# Effective and Efficient Methodologies for Social Network Analysis

Long Pan

Dissertation submitted to the faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
In
Computer Science and Applications

Eunice E. Santos, Chair
Elisa D. Sotelino
Yang Cao
Ezra Brown
Eugene Santos, Jr.

Dec 11th, 2007
Blacksburg, Virginia, USA

Keywords: Social Network Analysis, Parallel/Distributed Computing, Anytime-Anywhere Methodology

# Effective and Efficient Methodologies for Social Network Analysis

Long Pan

## ABSTRACT

Performing social network analysis (SNA) requires a set of powerful techniques to analyze structural information contained in interactions between social entities. Many SNA technologies and methodologies have been developed and have successfully provided significant insights for small-scale interactions. However, these techniques are not suitable for analyzing large social networks, which are very popular and important in various fields and have special structural properties that cannot be obtained from small networks or their analyses. There are a number of issues that need to be further studied in the design of current SNA techniques. A number of key issues can be embodied in three fundamental and critical challenges: long processing time, large computational resource requirements, and network dynamism.

In order to address these challenges, we discuss an anytime-anywhere methodology based on a parallel/distributed computational framework to effectively and efficiently analyze large and dynamic social networks. In our methodology, large social networks are decomposed into intra-related smaller parts. A coarse-level of network analysis is built based on comprehensively analyzing each part. The partial analysis results are incrementally refined over time. Also, during the analyses process, network dynamic changes are effectively and efficiently adapted based on the obtained results. In order to evaluate and validate our methodology, we implement our methodology for a set of SNA metrics which are significant for SNA applications and cover a wide range of difficulties. Through rigorous theoretical and experimental analyses, we

demonstrate that our anytime-anywhere methodology is an effective and efficient approach for large and dynamic social network analysis.

# Acknowledgements

First, I would like to give my sincere thanks to my advisor Dr. Eunice E. Santos for her expert guidance and full support. Her inspiration makes this research work possible and her direction leads to the accomplishment of my thesis. Moreover, what she has done enriches my mind and sheds light on my future career.

Also, I am thankful to all my committee members, Dr. Ezra Brown, Dr. Elisa D Sotelino, Dr. Yang Cao, and Dr. Eugene Santos, Jr. They have awarded me with generous support and encouragement during my time at Virginia Tech. Their criticism and assistance are invaluable in my academic process.

Special thanks to Dustin Arendt, Morgan Pittkin, and Huadong Xia for their help and collaboration in various parts of this work. Their suggestions and critiques helped me to avoid a tortuous path in my research and their support helped the fast progress of my work.

Last but not least, I want to express my deep thanks to my family for everything that they have done for me. Especially, my parents and my wife have given me unconditional love and support. They are the most important people in my life. Thanks for giving me a happy family.

This work was supported in part by the Air Force Office of Scientifics Research, and by the Defense Threat Reduction Agency.

# Table of Content

Table of Figures ..... vii
Table of Tables ..... ix
1. Introduction ..... 1
1.1 Social Network ..... 2
1.2 Social Network Analysis ..... 8
1.3 Current Social Network Analysis Software ..... 11
1.4 Summary ..... 13
2. Large Social Network Analysis ..... 14
2.1 Large Social Networks ..... 14
2.2 State-of-Art in Large Network Analysis ..... 17
2.3 Challenges of Large Social Networks Analysis ..... 20
2.4 Summary ..... 23
3. Methodology Design ..... 24
3.1 Parallel/Distributed Framework ..... 24
3.2 Anytime-Anywhere Properties ..... 25
3.3 Our Methodology ..... 27
3.4 Focuses of Our Methodology ..... 34
3.5 Summary ..... 35
4. Methodology Analyses and Validation ..... 36
4.1 Fundamental Definitions and Terminologies ..... 36
4.2 Centrality Measurements in SNA ..... 38
4.3 Maximal Cliques ..... 43
4.4 Algorithms for Measuring Closeness Centralities ..... 44
4.5 Algorithms for Measuring Ego-Betweenness Centralities ..... 48
4.6 Algorithms for Maximal Clique Enumeration ..... 52
4.7 Comparisons and Summarizations of Selected SNA Metrics ..... 56
4.8 Summary ..... 60
5. Domain Decomposition & Initial Approximation Phases Implementation ..... 61
5.1 Domain Decomposition Phase Implementation ..... 61
5.2 Initial Approximation Phase Implementation ..... 66
5.3 Summary ..... 67
6. Recombination Phase Implementation ..... 68
6.1 General Anytime Recombination Algorithm's Design ..... 68
6.2 General Anywhere Recombination Algorithm's Design ..... 69
6.3 Ego-Betweenness Centrality Recombination Approach ..... 71
6.4 Closeness Centrality Recombination Algorithm I – The Anytime Approach ..... 74
6.5 Closeness Centrality Recombination Algorithm II – The Anywhere Approach ..... 81

6.6 The Anytime Anywhere Recombination Approach for Closeness Centrality ..... 90
6.7 Maximal Clique Enumeration Recombination Algorithm I – The Anytime Approach ..... 92
6.8 Maximal Clique Enumeration Recombination Algorithm II – The Anywhere Approach ..... 93
6.9 The Anytime Anywhere Recombination Approach for Maximal Clique Enumeration ..... 100
6.10 Summary ..... 101
7. Experimental Results and Analysis ..... 103
7.1 Experiments Setup ..... 103
7.2 Experiments on Ego-Betweenness Centrality Measurement ..... 104
7.3 Experiments on Closeness Centrality Measurement ..... 111
7.4 Experiments on Maximal Clique Enumeration ..... 125
7.5 Summary ..... 133
8. Conclusion & Future Work ..... 134
Reference ..... 136

# Table of Figures

Figure 1-1. A friendship network of a small class. ..... 4
Figure 1-2. Examples of different types of social networks ..... 8
Figure 2-1. World-Wide Web page network ..... 15
Figure 3-1. Our anytime-anywhere methodology's architecture. ..... 29
Figure 4-1. An example social network. ..... 41
Figure 4-2. The selected ego *v* and its alters. ..... 42
Figure 4-3. The ego-network of node *v*. ..... 42
Figure 4-4. Pseudo-code of Dijkstra's algorithm. ..... 45
Figure 4-5. Pseudo-code of Floyd's algorithm. ..... 47
Figure 4-6. Pseudo-code of modified Dijkstra's algorithm for ego-betweenness measurement ..... 51
Figure 4-7. An example graph of maximal clique enumeration problem ..... 55
Figure 5-1. The architecture of the Domain Decomposition phase. ..... 62
Figure 5-2. The structure of graph domain decomposition approach ..... 64
Figure 6-1. An example of the ego-betweenness dynamic change's effect range ..... 72
Figure 6-2 Algorithm I: the anywhere recombination approach for ego-betweenness centrality measurement ..... 73
Figure 6-3. An example of super-graph based on decomposition of original graph ..... 75
Figure 6-4. Algorithm II: the anytime recombination algorithm for closeness centrality measurement ..... 77
Figure 6-5. The super-graph obtained based on partitioning of the example graph ..... 78
Figure 6-6. Edge pool for shortest paths at $P_1$ ..... 79
Figure 6-7. Algorithm III: the anywhere recombination algorithm for closeness centrality measurement when edge weight is decreased ..... 83
Figure 6-8. Algorithm IV: the fully dynamic anywhere recombination algorithm for closeness centrality measurement ..... 85
Figure 6-9. Algorithm V: the anytime anywhere recombination algorithm for closeness centrality measurement ..... 91
Figure 6-10. Algorithm VI: the anytime recombination algorithm for maximal clique enumeration ..... 93
Figure 6-11. Example for finding maximal cliques with edge addition ..... 95
Figure 6-12. Algorithm VII: the anywhere recombination algorithm for added edge for maximal clique enumeration. ..... 96
Figure 6-13. Algorithm VIII: the anywhere recombination algorithm for added edge for maximal clique enumeration. ..... 98
Figure 6-14. Algorithm IX: the fully dynamic anywhere algorithm for maximal clique enumeration ..... 99

Figure 6-15. Algorithm X: the anytime anywhere recombination algorithm for maximal clique enumeration ..... 101
Figure 7-1. Ego-betweenness centrality measurement: serial vs. parallel. ..... 106
Figure 7-2. Time cost for adopting 64 random edge changes for ego-betweenness centrality measurement..... 108
Figure 7-3. Performance comparison between handling random changes and handling max degree changes for ego-betweenness centrality measurement. 109
Figure 7-4. Relative cost for adopting 64 edge changes for ego-betweenness centrality measurement..... 111
Figure 7-5. Running time comparison of UCINet and our serial algorithm for closeness centrality measurement..... 112
Figure 7-6. Closeness centrality measurement: serial vs. parallel. ..... 114
Figure 7-7. The anytime property of our approach. ..... 115
Figure 7-8. The time cost of our system to incorporate an edge with decreased weight. ..... 117
Figure 7-9. Performance comparison between handling random increased edge weights and handling max degree increased edge weights for closeness centrality measurement..... 118
Figure 7-10. The relative cost for adopting an increased edge weight for closeness centrality measurement. ..... 120
Figure 7-11. The time cost of our system to incorporate an edge with decreased weight. ..... 122
Figure 7-12. Relative costs for adopting a decreased edge weight for closeness centrality measurement..... 123
Figure 7-13. The relative cost for adopting a decreased weight for closeness centrality measurement..... 125
Figure 7-14. Time costs for finding all maximal cliques contained in graphs. ... 127
Figure 7-15. The anytime property of our approach for maximal clique enumeration ..... 129
Figure 7-16. The performance of our maximal clique enumeration anytime approach for dense graphs. ..... 131
Figure 7-17. Time costs for adopting one random dynamic change for maximal clique enumeration..... 132
Figure 7-18. Relative cost for adopting one dynamic edge change for maximal clique enumeration..... 133

# Table of Tables

Table 1-1. The social data about friendships between students in a small class. .4
Table 4-1. Summary of selected SNA metrics. .....60
Table 7-1. Maximal cliques contained in each graph .....126
Table 7-2. Maximal cliques contained in graphs with density as 10%.....130
Table 7-3. Maximal cliques contained in graphs with density as 15%.....130
Table 7-4. Maximal cliques contained in graphs with density as 20%.....130

# 1. Introduction

Understanding the nature of relationships and connections between entities is key towards understanding a variety of phenomena throughout multiple disciplines. The concepts of how a disease is spread, or how people are influenced by information are all examples of the need to understand and analyze interactions and relationships.

