. For example, the complexity of today's web technologies (e.g., JavaScript and Ajax) makes it challenging when interpreting the content of a page. And, the access restrictions of most OSN services (e.g., login requirements, limited view, API query limits) impose difficulty to crawl the network with sufficient amount of samples. On the other hand, the privacy control policies do not allow a crawler to access the entire online social network. The objective of this paper is to design and implement a crawler for Facebook and provide practical recommendations to tackle the challenges during the crawling process.

One of the objectives of our work is to design and implementation of a crawler for OSNs, which is an automated program that systematically collects data on OSNs. The data collected can be used to rebuild the network topology, for statistical analyses, etc. The second objective is to provide practical recommendations to tackle the challenges during the crawling process. Another objective is to visualize and analyze the structure of the crawled network.

Note that our crawler is only able to collects the public information on Facebook, There are no

ethical issues since Facebook's user agreement states that all public information can be used by

The organization of this paper is as follow: background of our work is discussed in section 2. The design of crawling OSNs is presented in section 3. The method of implementing a crawler is proposed in section 4. Technical challenges during the crawling are discussed in section 5. Social Network Analysis is briefly introduced in Section 6. After that, our result in crawling Facebook is illustrated, visualized and discussed in section 7. Finally, it is concluded in section 8.

2 Background

2.1 Related Work

The task of extracting and analyzing data from online social networks has attracted the interest of researchers, as shown in [3,18,19]. The most popular social network, Facebook, naturally gets the most attention from researchers, who measured some large-scale network properties of the Facebook graph through sampling, crawling, and other methods to collect network data. In this section we review the preliminary work on social network crawling and social graph sampling and some relevant literature directly related to our approach.

Although there are a large number of social network publications, few have been dedicated to the data collection process. Chau et al. [20] exploited Breadth-First-Search (BFS) algorithms and illustrated the use of parallel crawlers to crawl eBay profiles efficiently. Mislove et al. [21] analyzed the graphs of a number of popular online social networks, including Flickr, YouTube, LiveJournal, and Orkut. Their analyses confirm the scalability properties of OSNs, such as a power-law degree distribution, a densely connected core, and small average path length.

In this paper, we will first discuss the data collection techniques and the data collected using these methods. Collected data are usually mapped onto graph data structures with the goal of analyzing their structural properties.

2.2 OSN Analysis

OSN analysis begins with the goal to understand the organization of popular OSN. Facebook is one of the most studied OSNs. According to iStrategyLabs.com, its growth rate has been proved to be the highest among all the other competitors in the last few years.

third-party organizations. We sent our crawlers to crawl a portion of the Macao Facebook graph.

Facebook allows a user to create a public profile with pictures and other personal information such as gender, date of birth, hometown, phone number, school, employer, interests, and current GPS location. Each user has a list of friends, but no more than 5000 of them. Two users can establish a friendship link by sending and accepting a friendship request.

One important aspect to be considered for representing the model of a social network is the amount of information about its structure that we have access to. The ideal condition would be to have access to the whole network data, for example acquiring them directly from the company managing the social networking service. For very large OSNs, such as Facebook, Twitter, etc., it is hard to collect a complete sample of the network. The first limitation is related to the computational overhead of a large-scale Web mining task. In the case of Facebook, for our case, to crawl the friend list Web page (dimension ≈ 3MB) for the 250 thousands of users in Macao, it requires to download more than 3MBx250,000 = 750 Gigabytes of HTML data.

3 Design of the crawling in an Online Social Network

3.1 Limitation of Crawler

The structure of an OSN can be modeled as a graph G = (V,E), where V is a set of nodes (users) and E is a set of edges (friendships). In this case, G is undirected because the friendship of two users in OSN is undirected. In Facebook, the whole G is not accessible for everyone due to the privacy control of individual users. That is, users can set their information to be only shown to their authorized people (e.g., their direct friends) so others (including our crawlers) are not able to get it. Therefore, in general, we can only access the public part of G, denoted as G'. Therefore, it has to be understood that only G' can be crawled in a legitimate way.

3.2 The Information to Collect

Before crawling an OSN, it is important to identify the goal of crawling and understand what kind of data is needed. The design of a crawler highly depends on the types of data to be crawled. It also affects the terminate condition during the crawling process. For example, if only the topology of the social network is required, the crawler just needs to grasp the friend list in a page. However, it has to

crawl the pages of a particular user domain, then the crawler may need to extract the information related to the user domain, such as the country, age, or interests of the user in the page. Extracting these attributes requires format analysis and content analysis in the HTML content. The more attributes are required, the more technologies are needed. Therefore, knowing what data to be crawled helps optimizing the crawler program.

3.3 Choosing the Initial Node

Generally, choosing those users with many links to others as the initial nodes will be helpful and able to speed up the whole crawling process. Those highly connected users can be identified by the number of friends they have.

However, if the users in a particular region are needed, you can identify them by using the OSNs' search functions. For example, Facebook provides a feature to show a number of randomly selected users from a given regional network. If you want some users from London, you can search them using the keyword "London user" on Facebook. Submitting this query several times can produce some seeds for our crawl.

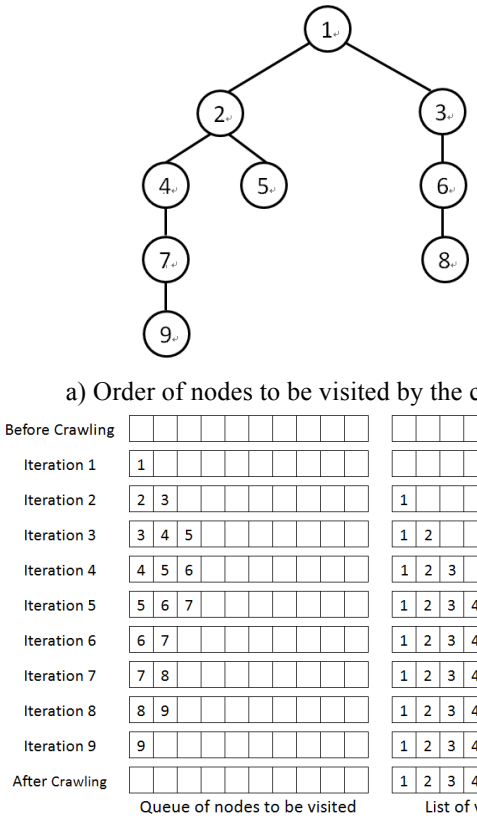
3.4 Search Algorithms

The crawling process of the social graph starts with an initial node and explores iteratively. In each of the iterations, we visit a node and discover its direct neighbors.

There are two major ways for the exploration, depending on which neighbor to visit next. The first way is Breadth-First-Exploration (BFE). The BFE first starts at the initial node and explores all its neighbors. Then for each of those unvisited neighbor nodes in turn, it explores their neighbors. Another way is Depth-First-Exploration (DFE). The DFE starts at the initial node and explores along each branch as far as possible before returning.

In our implemented crawler (as will be discussed in section 4), BFE is used. Before starting the search, 1) initial node; 2) a queue (that stores the nodes to be visited next); 3) a list (that stores all visited nodes) are needed. The process is as follow:

1. Add the initial node to the queue.
2. Dequeue the first node in the queue and explore its neighbors.
   If no neighbors are found, repeat step 2 until the queue is empty.
   If neighbor(s) are found, enqueue its neighbors if they are unvisited (by checking if the node is in the list of visited nodes).
3. Add the node to the list of visited nodes.
4. Repeat Step 2.



a) Order of nodes to be visited by the crawler

b) Headless browser operation

Fig. 1 An example of Breadth-First-Exploration

Fig. 1 shows an example of Breadth-First-Exploration. Fig. 1a) shows the order of search in our algorithm, whereas Fig. 1b) shows the contents in the queue of unvisited nodes and list of visited nodes. The process of exploration is very similar to Breadth-First-Search (BFS) in graph search. However, the only aim of this search is to expand the social graph as far as possible, not to find a particular node. Therefore, the stop condition in BFE is different from that in BFS.

3.5 Focused Crawling

Focused crawling or topical crawling refers to the crawling process that attempts to access only user profiles that are relevant to a pre-defined topic. A focused crawler has a main function for relevance judgment on URL crawled to decide which links to follow for further crawling.

The way to evaluate focused crawling is to measure the harvest ratio, which is the rate at which relevant pages are acquired and irrelevant pages are effectively filtered off from the crawl. The harvest ratio increases as the crawling time increases.

3.6 Stopping Criteria

Considering the size of an OSN is huge, crawling the entire OSN may not be necessary for the crawling objective. The crawling process continues until some criteria are met. Typically, the process stops when the number of sample is sufficient or the result of the crawled samples saturates.

There are several ways of evaluating if a crawl is successful. The first and most common criterion is the sufficient number collected samples. Required sample size depends on the maximum desirable error and the acceptable error risk such as confidence level. We used the following formula to estimate the number of samples required:

uare for the
f freedom, N is population size, P is population proportion, and E is the desired margin of error. Sample size should be used in quantitative research where the research aims are to test hypotheses, look at cause and effect, or make predictions.

On the other hand, saturation of the result of the crawled sample is also a useful stop criterion. For example, supposing that the average distance of a network has to be obtained, when the result saturates even thought more samples are collected, it is an indication on the stop of the crawling process.