These concepts are the building block in the field of Social Network Analysis. Social Networks (SN) are graphs employed to represent the structure of interactions/relationships among people, or any types of entities. Social Network Analysis (SNA) has been studied by researchers for more than a century. As the broad application of electronic data, numerous large social networks emerge from various fields.

While there have been a multitude of results and analysis techniques that have been used in SNA, as we will discussed, current-day approaches are not able to effectively deal with social networks that are large-in-scale and dynamic. As such, this will be the primary focus of this dissertation. Large network analysis is a non-trivial task. It introduces new challenges due to long processing time, large computational resource requirement, and graph dynamism. In order to effectively and efficiently analyze large and dynamic social networks, new techniques and methodologies need to be developed. In this dissertation, we describe an anytime-anywhere methodology based on a parallel/distributed computational environment for large social network analysis. In our methodology, large social networks are decomposed into small parts and a coarsen-level analysis (partial results) of the network is generated based on analyzing each part. These partial results are incrementally refined over time. Also, during the analysis process, network dynamic changes will be effectively and efficiently adapted.

In what follows, we first provide a brief introduction of social network and social network analysis. Then, we will specifically discuss about popularity, importance, and special properties of large social networks. According to the challenges of large social network analysis, we present the design of our anytime-anywhere methodology. Next, we provide both theoretical and experimental evaluation and validation by implementing our methodology on a selected set of SNA metrics. Finally, we present our conclusion and discuss future work. Part of the work and figures presented in this dissertation have already been published in [SantosPAXP'06, SantosPAP'06]. Discussion and results can also be found in [SantosPA'07]

Before we discuss specific research and design issues for large and dynamic social networks, we will first provide important background information and introduction of fundamental concepts of social networks and social network analysis (SNA).

## *1.1 Social Network*

In this section, we will present key definitions in the field of social networks, types of social networks, and ways in which social network data are gathered.

### a) Background Definitions

Social networks have typically been defined as graphs representing social relationships between people or organizations. Each node, also called an *actor* or *vertex*, in a graph represents an individual person or a group of persons. An edge connecting two nodes, also called a *tie*, represents relationship between the objects represented by these two nodes. Using graphs to represent social data enables social analysts to completely and rigorously describe, manipulate, and analyze the structural information embedded in social relationships. Also,

graph-theoretic concepts grant researchers a mathematical and systematic framework that can extend researchers' methodologies to other fields. In a general point of view, social networks can be used to represent, identify, and measure any type of correlations between any kind of entities, such as words, web pages, people, organizations, animals, cells, computers, and other information or knowledge processing entities [Krebs'06]. Thus, Social Networks have broad and successful applications in sociology, epidemiology, biology, criminology, and economics [Kadushin'05].

**b) Examples of Social Networks**

Throughout physics, biology, social sciences and engineering, an abundant number of systems take the form and structure of networks. In order to facilitate a clear understanding of social networks, we present one example below. This example is just a simple graph which is used to help readers to understand basic concepts in social networks. This graph is a network of friendships between students in a small class.

First assume that we have already obtained data about friendships among students which are shown in Table 1-1. Details about ways for gathering such social data will be discussed in a later section. In this table, the diagonal elements are all blanks. This is due to the fact that in friendship analysis, we do not need to consider if a person is a friend of himself/herself. The other elements in the table are binary. This means that data in the table only represents that two people are either friends or not. Based on the data contained in Table 1-1, we can build the network of friendships between students in the small class. The network is shown in Figure 1-1.

**Table 1-1.** The social data about friendships between students in a small class.

| | John | Susan | Tom | Jack | Alice | Jeff | Mike | Tiger | Jane |
|---|---|---|---|---|---|---|---|---|---|
| John | -- | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| Susan | 0 | -- | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Tom | 1 | 1 | -- | 1 | 0 | 0 | 0 | 0 | 0 |
| Jack | 1 | 0 | 1 | -- | 0 | 1 | 1 | 1 | 0 |
| Alice | 0 | 0 | 0 | 0 | -- | 1 | 0 | 0 | 0 |
| Jeff | 0 | 0 | 0 | 1 | 1 | -- | 1 | 1 | 0 |
| Mike | 0 | 0 | 0 | 1 | 0 | 1 | -- | 1 | 0 |
| Tiger | 0 | 0 | 0 | 1 | 0 | 1 | 1 | -- | 0 |
| Jane | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -- |

In this network, each node represents a student in the class. Two nodes are connected with an edge if they are friends. In this example, two students are friends if the value of the corresponding element in the table 1-1 is 1. If the value is 0, these two nodes are not friends and not directly connected.

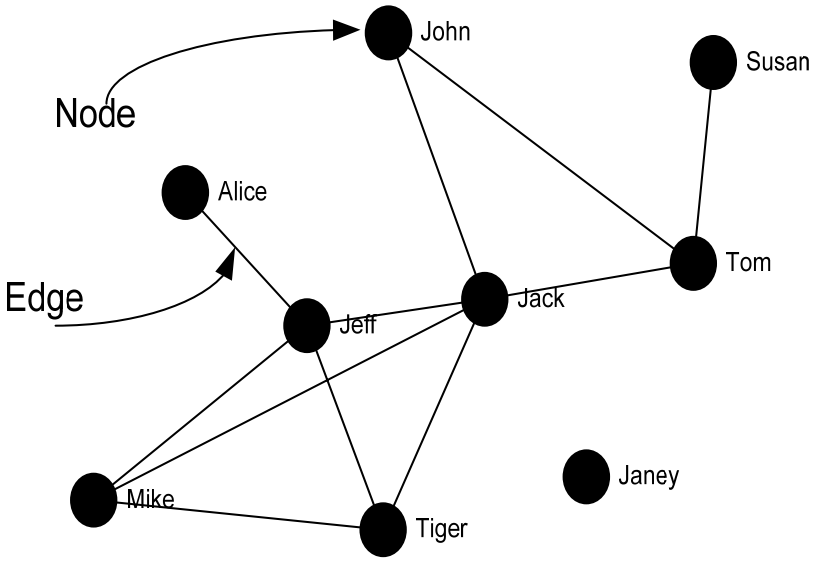



**Figure 1-1.** A friendship network of a small class.

### c) Social Network Data Gathering

In the previous section, we presented several examples of social networks of different kinds of entities and interactions. We also provided a simple example to show how to build social networks based on the obtained social data. However, before building social networks, there is a very important problem researchers have to face: how to acquire elementary data elements for building social networks? In order to give a complete and accurate description of interactions between individuals, researchers have done a lot of work on social data gathering techniques focusing on how to identify the population, how to measure relationships, etc. Since this is not our research focus, rather than going into specific details about this topic, we will instead provide a brief overview of popular social network data collection methods in the following part of this section. For interested readers who want further details on this topic, please refer to the second chapter of the textbook [CarringtonSW'05], or the first chapter of the online textbook [HannemanR'05].

Currently, there are mainly two kinds of approaches for social network data gathering: elicitation and registration [NooyMB'05]. Elicitation acquires interaction information via the questionnaire/survey. Registration acquires interactions through extracting from registered information, such as membership lists, email records, author records of scientific articles, etc.

In the early SNA research, questionnaire/survey was the method primarily used. In this method, questions about interactions are proposed and respondents are required to report their answers. Data gathered by this kind of method may be quite inaccurate and subjective [AlberB'02, CarringtonSW'05, Newman'03a, NooyMB'05]. It is hard to obtain complete set of data by survey/questionnaire. Also, the gathered data are affected by subjective biases of respondents. For example, people will have different definitions of friendships and different perceptions on friendship strength. A social network of people's friendship built based on data gathered by this type of method will seriously skewed due to

different definitions/perceptions of friendship. A comprehensive review of this topic can be found in [Marsden'90]. Moreover, survey/questionnaire method has high-labor cost. It will take social scientists and network researchers a myriad of efforts to gather data for a network of even middle size (several thousands of nodes). This intensive labor cost considerably limits the size of networks to be studied.

Through fast developments of computer technologies and universal applications of computers, automated data acquisition are found in most, if not all, fields. Interactions between objects can be stored as or implicated by electronic data. For example, co-authorship of research articles can represent the collaboration between research scientists. If two authors appear on the same paper, there will be a collaboration connection between them. Through rapid growth of network size and data-sharing techniques, huge databases of social interactions have emerged in various fields. For instance, there are many large databases that maintain records of article authors in publications of miscellaneous research fields [BarabasiJNRSV'02, Newman'01]. MEDLINE for example, the database that covers published papers on biomedical research, has about 2 million records from 1995 to 1999 [Newman'01]. Electrical registered information can provide even larger amount of data. Using electronic data, we can have more objective definitions of interactions. For example, the cooperation between scientists can be measures as the number of publications they published together. However, for some cases, how to interpret the physical meaning of the interaction data gathered by this mechanism and how to mapping them to system behavior needs more consideration.

**d) Types of Social Networks**

According to broad applications of SNA, there are many types of social networks. Social networks can be classified based on the combination of attributes and measurements of nodes and ties. In social networks, there could be different kinds of nodes or the same kind of nodes with various weights. For instance,

affiliation networks [WassermanF'94] contain two kinds of nodes: events (such as corporations/organizations) and actors. Ties between events and actors usually represent relationships of membership/participation [CarringtonSW'05]. Nodes in a social network can have various weights that can indicate, for example, their importance.