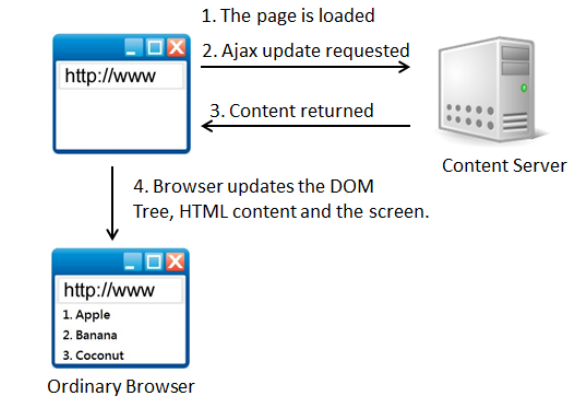
4 The implementation of Crawler

There are issues to consider when implementing an OSN crawler. Nowadays a lot of websites use JavaScript to load data only after the web page is loaded (to hide data from robot programs). This policy imposes difficulty for crawlers to access data on their websites. In this section, we discuss the problem with JavaScript and suggest a solution. We then discuss the operations of a crawler and demonstrate the design of our crawler.

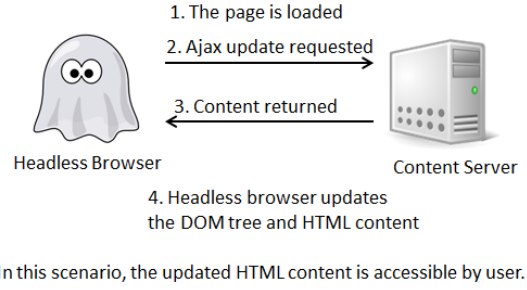
4.1 The Problem with JavaScript

A lot of visible information shown on a Web page does not directly appear as plain texts in the HTML file. After loading the page, dynamic JavaScript calls are made to retrieve the information from the remote server, as shown in Fig. 2. For example, in Facebook, the friend list of a user is not present in the HTML source. It is dynamically loaded using JavaScript after the page load. The DOM Tree of the page is updated after the execution of JavaScript. The key issue is that browsers do not show the updated HTML content.

To grab the textual data on a web page, it is desirable to obtain the HTML in which the embedded JavaScript has been already executed. A simple solution is to write a program to use a headless browser. Users are allowed to execute JavaScript on the page and obtain the updated HTML content in the browser engine. After obtaining the updated HTML content, we can perform HTML scraping to extract the data required.



a) Ordinary browser operation



b) Headless browser operation

Fig. 2 Ordinary browser vs. headless browser

4.2 The Use of Headless Browser

A headless browser is a full-feature web browser with no graphical user interface (GUI). It only accesses web pages but does not show them to user. Headless browsers are used as a web page content provider by other programs. Headless browsers interact with human and other programs through commands.

In our implementation, we used a headless browser called PhantomJS [6]. PhantomJS is a headless WebKit with JavaScript API. WebKit is an open-source web browser engine that powers popular browsers including Chrome and Safari. Headless browsers do not include graphical user

interface, making it feasible for integrating with our own programs/scripts. The JavaScript API allows us to easily write scripts that interact with PhantomJS in the language of the Web, allowing us to modify the DOM.

4.3 The operations of a crawler

Suppose that we want to crawl all users belonging to a particular region, say, Macao. The crawler first picks some initial users. For the page of each user in Facebook, the crawler performs the following:

1. Load the Friends page, extract the list of friends in the parsed HTML code and retrieve all the corresponding links.
2. In each retrieved links, access their profile's attributes to check its location information.
3. For each link retrieved, repeat the process.

We first specify an initial user HTML page (we call it root-page). The crawler then follows all links found in root-page. It will lead to more links, which will be followed again later. This process will result in a tree-like searching graph, where the root of the tree is the root-page. All links contained in that root-page are direct children of the root, and subsequent links then become the children of the previous children.

For each link found, the crawler extracts the profile page. The crawler then identifies the elements by performing string matching. We check if there is a text chunk of the targeted region to determine if the link is associated with a right user from the targeted region. To find the matching locations we used XPath, which is a query language used to locate a node in an XML document. It will match a single node or will be generalized to match a set of nodes. The XPath feature is supported in PhantomJS.

The crawler starts by parsing a specified friend list web page, then stores their friends' hypertext links on that page which point to other web pages. The crawler then parses those pages for new links, recursively.

The crawler simply sends HTTP requests for documents to Facebook, just like what a web browser does when the user clicks on links. What the crawler really does is to automate the process of following links.

The crawl process can be regarded as processing items in a queue. When the crawler visits a web page, it extracts links to other web pages. There is a queue used to store unvisited web pages, and a list used to store visited web pages. The root-page should be initially put in the queue. Based on the algorithm we used to explore the Facebook social graph, the crawler pushes the extracted URLs in a web page at the end of the queue, and continues crawling by popping the first URL in the queue. The operation repeats until the social graph is fully explored or the stop criteria are met. It is based on the first-in-first-out basis since BFE (which is described in Section 2) is used in our crawler. Table 1 shows the Pseudocode of our crawler.

```
initialization

function identifyFriendsLocation() {

  for each friend in friends list
    if this friend is in the list of visited nodes then
      ignore this friend
    end if
    Retrieve location of this friend
    if the location is the target region then
      add this friend to the queue of unvisited nodes
      store this link
    end if
  end for
  add the current node to the list of visited nodes
}

function extractFriends() {
  get list of friends
  identifyFriendsLocation();
}

function crawl() {
  if queue of unvisited nodes is empty then
    stop crawling
  end if
  pop the first node in the queue
  load the user profile page
  if friends pagelet exists then
    get number of friends
    estimate time needed to load the full friend list
    wait time estimated
    extractFriends();
  end if
  crawl();
}

login to facebook
crawl();
exit program
```

Table. 1 Pseudocode of our crawler

5 Discussions on technical challenges and countermeasures

There are several challenges when attempting to collect data from Facebook. In this section, we discuss how we tackle the challenges during the implementation of our crawler and the execution of the crawling process.