Similarly, a social network could contain multiple types of ties or the same type of ties with different weights. A network with multiple relations are called *multi-relational network*. A multi-relational network, for example, may contain relationships as friendship, collaboration, and co-membership. These relationships could have different importance or strength which is represented as the edge weight in a graph. Taking a friendship network as an example, people may use a number from 0 to 5 to indicate the strength of friendships between them. Also, relationships between objects may be non-symmetric. Still discussing friendship networks, person A taking person B as a friend does not necessary requires that B takes A as a friend too. Thus, ties may have directions. Symmetric ties can also be taken as directed ties on both directions.

We summarize the types of social networks in Figure 1-2. A real social network can be a combinatory of network types shown in this figure. The examples provided are simple and fundamental social networks. Clearly, there are various types of social networks. A comprehensive review can be found in [CarringtonSW'05, NooyMB'05, WassermanF'94]. In our research, we mainly focus on the most popular and fundamental type of social networks: graphs which have only one type of actors with the same weight and only one type of ties but with various weights. Without specific declaration, networks analyzed in this document are this type of graphs.

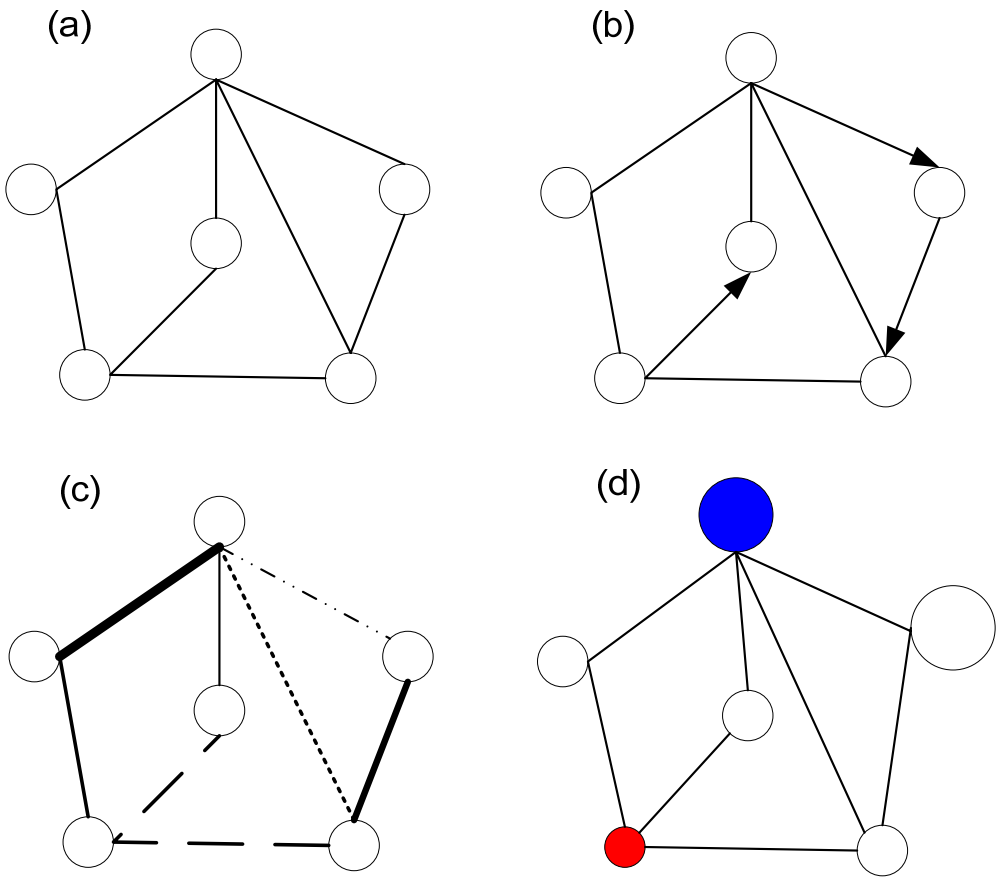



**Figure 1-2.** Examples of different types of social networks: (a) a social network with single type of nodes and undirected ties, (b) a social network with directed and undirected ties, (c) a social network with various types and weights of ties, (d) a social network whose nodes have different types and weights

## *1.2 Social Network Analysis*

In previous sections, we have introduced definitions and types of social networks. Also, we discussed different ways to gather social data and build networks. However, knowing how to build networks does not imply understanding the contents contained in networks. How to “dig up” embedded structural information from social networks falls into the field of Social Network Analysis.

### a) What Are Analyzed in Social Network Analysis?

Researchers define SNA as (a) “the mapping and measuring of relationships and flows between people” [Krebs’06], (b) the techniques “focusing on uncovered patterns of people’s interaction” [Freeman’02], (c) a set of methods for the investigation of relational aspects of social structures [Scott’92]. Essentially, these definitions are equivalent. They all emphasize that social network analysis is focused on the study of structural information contained interactions between entities.

The study of SNA is primarily focused on interactions between entities instead of entities themselves. In other words, measurements and analysis of social networks are mainly based on ties/edges between actors/nodes other than just attributes of actors. This does not indicate that attributes of actors are useless. In many cases, actors’ attributes will help researchers to verify hypothesis of social behaviors and analyze specific social phenomena. For example, in a friendship social network of students in colleges, researchers may find a universal social phenomenon and draw the conclusion that ethnicity has considerable effects on friendships between people.

However, the study of SNA is mainly from aspects of structural properties and patterns of entities’ interactions. Patterns of people’s interactions are important features of lives of individuals who display them [Freeman’02]. Most SNA researchers, if not all, have the same assumption that structure implicates and affects functions. What SNA measures/analyzes are structural properties of individuals or groups of individuals in a network. These measurements includes how individuals are connected with others, how individuals will affect connections between others, how groups of individuals are connected in a network. Also, from a global point of view, SNA researchers are usually interested in such questions as Is the whole network connected?; Is the network densely connected?; Can the network be decomposed into blocks based on the individuals connections?; etc.

**b) Social Network Analysis Broad Applications**

As we discussed, social network analysis techniques can be applied to study structures of any types of interactions/relationships between any kinds of entities. From late 1970s, SNA techniques have gained massive attentions, considerable developments, and successful applications in broad fields [CarringtonSW'05].

For example, SNA techniques are used in organization management. In current companies and government agents, there is more and more cooperation and information sharing between workers. Using SNA tools on collaboration and/or information-sharing networks, managers can easily find the "important to go people", and build appropriate management strategies to improve efficiency.

Combating terrorism is another field where SNA techniques have important and successful applications. Terrorist organizations have special structures on recruitment, evolution, and radical ideas diffusion [Ressler'06]. SNA tools can be used to identify these unique organization structures and provide critical information for terrorist detection and terrorism prediction.

Social Network Analysis techniques also have been successfully applied in epidemiology. A lot of researchers try to analyze the spread of diseases based on the interactions between people.

A SNA researcher, Valdis Krebs, listed a number of recent successful applications of SNA in [Krebs'06]. A selected set of applications are listed below:

- "*Examine a network of farm animals to analyze how disease spreads from one cow to another*
- *Discover emergent communities of interest amongst faculty at various universities*
- *Reveal cross-border knowledge flows based on research publications*
- *Determine influential journalists and analysts in the IT industry*
- *Unmask the spread of HIV in a prison system*

- *Map executive's personal network based on email flows*
- *Discover the network of Innovators in a regional economy*"

## *1.3 Current Social Network Analysis Software*

Modern social network analysis has been studied for more than seventy years and many researchers and commercial companies have put huge amounts of efforts on developing computer software tools for social network analysis. Currently, numerous commercial/free SNA software tools are available. These tools can perform comprehensive analysis on small social networks and provide significant insights for fine-grain interactions or small domain spaces.

### a) Popular Social Network Analysis tools

Currently, there are many comprehensive tools developed for SNA, such as UCINET [BorgattiEF'02], Pajek [BatageljM'04], Agna [Benta'04], NetDraw [Borgatti'02], NetMiner [Cyram'04], MultiNet [RichardsS'03], StOCNET [BoerNHSSZ'04], etc. A brief review of these software tools can be found in [CarringtonSW'05].

From functionality, SNA software tools can be primarily classified into two types. One type, including NetDraw, NetMiner, and Pajek, focuses on the visualization of networks. Developers of these tools believe that human eyes are powerful network analytic tools. Visualizing networks will help analysts easily understand structure information contained in them. The other type of tools is based on text reports of SNA measurements and analysis. UCINET, Agna, and MultiNet all belong to this type. These two types of tools are usually employed jointly to facilitate a more comprehensive analysis of social networks.

There is a special type of tools, such as StOCNET, which provides statistical analysis of networks. This type of tools is built based on statistical models of

social networks and can provide a global-scale analysis of networks based on a set of statistics, such as degree variance, index of heterogeneity, dyad and triad census, etc [CarringtonSW'05].

**b) Issues in Current Social Network Analysis Software**

Although various types of SNA software tools have gained a lot of success in extensive research fields, their development is still maturing. There are still considerable problems in these tools. One of the most critical problems is that current SNA software tools lack scalability and cannot be used to analyze large-scale and dynamic interactions.

Currently, most SNA software tools are designed for analyzing small social networks. This is due to the history of social network applications. Modern social network analysis theory originated in the 1930's [BarabasiJNRSV'02]. As we mentioned, at that time, survey/questionnaire was the primary method used. This labor-intensive method substantially limited the size of networks obtained. The social networks analyzed at that time were usually in sizes of tens, at most several hundreds, of nodes. SNA software packages are primarily designed according to requirements of these classical small networks. Most approaches used in current SNA tools are built based on serial algorithms. Moreover, usability of software tools for visual exploration/analysis of social networks will seriously degrade, even become useless, as the size of networks increases. When a network is large and complex, using human eyes to identify/extract structural information is not only quite burdensome but also near-impossible to achieve a complete and accurate analysis. What is even worse is that a lot of elementary SNA measurements cannot be obtained in current SNA tools due to the large size of networks. For example, few of current SNA tools can measure *centrality* (defined in later chapters), one of the most used SNA metrics, when a network is large. As we show in [SantosPAXP'06, SantosPAP'06, SantosPA'07], using a computer with 512MB of memory, the maximum size of a network for UCINET to load and perform closeness centrality measurement is about 15,000

actors. For Pajek, the maximum allowable network size is about 5,000 actors. However, there are many social networks with size larger than 15,000. For example, a large citation network presented in [Redner'98] has 783,339 nodes.