5.1 Page Dimension Limitation

During the crawling process, a technical limitation is imposed by Facebook on the dimension of the web page showing the list of friends. To reduce workload offered to the server, Facebook will only show a batch of friends at a time (e.g., 100 friends). When the browser detects that all the current batch of friends has been loaded, asynchronous JavaScript will be executed to fetch another batch of friends from the server and show them on the same page.

To avoid this situation, we can enlarge the page dimension in the WebKit browsing engine. In PhantomJS, we can modify the page dimension by setting the page's viewportSize property. It effectively simulates the size of the windows in a traditional browser. We can set the height as large as we need for the layout process such that friends will be retrieved directly from the web page. Doing this can speed up the overall crawling process.

5.2 Asynchronous JavaScript and the delay time setting

The execution of JavaScript takes time. We are not able to know the exact time when it will finish. Therefore, we cannot ensure when the page is completely loaded. If a page we are trying to download contains lots of JavaScript calls, normally we get the HTML source that is not fully expanded. For instance, when we are trying to load a big page (e.g., when there are large number of friends), we have to wait until the JavaScript is fully executed in order to obtain the entire friend list. However, when the execution of JavaScript terminates is unknown. To solve that, for the crawler to decode a page, it has to wait a time period, so that all the JavaScript has been executed before it attempts to serialize its DOM structure.

We can roughly estimate the waiting period needed, which is based on the number of friends shown in the page. Each time Facebook loads additional friends to the friend list, it loads 20 more friends, and the process take about 1 second. So, we can estimate the waiting time by diving number of friends by 20. For example, we have to set the waiting time to 25 seconds for a person who has 500 friends in order to get the full friend list.

5.3 Obfuscated Web Page

The programming code of the Web pages by Facebook has been obfuscated. For example, the formats of the Friend Pagelet IDs are different for the people shown in a page. To solve this, we first use Firebug [7] to parse the obfuscated elements. Firebug is a Firefox browser plugin that facilitates the debugging, editing, and monitoring of web page's HTML, CSS, DOM, and JavaScript. CSS selectors [8] is a common tool to select the HTML elements/nodes. However, since the Pagelet ID formats are in a mess, it is inefficient to use CSS selectors to select all the needed elements properly. In this case, XPath selector is more desirable because it provides better and more user-friendly selection methods.

However, PhantomJS only have basic support of XPath feature which makes the use of XPath difficult. Therefore, we used CasperJS [9], a PhantomJS plugin which provides better support of XPath feature to do the heavy selector search. We should use Firebug to extract the XPath pattern for XPath filtering so as to filter the expected results.

5.4 Incorrectly Formatted Characters

When we were trying to extract some inner text in a web page, not all characters are in English. In our case, we found a lot of Chinese characters. We have to handle foreign or accented characters during html scraping. Scraping information with non-English characters does not always work. As a result, we need to fix this issue by fetching readable Unicode data (in UTF-8 format) while scraping. The script file for the headless WebKit browsing engine should be encoded in utf-8 format since Facebook is also encoded in utf-8.

5.5 Location Identification

The majority of Facebook users belong to a certain regional network, and most users do not modify their default privacy settings. However, many people do not fill in their location information or fill it with the wrong location in their profile. Just retrieving the place field of a Facebook user's profile to identify its location is insufficient and inaccurate. For example, some people claim that they are living in Mars although they actually live in Macao. It causes difficulty when the crawling goal involves the user location information.

To resolve it, three-party information can be used. The website of SocialBakers provides information to identify user location. We searched SocialBakers for demographic facts. We can crawl a regional network by accessing a large portion of

Facebook's user profiles. Facebook graph is divided into networks that represent different schools, institutions and geographic regions. Therefore, we can use these data to attempt to identify their location by checking SocialBakers.

5.6 Limited Access Rate

This challenge happens during the execution of the crawl program. Facebook employs various rate-limiting techniques to restrict the rate to access the web site too frequently. These techniques typically rely on limiting the number of user profiles a single user account or IP address can access in a given period of time. In the crawler, an account has to be used to login Facebook so as to access other users' profiles. After crawling the Facebook network for a few hours, the account will be temporarily blocked because our crawler has made sufficient high number of access to Facebook. After the account is blocked, phone verification is required to unblock it. This verification process interrupts the crawling process, making the large-scale crawling very difficult.

To solve this problem, what can we do is to create a number of Facebook accounts to gain access using different IP addresses from different computers.

5.7 Simultaneously Crawling

A crawler may require a number of visits to the remote server to collect the information needed for one user. Thus, a single crawler is inefficient for crawling large OSNs. In order to shorten the data collection time, multiple machines can be used. In each machine, multiple crawlers can be run simultaneously. And each crawler can use multiple threads to process a number of pages concurrently. There is a master process coordinating the progress of the crawling among the crawlers.

5.8 Resource Constraints

Crawling consumes system resources: 1) network bandwidth to download pages, 2) memory to maintain the data for running the crawling algorithm, 3) processing power to evaluate, select and filter URLs, and 4) disk space to store the content of the fetched pages. A powerful processor and high amount of network bandwidth are desirable.

6 Social Network Analysis

A crawler is designed with the objective of collecting data for social network analysis (SNA). SNA aims at measuring the network nodes (e.g., the individuals in an organization) and their ties (e.g., friendship or partnership) in terms of graph theory. SNA commonly requires the use mathematical equations to calculate certain metrics, and typically use some tools to help visualize the network.

SNA is important because it reveals how the society functions and explains phenomena and social behaviors in the society. It focuses on the relations between individuals, groups and organization rather than individuals and their attributes.

In this section, we discuss the social network analysis methodology and some basic techniques. We also introduce tools for doing network analysis.

6.1 Metrics

In online social network analysis, after obtaining data from a social network (e.g., by a crawler), the network should be reconstructed based on the entities and relationships so as to perform SNA.

There are three major types of metrics used in SNA: 1) Connection, 2) Distribution and 3) Segmentation.

6.1.1 Connection Metrics

1. Network Closure – A measurement of completeness of the network. For example, network closure can tell how many of your friends are also friends themselves.
2. Multiplexity – A measurement of content-forms contained in a tie. Also referred as relationship strength. Nodes that have multiple relationships simultaneously have higher multiplexity. For example, if two people are both friends and colleagues, they have multiplexity 2.
3. Homophily – A measurement of how similar or dissimilar nodes are tied. Also referred as assortativity. Similarity is user-defined (can be age, gender, education, income, etc). If the majority of nodes connect to similar nodes, the network have high homophily.
4. Mutuality - A measurement in directed network of how much the nodes reciprocate their relationship, i.e. nodes are tied in both directions. Also referred as reciprocity. For example, A treats B as a friend (A links to B), but B might not consider B as a friend of his (B does not link back to A).
5. Propinquity – A measurement of how nodes are connected geographically.

6.1.2 Distribution Metrics

- Centrality – The measurement of importance of a node or a group. There are different method to measure centrality, including degree centrality, betweenness centrality, closeness centrality, eigenvector centrality and alpha centrality.
- Density – The proportion of number of existing ties and number of possible ties in the network.
- Bridge – A node whose removal will disconnect the network in two or more parts.

6.1.3 Distance Metrics

- Distance – The maximum number to nodes to travel between any two nodes in the network.
- Average path length – The average of all shortest path lengths in the network.

6.1.4 Segmentation metrics

1. Clique – In graph theory, clique is defined as a fully-connected subgraph in the network. In sociology, clique is referred as a group of people who interact with each other more frequently and intensely.
2. Clustering Coefficient – A measurement of likelihood that nodes tend to cluster together in the network.
3. Connectivity – The minimum number of nodes whose removal will disconnect the network. Also referred as cohesion.

In addition to basic statistics of the network, network properties and network dynamics are also of interest to researchers. The properties include, small-world, scale-free, and preferential attachment. The dynamical behaviors of the social network include spreading behavior, robustness against different types of attack, and synchronization. However, these behaviors are difficult to be observed from a (especially large) visualized network. Computer programs are needed to aid the analysis of dynamical behaviors of social network. However, this is beyond the discussion of this paper.

6.2 Tools

In analysis for online social network, tools are important and helpful for performing data analysis and network visualization. There are various SNA tools available that are powerful and free. The popular tools are listed below.

1. Pajek [10] – A free program for analysis and visualization of large-scale networks in Windows platform. This software is well-known for its comprehensive functions and performance. The authors of Pajek proposed a .net format network data file which is commonly used in network analysis. There are also books for Pajek available.

2. NodeXL [11] – An open-source network analysis plugin for Microsoft Excel. This tool is good for people who are familiar with Excel data input.

3. Gephi [12] – An open-source and cross-platform network analysis and visualization tool written in Java. Gephi is sophisticated and interactive. It also provides library/toolkit for programmers to develop their own program which incorporate network analysis and/or network visualization.

4. NetDraw [13] – A free network visualization program in Windows platform. Typically designed for social network.

5. UCINET6 [14] – A free program for network analysis in Windows platform. It is designed to be used in conjunction with NetDraw.

In this work, we used Gephi to present our result because it is cross-platform, free, powerful and easy to use.