Moreover, current SNA software and algorithms are "all-or-none" approaches. That is, there is no means to stop algorithms in midway to obtain a meaningful partial result. In fact, it is not simply the means to stop but the fact that these approaches typically would provide non-meaningful results except for the final results. For time-critical applications, providing a coarse-level useful analysis of the network within a short time may be quite helpful for analysts.

Furthermore, current SNA tools cannot adapt dynamic behaviors of large networks. To the best of our knowledge, no current SNA software tools have the ability to incorporate dynamic changes in networks into on-going processing of network analysis. If a network whose connection structure has been changed during processing, the only way to get a meaningful analysis of the current network is to stop, re-load, and re-analyze it from scratch. For small networks, this approach seems to be feasible since it only takes a short time for re-analyzing. However, this kind of approach will have significant costs in large networks analysis. We will discuss details of this problem in Chapter 2.

## *1.4 Summary*

In this chapter, we discussed general definitions and broad applications of social networks and social network analysis. We also introduced available SNA software tools. Currently, SNA computer tools have the ability to provide significant insights for studying small-scale interactions between objects. However, they have critical issues for analyzing large and dynamic social networks.

# 2. Large Social Network Analysis

Social network analysis has been applied in a broad range of research fields. Due to the wide usage of computerized data acquisition and rapid developments of networked information-sharing techniques, numerous types of large social networks have emerged in a wide range of research fields. These large networks play critical roles in studying structural properties, understanding social phenomena, and predicting system behaviors from the point of view of large-scale interactions. However, analyzing large and complex social networks introduces specific crucial and fundamental problems which have not been considered nor addressed in current SNA tools.

## *2.1 Large Social Networks*

There are many large social networks that have emerged from various fields. These include networks of acquaintance/communication [Compbell'04, NowellNKRT'05, LiveJournal, MySpace, FaceBook], phone calls [AielloCL'00], collaboration [Newman'01, BarabasiJNRSV'02], sexual contact [LiljerosEASA'01], paper citation [Redner'98], metabolic networks [GuimeraA'05], World-Wide Web (WWW) pages networks [Adamic'99, BroderKMRRSTW'02], the Internet networks [GovindanT'00], food webs [WinemillerL'03], linguistic networks [CanchoS'01], etc. In following paragraphs, we will briefly introduce several popular large networks obtained based on real-world databases.

### a) Networks of Acquaintance/Communication

Currently, there are various social utilities on the Internet, such as LiveJournal (LJ) (www.livejournal.com), MySpace (www.myspace.com), FaceBook (www.facebook.com) etc. These tools provide a platform for social

communication and a mechanism for people to connect with their friends and build their communities. Social networks, which are built on registered data of these social utilities, can help social analysts to study large-scale interactions between people in the aspect of acquaintance/friendship. Usually, the size of these social networks can be very large and the data are inherently changing and evolving. For example, by Oct 9th, 2006, LJ has a total of 11,322,901 users out of whom 1,889,233 users are active. There are approximately 50,000 updates within every 24 hours.

### b) Networks of World-Wide Web Pages

Researchers recently began to build and study large networks of World-Wide Web pages [Adamic'99, BroderKMRRSTW'02] in order to a) understand the sociology of content creation of the Web, b) analyze the behavior of and provide valuable insights into Web algorithms for gathering, searching and discovering information, c) and predict the evolution of Web structures. For example, a Web page network studied in [BroderKMRRSTW'02] has over 200 millions pages and 1.5 billion links. In this network, nodes are web pages (documents). Edges are hyperlinks (URL's) pointing from one document to another [AlberB'02]. A simple example network of World-Wide Web pages is shown in Figure 2-1.

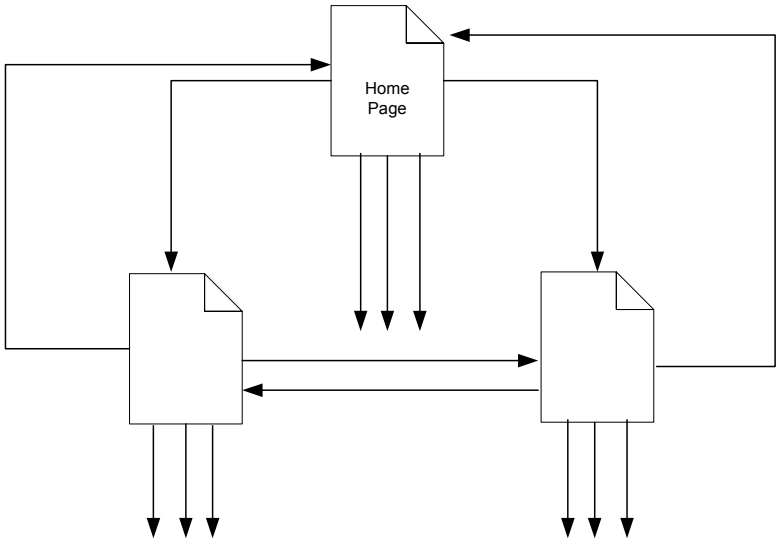



**Figure 2-1.** World-Wide Web page network

### c) Collaboration Networks: Co-Authorship Networks and Movie Actor Networks

Studying human collaboration is always an important topic in sociology. Currently, there are two popular types of networks employed by researchers to analyze large-scale interactions between people based on their collaborations. They are movie actor collaboration networks and science co-authorship collaboration networks.

Most movie actor networks are built based on the Internet Movie Database (www.imdb.com) which contains all movies and their casts since the 1890s. In these networks, nodes represent actors and ties between nodes represent that the connected actors acted in the same movie at least once. Obviously, this network is dynamic since new movies and actors keep joining the database. For example, in 1998, the movie actor network contained 225,226 nodes [WattsS'98]. In 2000, the size of network increased to 449,913 [Newman'00].

Studying the co-authorship of scientific papers is an effective way to investigate the collaboration between scientists. In co-authorship networks, nodes are authors and two nodes are connected by an edge if the corresponding two authors write at least one paper together. Two examples of large co-authorship networks can be found in [Newman'01, Barabasi01]. The size (number of nodes) of the co-authorship network based on the database MEDLINE (biomedical research) is 1,520,251 [Newman'01]. The size of co-authorship network for neuron-science is 209,293 [Barabasi01].

### d) Citation Networks

Usually, citation networks are based on databases of scientific publications. These networks are employed to study scholar communication, as well as popularity and evolution of technologies. In this network, nodes are scientific papers. One node will be connected with another by a directed edge if its corresponding paper cites the paper represented by the other node. An example

of large citation network can be found in [Redner'98]. In this example, the network has the size of 783,339.

**e) Linguistic Networks**

Linguistic networks are special networks employed to study language organization and generation based on word interactions. These networks are built based on words co-occurrences. In a linguistic network, nodes are words. Two words are connected by an edge if they appear next to or one word apart from each other. Details of linguistic networks can be found in [CanchoS'01]. The size of the word network presented in [CanchoS'01] is 460,902.

Currently, there are numerous other large networks being studied in various research fields. Due to the space limitation, we cannot introduce all of them. For interested readers who want further details of this topic, please refer to comprehensive reviews of large and complex networks in [AlberB'02, Newman'03a].

## *2.2 State-of-Art in Large Network Analysis*

Large and complex social networks have already attracted considerable attentions from SNA researchers. There are many papers discussing special structural properties obtained from large networks. Most of these researches are done in a statistical fashion. A good review of this topic can be found in [AlberB'02, Newman'03a]. Here, we will only briefly discuss the contributions that have been made and the potential pitfalls in their approaches.

**a) Special Structural Properties of Large Social Networks**

To the best of our knowledge, most, if not all, current researches on large and complex networks are dealing with using some form of statistical parameters to describe and analyze structural characteristics of large networks. Based on their

results, researchers found that the graph model which can be used to generate real-world's large networks is quite different from the classical random-graph model, the Erdös-Renyi model [Erdös59], which is proposed in 1959 and used to analyze small networks. Recently, various theories and graph models have been proposed for large and complex social networks [AlberB'02, Newman'03a]. All these models and theories are mainly built based on three observed special structural properties of large networks. These properties are listed below.

- Small-world effect [Milgram'67, Adamic'99, WattsS'98]: most pairs of nodes are connected by short paths through networks. In other words, the distance between any pair of actors is much smaller than the graph size. Usually, the average value of shortest paths is increased as a logarithm of network size.
- Degree scale-free distribution [Price'65, DorogovtsevM'01, Strogatz'01]: node degree is defined as the number of edges that connect this node with others. The distribution of node degree in a network follows the power-law. More precisely,

  $$p_k \propto k^{-\alpha}$$

  where $p_k$ is the probability that a node has a degree as $k$, and $\alpha$ is a constant and usually $\alpha$ is a value between 1.6 and 3.0 [Newman'03a]. This means that in a large network, we have a great amount of nodes with small degrees and a small tail of nodes with large degrees.
- High clustering coefficient [AlberB'02, FronczakHJS'02, Newman'03a]: in [Newman'03a] the cluster coefficient is defined as:

  $$C = \frac{3 \times number\ of\ \ triangles\ in\ the\ network}{number\ of\ \ connected\ triples\ of\ \ vertices}$$

  This property is also called as high transitivity. Real-world large networks have much higher transitivity than networks generated by the random graph model presented in [ErdosR'59].

These three properties are detected by statistical analysis from a great amount of real-world's large social networks. It seems that they are common properties for

real-world large social networks. These properties have shown differences between large-scale interactions and small-scale interactions, and highlighted the necessity and importance of large social network analysis.

**b) Pitfalls in Current Large Network Analysis Approaches**

Although the achievements of current SNA research on large social networks are exciting, they are far from satisfactory. First, all results/theories presented in the previous subsection are only based on a few basic and simple structural measurements of large social networks, such as node degree, network density, connectivity, and diameter, etc. Many complicated but more crucial structural explorations including centrality measurements, cohesive subgroup detection, roles and positions detection, and blockmodeling, which are frequently used in classical SNA on small-scale interactions, have not been studied yet. This is not because such investigations are not important. Rather, the reality is that current SNA software tools have poor ability to handle large networks.

Second, most of these analyses of large networks are done in the aspect of statistics, which means that networks are explored from the global-scale. Usually, researchers use some statistics, such as clustering coefficient, degree distribution, to represent the structure of the whole network. Besides the problem for choosing appropriate statistics for large networks, details of individuals inside a network are omitted. However, local-scale analysis is indispensable in social network analysis. No matter how large the network size is, detecting and analyzing actors or groups of actors with different structural characteristics is always one of the essential tasks for SNA.

Lastly, the common method used to achieve statistics of these large networks is sampling. Based on some strategies, SNA researchers sample a large network, and statistics of the whole network is generated from the analysis of these samples. New questions are introduced by this kind of approaches such as How to sample the network to get the best analysis results?; How many samples are

enough to obtain reliable results?; And how accurate is the analysis obtained by these samples? In order to answer all these questions, quantitative tools for social network analysis is required.

## *2.3 Challenges of Large Social Networks Analysis*

Analyzing large social networks is not a trivial task. Large social network analysis introduces a multitude of new research issues. Before discussing its challenges, let us first explain why we should bother with analyzing large social networks.

### a) Why Should We Analyze Large Social Networks?

First of all, large social networks have their own special structural properties. These properties cannot be obtained by simply scaling up small networks. It has been shown that large social networks have graph models which differ from those of small networks [AielloCL'00, AlberB'02, Newman'03a]. The structural properties of large networks and corresponding decisions-making strategies based on large social network analyses cannot be directly borrowed from what we achieved from the analysis of small networks.

While some researchers may argue that their interests are only focused on a small number of nodes and ties so that large social network analysis seems unimportant to them. In some cases, researchers can use some priori-knowledge of the problem to designate the nodes/ties they want to analyze. However, in many cases, researchers usually have no clear idea about which nodes/ties are relevant to their study. Moreover, most, if not all, small social networks are contained in and/or extracted from large-scale interactions. Individuals or groups of individuals are nested in large networks and relevant/useful objects are always coupled within their contexts. Without elaborative analysis of every element in networks, it is near-impossible to achieve an accurate and complete data set for small-scale analysis.

As we discussed in chapter 1, SNA tools can successfully handle small networks. However, they are not suitable for analyzing large-scale interactions. This is because that due to historical reasons, these tools are originally designed for analyzing small networks. Comparing with small social networks, large and complex social networks introduce new challenges, which are not addressed in current SNA tools. After carefully studying problems of large and complex social network analysis, we found that there are mainly three fundamental challenges needed to be addressed if an approach wants to effectively and efficiently analyze large social networks. These challenges are: long processing time, large computational resource requirement, and graph's dynamism. In the following paragraphs, we will briefly discuss these challenges.

**b) Long Processing Time**

An obvious characteristic of large networks introduced in chapter 2.1 is the huge network size (number of nodes contained in a network). Sizes of most of these large networks are easily at least several tens of thousands. The size of the friendship/acquaintance network based on the LiveJournal database is even more than 10 million. Moreover, these networks keep expanding. Through rapid developments and broad applications of electronic monitoring techniques, more and more large social networks will arise and current social networks will become larger and larger. As network size increases, the time for analyzing networks grows rapidly. Usually this growth is not linear. The growth of work load for comprehensively analyzing a large network can easily go to the second or third order of graph size. Consider the problem of measuring how far away each actor is from each other (all-pairs shortest distances). The computational work increases approximately at the speed of $n^3$ where $n$ is network size. Although computer power has been grown fast, handling large networks will take a great amount of time even if we use the fastest single processor available. However, in many applications, time is vital. For some time-critical applications such as criminal/terrorist detection and disease spread mitigation, it may be too late to

prevent disasters from happening by the time analysis results are returned from SNA tools.

### c) Large Computational Resource Requirement

Processing large social networks will require a great amount of computational resources, such as memories in computers. Every social network analysis package runs on a single computer making it bottlenecked both by processor speed and memory size. A 32 bit processor cannot address more than $2^{32}$ bytes of memory limiting the total system memory to approximately 4GB. Computing the shortest paths for all pairs of actors requires $n^2$ memory where $n$ is the number of actors. If we allow 4 bytes per actor then the maximum number of actors allowable in an *all in memory* serial SNA is $\sqrt{2^{32}}/_{4} \approx 16{,}000$ actors. Thus, we can see that it is infeasible to employ a single processor to perform analysis on large social networks.

### d) Graph Dynamism

Almost all networks are dynamic. Communities in friendship/acquaintance networks keep evolving as people join new groups or quit old ones. There are always new papers or collaborative work inserted in citation or co-authorship networks. Physical connecting backbones of the Internet keep changing as new routers are added and current ones fail. At any minute, there are new web-pages/information put onto the Internet and outdated ones vanished. In fact, almost all networks keep changing at various rates. The dynamism does not seem to be quite troublesome for analyzing small networks. This is because that when a network is small, it usually only takes a very short period of time for analyzing it. During the analyses process, dynamic changes may have little chance to happen. Also, if the some dynamic change happens, users can take this changed network as a new input and generate a new set of results within very short time. However, dynamism is vital for large network analysis process. As we discussed, analyzing large networks will take a very long time. In some cases, the whole network's structure may have already changed by the time

analysis results are returned from SNA tools. Thus, what we obtained from SNA tools may not be valid anymore. In some cases, after dynamic changes, the structure is altered only in a small part (or parts) of the large network. However, current tools cannot provide the information about which part of results and how they are affected by the dynamic change. In order to obtain a useful analysis of the network, current software tools have to take the changed network as a new input and perform the analysis from scratch. Clearly, this will introduce a formidable overhead especially when the graph size is large. In order to achieve effective and efficient analysis of dynamic large networks, how to effectively adapt the dynamic behavior of networks must be considered in methodology design.

## *2.4 Summary*

Recently, numerous large and complex networks have emerged and been studied in various research fields. Significant insights of large-scale interactions are obtained only by primitively analyzing them on simple SNA metrics. We have confidence to believe that, under large social networks, there is much more important information waiting to be investigated. Analyzing large and complex networks is an important and promising task. However, the poor ability of current SNA software tools prevents further successes of current large social network analysis. In order to achieve a comprehensive and profound understanding of large-scale interactions, it is vital to develop key researches and design appropriate methodologies for analyzing large social networks.

# 3. Methodology Design

The special structure information (introduced in chapter 2) contained in large-scale interactions and the poor ability of current SNA software tools on handling large networks spotlight the need for reconsidering SNA methodology design. New techniques need to be specially developed for analyzing large social network. In order to design an effective and efficient methodology for analyzing large social networks, we should consider and combine approaches from following different fields:

- graph theory
- optimization
- parallel/distributed computation
- algorithm design
- networking/communication

In this chapter, we will first introduce fundamental and significant concerns for designing methodologies for large social network analysis. Then, according to these concerns, we will propose our methodology, an anytime-anywhere methodology based on a parallel/distributed computational framework. Following this, detailed discussions about the architecture of our methodology and design and function of each component in our methodology are presented.

## *3.1 Parallel/Distributed Framework*

Serial algorithms are not suitable for analyzing large networks. Long processing time and large computational resource consumption are apparently two of main challenges which must be addressed if we want to effectively analyze large social networks. Some large social networks, such as friendship/acquaintance

networks based on the LiveJournal database (www.livejournal.com), even cannot be loaded into the memory of a single machine. Many basic SNA metrics, such as all-pair shortest paths, require $O(n^2)$ (or higher order) storage space and processing time. Also, some complicated but useful analyses processes, such as the maximum clique detection, node role assignment, and checking automorphic equivalence [BrandesE'05], are NP-hard or NP-Complete. Obviously, due to the lack of scalability, serial algorithms are typically ineffective towards handling large social networks.

An alternative to serial approaches is parallel/distributed processing. We find that using multiple processors for large social network analysis is an important endeavor. This is because, for one, employing connected computers/processing-units to analyze large social networks will relieve the limit on computational resources. Moreover, parallel/distributed computation can accelerate the analysis process. Thus, we believe that utilizing a parallel/distributed computational framework is a more effective means to provide large social network analysis.

## *3.2 Anytime-Anywhere Properties*

Even using multiple processors, building comprehensive analysis for large social networks inevitably requires large periods of time. "All-or-none" mechanisms will become infeasible for large social network analysis. Here, the term "all-or-none" represents the idea that an algorithm cannot be stopped in the midway to provide useful partial results. Users are either waiting for the complete solution or receiving the complete/final solution for the whole network. By and large, current SNA tools work in this way. "All-or-none" approaches seem to be fine when network size is small. This is due to the reason that the processing time for a small network is very short. However, these approaches have vital pitfalls when dealing with large social networks, especially for time-critical cases. For example, assume that a group of epidemiologists are studying a communication/interaction network of people in a big city, such as Beijing China, to prevent the spreading of

a serious infectious disease, such as SARS. This network may contain tens of millions nodes. Even if we have the ability to handle this large network, analyzing this large graph (predicting the spread of the disease, detecting important persons for effectively and efficiently blocking the spreading of the disease, and helping analysts to making proper decisions) will take a huge amount of time which can be several months. After such a long time, due to the late response, the spread of the disease may be so broad that it cannot be controlled. Furthermore, the network is not static. Individuals may be infected or recovered during this long time. The analyses returned will be too "old" to represent the current status of the disease spread in the city and cannot help relevant health agencies to take proper reactions.

In order to effectively solve these problems, methodologies for large social network analysis should have at least two properties. First, they should be able to be interrupted midway in order to provide useful partial or coarse-level results for quick response. Also, the quality of these partial results can be refined and finally the exact (or a good approximate) analysis results of the whole network can be obtained. Second, they should have the ability to easily incorporate new information in networks during their analysis process.