7 Crawling Result and discussion

We have run a crawling job on Facebook with the objective of finding the social graph of the users located in Macao. According to SocialBakers Statistics [15], there are 243,860 registered Facebook users in Macao as of Feb 2013. We successfully obtained 156,297 nodes. The time to perform the whole crawling process took about six weeks by using three registered Facebook accounts in three different machines. The stop criterion is the sufficient number of samples obtained. The results

are useful for the analysis of the structure and other behaviors, e.g., virus spreading [16,17].

SNA provides some useful techniques for answering substantive questions about structures. However, it is not trivial to discover models or anomalies while dealing with huge amount of data. The network graph should be visualized which helps to simplify the work of analysis. However, it could be tricky for the computational cost of very high dimension of data. Analyzing large-scale network becomes harder because of the overlapping thousands of nodes, edges and other elements. For these reasons we should analyze data and visualize network by filtering and appropriating layout. The statistics and visualized graph are based on the data we crawled from Facebook.

7.1 Metrics and Measures

The measures for Macao SNA have been standardized: average degree, diameter, radius, density, modularity, connected components, average clustering coefficient and average path lengths are estimates based on the assumption of nodes with 100-brace filtering in graph. A short summary of some metrics is shown in Table 2. Evaluation is done using statistic tools in Gephi [12].

Graph Type:	undirected
Nodes:	156,297
Edges:	6,058,992
Average Degree:	12.774
Network Diameter:	16
Radius:	1
Graph Density:	0.009
Modularity:	0.473
No. of Communities:	100
Connected Components:	83
Avg. Clustering Coefficient:	0.491
Avg. Path length:	4.634

Table 2 Results of Overall Network Metrics

Network	Users Crawled	Links	Rad.	Diam.	PathLen.	C. Coef.
London, UK	1,241K	30,725K	11	15	5.09	0.170
Australia	1,215K	121,271K	10	14	5.13	0.175
Turkey	1,030K	42,799K	13	17	5.10	0.133
France	728K	11,219K	10	13	5.21	0.172
Toronto, ON	483K	11,812K	10	13	4.53	0.158
Sweden	575K	17,287K	8	11	4.55	0.157
New York, NY	378K	7,225K	11	14	4.80	0.146
Colombia	565K	10,242K	9	12	4.94	0.136
Manchester, UK	395K	11,120K	11	15	4.79	0.195
Vancouver, BC	314K	35,518K	9	14	4.71	0.170
Total/Average:	10,697K	408,265K	9.8	13.4	4.8	0.164
Macao	156K	6,058K	1	16	4.63	0.491

Table 3 Social Graph Measurements Comparing

Table 3 lists some statistics and social graph measurements on the ten largest regional networks in Facebook data set, as well as the totals for entire data set, which are presented in [22]. We found that the average path length of our crawled network (i.e., the Macao network) is 4.63, which is similar to other regional networks (as shown in Table 3). The radius is low but the diameter is high when compared to other large network graphs. The clustering coefficient shows graph neighborhoods of users containing dense structure.

7.2 Network Visualization

A network graph can feature an overview of the structure of the network, calling out cliques, communities, and key participants. Drawings of relational structures like social networks are only useful if they effectively convey information to the people that use them.

Network graphs with a large number of vertices can easily get too dense and large to make out any meaningful patterns. There are many obstacles like vertice occlusions and edge crossings that can make creating readable network graphs challenging.

Therefore, there should be an upper limit on the numbers of vertices and edges that can be displayed in on computer screens.

7.3 K-brace Filtering and Partition

To avoid dissatisfaction with suboptimal drawings, we strive to find optimal layouts that the information can thus be represented accurately. Moreover, analyzing large graphs is not a trivial problem, but the computational cost of visualization. For this reason, our analysis relies on filtering data calculating metrics for highlight important area and displaying only relevant information.

We applied k-brace filter to deal with the full sub-graph of all nontrivial components. Cohen [23] defined the k-brace of a graph to be the sub-graph formed by repeatedly deleting all edges of embeddedness less than k and then deleting all single node connected components. Fig.3 illustrates k-core and the k-brace, delineating the connected components of the 2-core and the 1-brace.

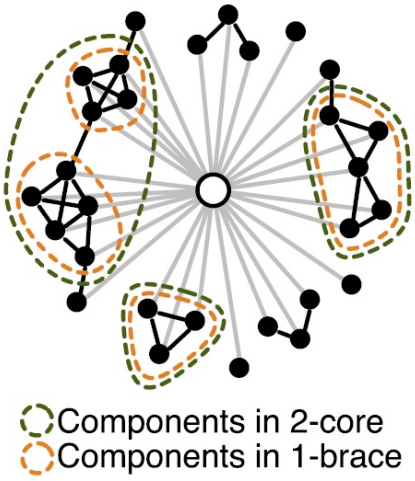


Fig. 3 1-brace filtering

We consider the components of the 100-brace, which removes small components and severs un-embedded edges and isolate "substantial" social contexts.

We also use community detection to help visualize network structure. Nodes are coloured by modularity class in order to find relatively stronger ties among subgroup members compared to non-members. Nodes are sized by degree to let look like the nodes with higher degree value is much bigger.

7.4 Macao Facebook Network Graph: Our Visual Results

After filtering, several graph layout algorithms can be used. Our result of applying these algorithms varies depending on the size and topology of the network. Fig. 4 shows the network graph visualized by using the Fruchterman Reingold (FR) layout, whereas Fig. 5 by the Yifan Hu (YH) layout and Fig. 6 by the ForceAtlas 2 (FA2) layout.

Different layouts provide different advantages and disadvantages. The FR layout is optimized for big graphs, which can prevent the mass particles from overlapping. But it does not provide cluster representation and it may produce confusing edge connection. On the other hand, the YH layout uses a fast algorithm and provides a good quality on large graphs, but it shows coarse looking, and is unable to reach a balanced position. Finally, the FA2 layout can make clusters tighter, but it shows poor local minima, and takes much computation time.

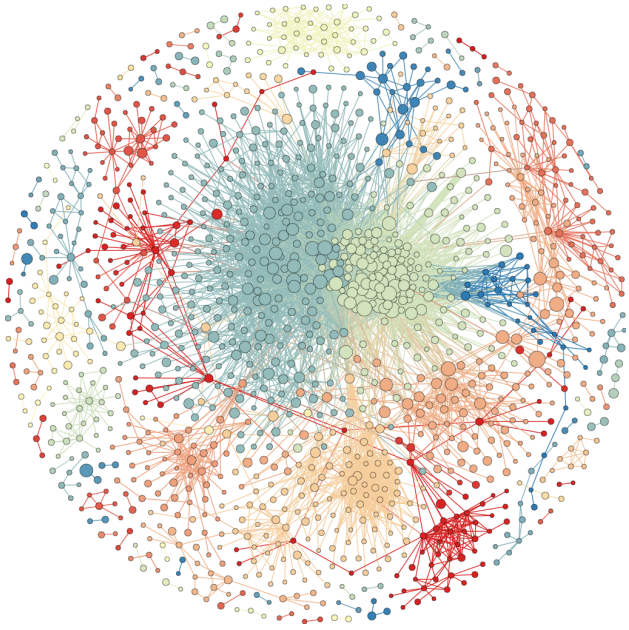


Fig. 4 Result visualized by the Fruchterman Reingold layout

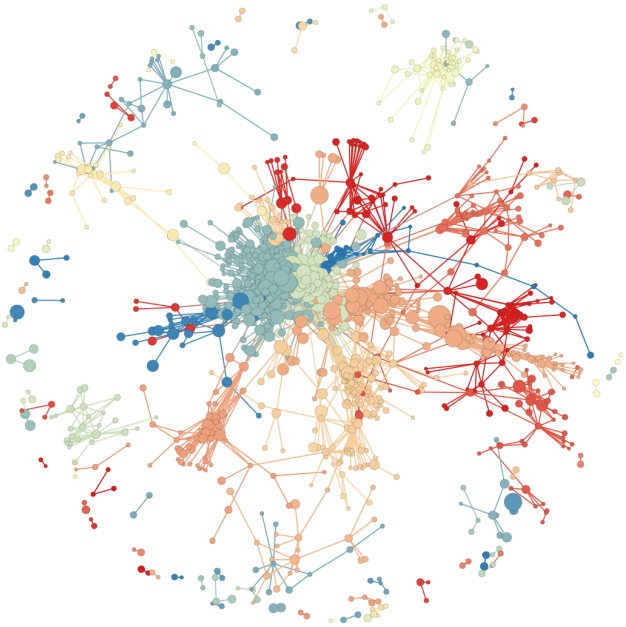


Fig. 5 Result visualized by the Yifan Hu layout

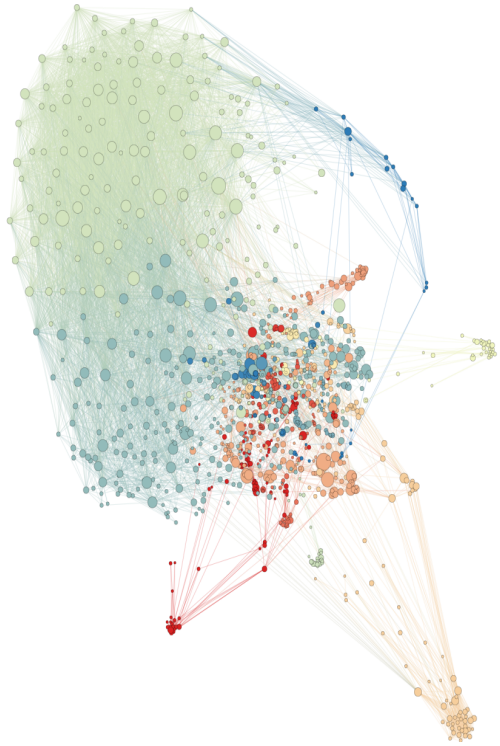


Fig. 6 Result visualized by the ForceAtlas 2 layout

8 Conclusion

In this paper, we present a practical design method of building an efficient crawler to collect data from Facebook. We discussed the key challenges during the implementation and execution of the crawler. We also provided suggested solutions to tackle the challenges. We then discussed the operation design of a crawler, the issues to consider, and methods to evaluate if the crawl succeeds. We found that the performance of crawler can be significantly improved by parallelizing crawl tasks. We implemented our own crawler and performed crawling Facebook to obtain the structure of the social network of Macao people. Then, we discussed some commonly used social network analysis techniques and visualized the crawled data using different layout algorithms. We also compared the network metrics of our graph with some previous results. We believe that the design and implementation of crawler we conclude here shed light on future OSN studies, which will increasingly rely on crawled sub-graphs.