These two properties are not new to the field of algorithm design and analysis. In fact, these concepts have been studied and have been given the terms anytime and anywhere properties [SantosSW'99]. **Anytime-property** was proposed to balance execution time with solution quality [DeanB'88]. Four characteristics of anytime-algorithms differentiate them from traditional algorithms: quality measurement, predictability, interruptability, and monotonicity. Quality measurement means that partial results' quality can be estimated. Predictability is used to refer the ability that the time cost for obtain partial results at some stage can be estimated or bounded. Interruptability represents that programs can be interrupted and present obtained partial results to users. Monotonicity is used to constrain the quality of partial results. It requires the partial results quality can

be only non-decreasing. By *having an anytime-property, algorithms can provide users partial solutions with the good quality that can be achieved within the given time*. As time evolves, the partial solutions will be refined step by step.

**Anywhere-property** originally was used to refer to the idea of information sharing for problem-solving [SantosSW'99, Santos'01]. It represents that algorithms have the ability to accept complete or partial solutions generated elsewhere and incorporate external solutions into its own processing. It is necessary for a parallel/distributed framework of large social network analysis. As we mentioned, when a social network is large, due to the limit of computational resources, it is typically infeasible for a single processor to handle the whole network. In parallel/distributed computational environments, it is necessary for each processor to handle only a part of the graph. For some SNA metrics, in order to achieve complete analysis results on each processor, the program may need partial/complete results obtained on other processors. In this dissertation, the term "anywhere" is employed to emphasize another idea. That is no matter where and when changes happen, they should be first incorporated in the analysis locally and the new information/solution will be propagated through the whole network as time evolves. In other words, an *anywhere property refers to the ability of algorithms to adapt the new information in the network during the algorithm processing*.

## *3.3 Our Methodology*

Based on the important issues discussed in the previous sections, it is clear that there must be a focus on designing an anytime-anywhere methodology on a parallel/distributed computational framework for large social network analysis. Using a parallel/distributed framework will to enlarge computational resources and accelerate processing process. When the problem to be solved requires large computational work, usually we can decompose it into smaller sub-problems and use a set of processors to solve it in the way that each processor

only handles a single sub-problem. Although based on local sub-problems each processor can only obtain partial solutions, these partial results will provide significant insights for the original problem if it is carefully decomposed. Thus, these results can be used as an initial approximation of the solution for the original large problem. In order to obtain the exact or an accurate enough solution, the partial results need to be refined. For complex problems, this refinement usually takes a long period of time. In order to provide users with various levels analyses (from coarse to fine), the refinement is incrementally achieved stage by stage. By each stage, the obtained partial results can be presented to the user with an estimated quality. Also, during the processing, problem's dynamic information needs to be adapted. When problem's inputs change, we do not recalculate solutions from scratch. In order to effectively and efficiently handle problem's dynamism, the dynamic change adoption is accomplished by refining affected results based on the obtained partial solution.

Thus, in our methodology, we will decompose a large social network into small parts, build a coarse-level of network analysis based on the analysis of separated parts of the network, and incrementally refine these partial results stage by stage. A graph's dynamic changes will be adopted during the analysis process based on the obtained partial results.

**a) Methodology Architecture**

According to working processes, our methodology mainly consists of three phases, Domain Decomposition (DD) phase, Initial Approximation (IA) phase, and Recombination (RC) phase. The architecture of our methodology is shown in Figure 3-1.

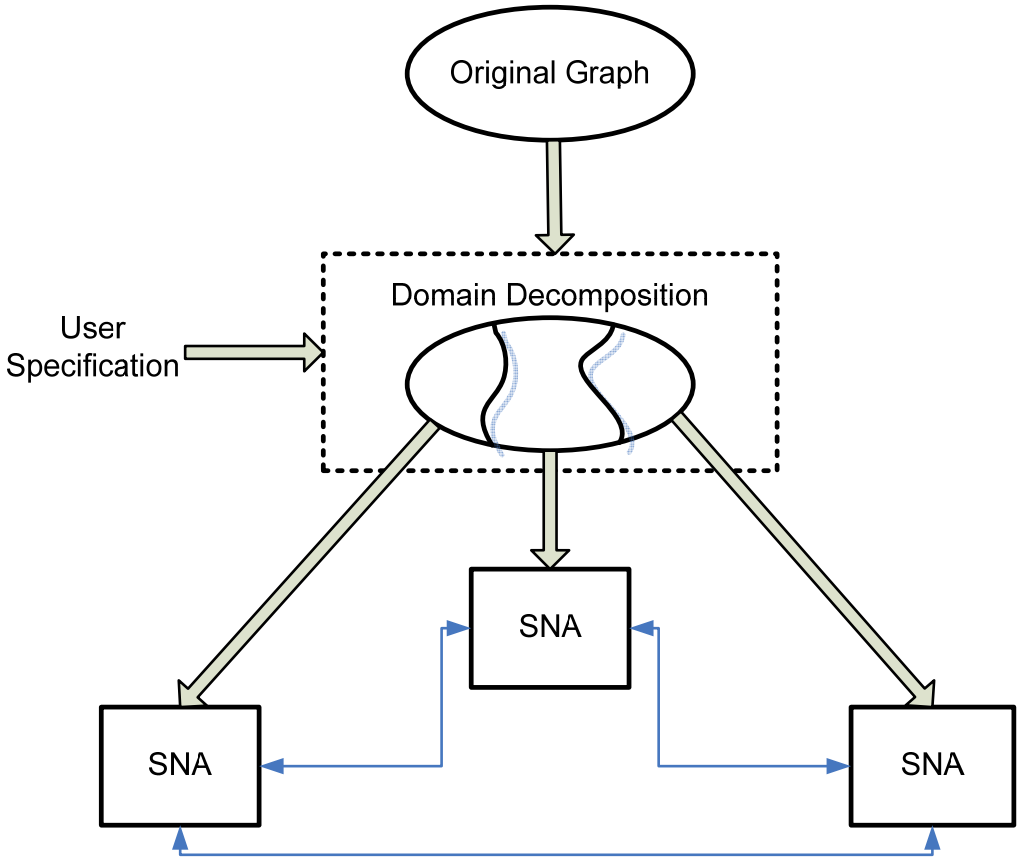



**Figure 3-1**. Our anytime-anywhere methodology's architecture.

Our methodology works as follows. The DD phase is the first phase used in handling large networks. The DD phase takes the charge of breaking a large graph into small-ones. According to general criterion and specific requirements posed by users and SNA applications, the original large network will be cut into several parts in the DD phase within a relative small amount of time. After graph partitioning, sub-graphs are distributed to a set of SNA agents which can be one or a group of processors. At each agent, current SNA technologies or specially designed approaches are applied and the analyses of the sub-graphs are generated. We take these analyses as an initial approximation of the original network. Thus, this phase is called Initial Approximation phase. The function of RC phase is to incrementally build the exact (or a good approximate) solution of the whole network. In this phase, each agent may iteratively communicates with each other, refine local solutions with the results obtained on its own or at other

agents, incorporates graph's dynamic information during processing. The anytime-anywhere property of our methodology is mainly embodied in functions of this phase.

One of our main goals is to build a framework for large social network analysis which can be used in a broad range of applications. Thus, our system is designed based on modular architecture since it provides good flexibility. In our system, there is one module corresponding to a single phase and each module is a plug-and-play component. Algorithms and mechanisms employed in each component may change. However, by the plug-and-play design, the framework of the system does not change. In the following paragraphs, we will provide details of each component design.

**b) Domain Decomposition Phase**

As we mentioned, it is not feasible for each processor to handle the whole network. We need to partition the large graph into small parts. The Domain Decomposition phase takes the charge of partitioning a large graph into computationally tractable intra-related balanced sub-graphs.

The Domain Decomposition phase is important for our methodology. From the methodology architecture, which is shown in Figure 3-1, we can see that how well a graph is decomposed will affect the quality of the initial approximation of the whole graph and the work load remaining in the recombination phase.

Arbitrary/random graph partition is not suitable for the DD phase. The Domain Decomposition phase has its own requirements on graph partitioning. First, sizes of sub-graphs generated by the DD phase should be small enough to meet the limits of SNA agents/processors. Second, all these sub-graphs should have balanced sizes. Dividing a big problem into balanced small parts will help us to improve the system's efficiency. Third, generated sub-graphs should be "isolated" from each other. Since we want to use the results from each sub-graph

to initially approximate the solution of the whole graph, the more isolated sub-graphs, the more accurate approximation we will get and the less work remained in the RC phase.

During the design of the domain decomposition, it is vital to consider the balance of the work in dividing and combination. Putting all the work on dividing, such as quick sort algorithm [BaaseG'00], or leaving all the work to combining, such as merge sort algorithm [BaaseG'00] are both unsuitable for our methodology. Putting a lot of efforts on dividing, we can obtain initial approximation with very good quality. However, this will break the ability for making quick response. Putting main efforts on combination, we can get the decomposition done within a very shot time. Nevertheless, the obtained quick analysis should be useful. We cannot afford to miss a lot of important information in the graph in the initial approximation. Thus, it is necessary to design a specific algorithm for decomposing a large graph into intra-related small parts within a relatively short period of time.

Obviously, in the DD phase, we can check the connectivity of a graph and put each connected component onto a single processor. Checking graph's connectivity does not cost long time. However, for a large network, its biggest connected component may be also large. Usually, for a large graph, we have to partition it with discarding some connections in the graph.

In order to improve the quality of initial approximation of a network and balance the work between dividing and combining, there are many metrics that may be considered for guiding the process of graph decomposition. The number of cut-edges is one of the most fundamental factors for graph partitioning. Cut-edges are defined as those edges whose endpoints belong to different sub-graphs. The sum of cut-edges' weights is defined as graph cut-size. When we partition a graph, we will remove cut-edges and convert the graph into several disconnected components/sub-graphs. The more edges cut, the more information we will lose.