Acknowledgements:
This work is supported by Macau Science and Technology Development Fund numbered 039/2010/A.

References:

[1] Angus Wong, Cell phones as mobile computing devices, *IT professional*, Vol.12, Iss.3, 2010, pp. 40-45.

[2] D. M. Boyd and N. B. Ellison, Social Network Sites: Definition, History, and Scholarship, *Journal of Computer-Mediated Communication*, Vol.13, Iss.1, 2007, pp. 210-230.

[3] L. Garton, C. Haythornthwaite and B. Wellman, Studying Online Social Networks, *Journal of Computer-Mediated Communication*, Vol.3, Iss.1, 1997, pp. 0.

[4] F. Schneider, A. Feldmann, B. Krishnamurthy and W. Willinger, Understanding online social network usage from a network perspective, *IMC '09 Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference, Chicago, IL, USA*, 2009, pp. 35-48.

[5] C. Steinfield, N. B. Ellison and C. Lampe, Social capital, self-esteem, and use of online social network sites: A longitudinal analysis, *Journal of Applied Developmental Psychology*, Vol.26, Iss.6, 2008, pp. 434-445.

[6] PhantomJS headless WebKit with JavaScript API [online]. Available: http://phantomjs.org/, Last Accessed on 01/02/2013.
[7] Firebug (Mozilla Firefox browser plugin) [online]. Available: http://getfirebug.com, Last Accessed on 27/02/2013.
[8] CSS Selectors [online]. Available: http://www.w3.org/TR/CSS2/selector.html, Last Accessed on 18/03/2013.
[9] CasperJS: navigation scripting & testing utility for PhantomJS [online]. Available: http://casperjs.org/, Last Accessed on 09/03/2013.
[10] Program for Large Network Analysis [online]. Available: http://vlado.fmf.uni-lj.si/pub/networks/pajek/, Last Accessed on 19/05/2014
[11] NodeXL: Network Overview, Discovery and Exploration for Excel [online]. Available: http://nodexl.codeplex.com/, Last Accessed on 19/05/2014
[12] The Open Graph Visualization Platform: Gephi [online]. Available: http://gephi.org/, Last Accessed on 20/03/2013
[13] NetDraw Software for Network Visualization [online]. Available: https://sites.google.com/site/netdrawsoftware/home, Last Accessed on 19/05/2014
[14] Ucinet for Windows: Software for Social Network Analysis [online]. Available: https://sites.google.com/site/ucinetsoftware/home, Last Accessed on 19/05/2014
[15] Social Media Marketing, Statistics & Monitoring Tools, Socialbakers [online]. Available: http://www.socialbakers.com/facebook-statistics/, Last Accessed on 25/03/2013.
[16] W. Fan, K.H. Yeung, and K.Y. Wong, Assembly Effect of Groups in Online Social Networks, *Physica A: Statistical Mechanics and its Applications*, Vol.392, Iss.5, 2013, pp. 1051-1262.
[17] R. Pastor-Satorras and A. Vespignani, Epidemic Spreading in Scale-Free Networks, *Physical Review Letters*, Vol.86, 2001, pp. 3200-3203.
[18] R. Albert and A. Barabasi, Statistical mechanics of complex networks, *Reviews of Modern Physics*, Vol.74, Iss.1, 2002, pp. 47-97.
[19] S. Ye, J. Lang and F. Wu, Crawling Online Social Graphs, *In: Proceedings of the 12th International Asia-Pacific Web Conference, IEEE*, 2010, pp. 236-242.
[20] D. H. Chau, S. Pandit, S. Wang, and C. Faloutsos, Parallel crawling for online social networks, WWW '07: Proceedings of the 16th international conference on World Wide Web, 2007, pp. 1283-1284.
[21] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, Measurement and analysis of online social networks, IMC '07: Proceedings of the seventh ACM SIGCOMM conference on Internet measurement, 2007, pp. 29-42.
[22] C. Wilson, B. Boe, A. Sala, K. P. N. Puttaswamy, and B. Y. Zhao, User Interactions in Social Networks and their Implications, 4th ACM European conference on Computer systems, 2009, pp. 205-218.
[23] J. D. Cohen, Trusses: Cohesive subgraphs for social network analysis, National Security Agency Technical Report, Fort Meade, MD, U.S., 2008