In order to alleviate the work in combination and achieve more accurate initial approximation, cut-size should be optimized. Actor degree is another significant factor we may need to consider during graph partitioning. For many cases, nodes with high degrees are centers of communities. It is not good to place nodes with high degree on or close to the boundary of sub-graphs. Here, a sub-graph's boundary is defined as the set of nodes with which cut-edges are incident. In some cases, we should also consider the similarity between actors during partitioning a graph. Putting similar actors into the same part will provide non-trivial insights for social network analysis. According to various types of social networks and applications, there are many other factors which may be considered in the domain decomposition, such as the importance of each node, the connectivity property of each node and the sub-graph we generated, etc. When to use them and how to use them will be determined by the specific social network and its application.

Essentially, the DD phase can be treated as a multi-objective optimization process. The objective function will be determined by general criteria for graph decomposition and specific concerns required by applications or users. The task of the DD phase is to find an optimized graph partition within a relatively small time.

**c) Initial Approximation Phase**

After the DD phase, a large network is partitioned into small sub-networks. These small graphs can be easily analyzed by current SNA techniques or specially designed approaches. Comparing with analyzing the whole large graph, analysis results for sub-graphs can be obtained within a much smaller period of time. These results can be used as a preliminary approximation of the original network. These initial approximation results can help SNA users to establish the fundamental feeling, recognize the basic structure, and identify primary important components and properties of the network. In the IA phase, tools employed at each SNA agent can be current commercial SNA software packages or some

specially designed algorithms.

**d) Recombination Phase**

The function of the RC phase is to adopt graph's dynamic change and incrementally refine partial results over time, based on results obtained locally and/or externally, so as to achieve the exact (or a good approximate) analysis of the original network. The anytime-anywhere property of our methodology is mainly demonstrated in this phase.

From the methodology architecture (Figure 3-1) we can see that SNA agents/ processors are connected to communicate with each other through physical networks. The analyses of the local sub-graph are generated at each SNA agent in IA phase. Then, each agent refines its local solutions based on local information and solutions obtained elsewhere (if needed), and propagates its new solutions through the whole network stage by stage.

Since networks are dynamic, there will be changes happening in large networks during analyses processes. In the RC phase, no matter where and when these changes happen, they will be adapted locally by each SNA agent. Then, if needed, the relevant information will be transmitted to other agents and the effects of these changes will be incrementally propagated through the whole network.

For both anytime and anywhere approaches in the RC phase, there are two significant concerns which should be considered. First, we should have the ability to predict or check the convergence of algorithms. In other words, we should know when the obtained results are exact or accurate enough. Second, it is necessary for us to measure or predict for users the accuracy of partial solutions obtained in the RC phase.

## *3.4 Focuses of Our Methodology*

Currently, in the field of SNA, social networks are analyzed based on numerous types of SNA metrics which have different requirements and properties. Unsurprisingly, such variations imply diverse SNA methodological constructs. In our anytime-anywhere methodology design presented in this dissertation, we focus our attention on a broad group of on SNA metrics that have certain structure or criteria.

In particular, we consider, SNA metrics which can be recursively defined. In other words, the metric value on the next stage can be calculated from partial results obtained at current stage and/or previous stages. This requirement is easy to understand. If a metric cannot be recursively defined, then after obtaining partial results on each stage, we need to do the recalculation from scratch for the next stage. Comparing with methods which directly calculate the exact results, this will introduce a great amount of overhead.

In order to effectively and efficiently handle a graph's dynamic information during analysis process, we determine elements in partial or accurate results which are affected (or non-affected) by the dynamic change. Without this ability, we may have to work on all elements in the obtained results. It is similar as recalculating whole results from scratch. If we can identify the affected elements, we can focus our efforts on making proper changes for these affected elements. Also, in order to achieve better efficiency, the effects of dynamic changes should be able to be calculated either based on obtained results or from only a portion of original problem inputs.

## *3.5 Summary*

In this chapter, we proposed an anytime-anywhere methodology for effectively and efficiently analyzing large and dynamic social networks. According to working processes, our methodology can be decomposed into three main phases: Domain Decomposition, Initial Approximation, and Recombination. The anytime-anywhere property of our methodology is mainly implemented and expressed in functional design of the Recombination phase. Our methodology is designed based on a modular architecture. Each phase can be taken as a plug-and-play component. The specific implementation or employed algorithms of each phase can be modified according to different requirements of various SNA metrics and real applications. The modularity design endues our methodology with great flexibility.

# 4. Methodology Analyses and Validation

In Chapter 3, we present details about the design of our anytime-anywhere methodology for analyzing large and dynamic social networks. The proposed methodology is designed on a modular architecture and it can be applied on a broad range of social network metrics and social network analysis techniques. Based on common knowledge of system design, intuitively we believe that our methodology can provide significant advantages for large social network analysis, such as accelerating the analyses process, providing various levels of analysis results, effectively handling graph's dynamism, etc. In order to evaluate and validate our methodology, we decide to study our methodology's performance, both theoretically and experimentally, on a set of SNA problems which cover a broad range of difficulties. According to application importance and computational costs, we decide to choose the following three SNA metrics:

- ego-betweenness centrality,
- closeness centrality,
- and maximal clique enumeration.

In this chapter, we will first introduce a number of basic and fundamental terminologies frequently used in SNA, which are of particular interest in our discussion. Then, we will provide definitions, and corresponding popular/common algorithms employed in current SNA tools and their computational complexities respectively.

## *4.1 Fundamental Definitions and Terminologies*

As we mentioned before, social networks essentially are graphs. A *graph* can be presented as *G(V,E)* where *V* is the set of elements called vertices/nodes/actors

and *E* is the set of unordered pairs of nodes called edges/links/ties. |*V*| and |*E*| are the cardinality of set *V* and *E* respectively and usually are denoted as *n* and *m*. |*V*| is also called as *graph size*.

We say a vertex *u* is *adjacent* to (or has a direct neighbor of) *v* if {*u,v*} is an edge included in the set *E*. This edge is denoted as *e*(*u,v*). We call vertices *u* and *v* as endpoints of *e*(*u,v*), and we say that the edge *e*(*u,v*) is incident with vertices *u* and *v*. Edges in networks can have directions. We use $\vec{e}(u,v)$ to represent the directed edge connecting *u* and *v*, and pointing from *u* to *v*. If edges have directions, a graph is called *directed graph* or *digraph*. Otherwise, it is called *undirected graph*. All edges in an undirected graph are symmetric. More precisely, this can be formed as

$$e(u,v) \Leftrightarrow e(v,u).$$

Also, an undirected graph can be treated as a directed graph by adding both directions onto each edge. This is mathematically represented as

$$e(u,v) \Leftrightarrow \vec{e}(u,v) \,\&\, \vec{e}(v,u).$$

*Degree* of a vertex *v* is defined as the number of edges incident with *v*. Usually, we use *deg*(*v*) to denote the degree of the vertex *v*. The *maximum degree* of graph *G* is the largest degree over all vertices. Usually, we use *Δ*(*G*) or simply by *Δ*, if no ambiguity exists. In a directed graph, nodes may have two types of degrees, in-degree and out-degree. *In-degree*, which is represented as $deg(v^+)$, is the number of edges pointing to node *v*. *Out-degree*, denoted as $deg(v^-)$, is the number of edges leaving from the node *v*.

Each *e*(*u,v*) can be assigned some value *w*(*u,v*), which are variously referred to as *weight*, *cost*, or *length*. If all edges in a graph have uniform weights, this graph is called an *unweighted graph*. Otherwise, it is called a *weighted graph* and is represented as *G*(*V*, *E*, *W*) where *W* is the set of edge weights. A *path* connecting nodes *u* and *v* is defined as an alternating sequence of vertices and edges,

$$e_1(v_1, v_2), e_2(v_2, v_3), \cdots, e_{k-1}(v_{k-1}, v_k),$$

where $v_1=u$ and $v_k=v$. All the vertices and edges in the sequence are distinct (exception $v_1$ and $v_k$). In this document, we use $p_{uv}$ to represent a path connecting u and v. *Path length/cost/distance* is the sum of all the weights of the edges belonging to this path and is represented as $d_p(u,v)$. The *geodesic path* between two nodes *u* and *v* is the path with the shortest distance, and this distance is called *geodesic distance* and denoted as $d(u,v)$.

A graph *G`*(*V`,E`*) is called a sub-graph of *G*(*V,E*) if its vertex set *V`* and edge set *E`* are subsets of *V* and *E* respectively. This sub-graph is called an *induced sub-graph* of *G* if for every pair of vertex *u* and *v* of *G`*, *e(u,v)* exists in *G`* if and only if there is an edge *e*(*u,v*) in *G*. In a sub-graph, a *boundary node* is defined as a node which has connections with nodes belonging to other sub-graphs. The *boundary size* of a sub-graph $G_i$ is defined as the number of its boundary nodes contained, and is denoted as $|B_i|$. An edge is called as *cut-edge* if its endpoints belong to two different sub-graphs. The set of cut-edges of sub-graph $G_i$ is denoted as $C_i$. *Cut-size* is defined as the number of cut-edges and represented by $|C_i|$.

In this dissertation, our study is mainly focused on weighted digraphs with real (positive or negative) edge weights.

## *4.2 Centrality Measurements in SNA*

Centrality is one of the most important and frequently used measurements in SNA [CarringtonSW'05]. It is a descriptive characteristic for actors or groups of actors with various structural properties and a crucial parameter for understanding and analyzing actor roles in social networks [Newman'05]. Usually, centrality is used to identify powerful, influential or critical actors.

Centrality has diverse definitions because of different understandings of social power and various SNA applications [CarringtonSW'05, HannemanR'05]. The most widely accepted definitions of centrality are proposed in by Freeman in [Freeman'79] in the late 1970s. In these definitions, centrality measurement is measured mainly based on three aspects, degree, closeness, and betweenness.

**a) Degree Centrality**

Degree centrality is defined as the number of ties which are incident with a given node. This measurement usually reflects the popularity and relational activity of an actor [Marsden'02, Frank'02, Newman'05]. For example, in a friendship network, we can find the most popular persons by identifying the actors which have the largest degree centrality. If we have a graph with *n* vertices, degree centrality is mathematically defined as formula 4.1.

$$C_D(v_i) = \sum_{k=1}^{n} a(v_i, v_k) \qquad (4.1)$$

where $a(u,v)$=0 if *u* and *v* are not connected by an edge, otherwise, $a(u,v)$=1.

**b) Closeness Centrality**

Closeness centrality measurement is based on geodesic distances. It measures how far away a node is from all other nodes. It indicates the actor's availability, safety, and security [Frank'02]. More precisely, closeness centrality is defined as

$$C_C(v_i)^{-1} = \sum_{k=1}^{n} d(v_i, v_k) \qquad (4.2)$$

Many social researchers argue that for large networks, closeness centrality measurement defined in formula (4.2) is not attracting. This is because in a large social network, usually an actor is only close to a limited set of other actors. The closeness centrality measurement for most actors in large social networks will be very small. Typically not many insights are contained in the closeness centrality measurement. The reason for this problem is that summing all the geodesic distances will lose a lot of information. However, the distribution of geodesic distances from a source node to all other nodes contains non-trivial information. For example, when we analyze spreading of diseases, we need to use these

distances to estimate the propagation of the disease in a network. Thus, in large social network analysis, the closeness centrality for node $v_i$ is represented by two kinds of parameters. One is the closeness value which is defined in formula (4.2). The other one is a distance vector which stores the geodesic distances from this node $v_i$ to all other nodes.

**c) Betweenness Centrality**

Betweenness centrality of a vertex *v* is defined to be the fraction of shortest paths that go through *v*. This measurement represents the actor's capability to influence or control interaction between actors it links [Marsden'02, Frank'02, Newman'05]. Mathematically, it is defined as the following formula:

$$C_B(v_i) = \sum_{j=1}^{n} \sum_{k=1}^{j-1} g_{jk}(v_i) / g_{jk} \qquad (4.3)$$

where $g_{jk}$ is the total number of geodesic paths (shortest paths) linking $v_j$ and $v_k$, and $g_{jk}(v_i)$ is the number of geodesic paths that pass through $v_i$.

From the definition, we can see that in order to measure betweenness centrality for all nodes, we have to find and store all geodesic distances for all pairs of actors. There may be multiple shortest paths between a pair of nodes. Algorithms for betweenness centrality are quite complicated and require a great amount of memories, which can be $O(n^3)$. Calculating betweenness centrality for each node in a large network seems to be impractical due to this giant storage space requirement and expensive computational cost. Therefore, many SNA researchers try to employ other metrics to approximate and substitute betweenness centrality. Ego-betweenness centrality has been verified to have high correlation with the original betweenness centrality and can be used as a good approximation for it [Marsden'02, Newman'05].

**d) Ego-Betweenness Centrality**

Ego-betweenness centrality is defined based on ego-centric networks or simply ego-networks, which are also called first-order neighborhood networks. This kind

of networks consists of a single vertex, called *ego*, together with its direct neighbors, denoted as *alters*, and all the interactions between the ego and alters and among alters [Marsden'02]. In other words, an ego-centric network is an induced sub-graph of the original network on a set of vertices which consists of an ego and its direct neighbors. We use the following example to explain the definition of ego-centric networks. Assume we have a network as shown in Figure 4.1.

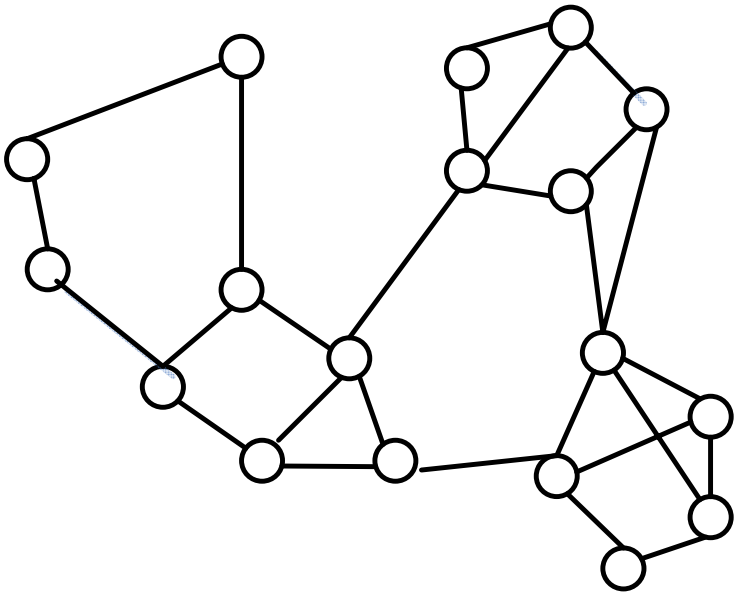


**Figure 4-1.** An example social network.

We can pickup any node as an ego. Assume that we randomly pickup a node *v* and set its color as yellow. Then, we can find all its direct neighbors and set their color as blue. This is shown Figure 4-2. In this figure, the yellow node is the ego, and blue nodes are ego's alters

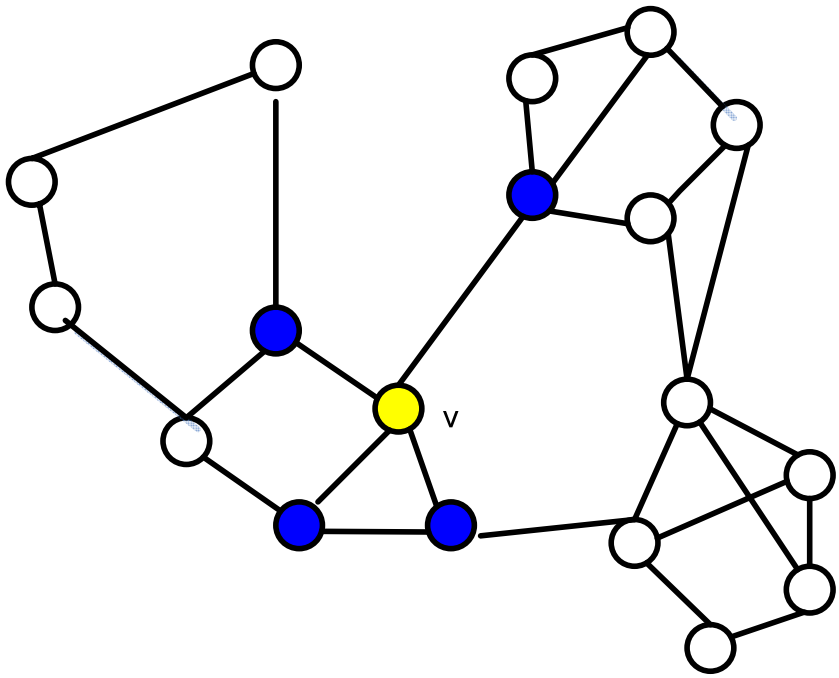



**Figure 4-2.** The selected ego *v* and its alters.

The ego-centric network is the sub-graph induced on the ego and its alters, which is shown in Figure 4-3. This ego-centric network is also called as the ego-network of node *v*.

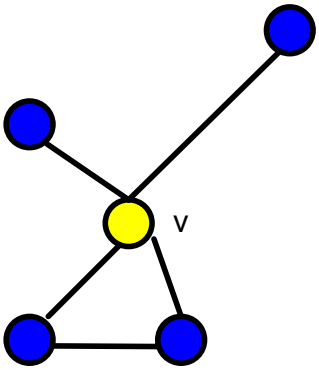



**Figure 4-3.** The ego-network of node *v*.

In the definition discussed before, we can see that the ego-networks we discussed only focus on the first-order zone of the ego. In other words, all the vertices in an ego-centric network are within distance 1 to the ego. More generally, an ego-network can be constructed with alters which lie within a given distance *K* to the ego and links among them. This type of ego-networks are

called $K^{th}$ order/step neighborhood network [HannemanR'05]. In current SNA applications and researches, first-order neighborhood networks are employed the most frequently. Thus, in this document, we focus our study on this type of ego-networks.

Ego-betweenness centrality, as its name shows, is focused on ego-networks. The ego-betweenness centrality for a vertex *v* is measured similar as formula (4.3), except that instead of using the whole network, ego-betweenness centrality is measured on the ego-network of *v*. Researchers' experimental results have shown that ego-betweenness is highly correlated with and could be a reliable substitute for the Freeman's betweenness measurement [Marsden'02, Newman'05].

## *4.3 Maximal Cliques*

A *clique* is a completely connected graph. In other words, a clique is a set of vertices within which there is an edge between any pair of vertices. Each vertex contained in a clique is called as *clique member*. In this chapter, we use clique member set, *S*, to represent a clique. Usually, cliques are defined on undirected graphs. We do not consider edge directions during processing cliques. From clique's definition, we can also see that during handling cliques, we do not care about weights of edges. Instead, we only concern if all vertices in a clique are directly connected. Thus, when studying cliques, we only consider dichotomized undirected graphs. For weighted graphs, usually we can transform it to a dichotomized graph by setting a threshold. If an edge weight is larger than the threshold, we set the new edge weight as1. Otherwise, we set it as 0.

A clique *S* is contained in a clique *S'* if and only if:

$$S \subset S'$$

A *maximal clique* is defined as a clique that cannot be contained in other cliques. *Clique size* is defined as the number of vertices contained in the clique, which is

represented as |*S*|. The maximal clique with the largest size is called as *maximum clique*, and is represented as *S**.

Essentially, cliques are special type of structure and implicate significant insights in a social network. Due to inside highly intensive interactions, cliques usually are the most fundamental and key elements in communities (cohesive groups of actors) in a social network. A c*ommunity* is defined as a set of actors which has more interactions within the set and has less interactions inter sets. Community is one of the most important structural information contained in social networks. Identifying and analyzing communities in a social network is critical for studying how organizations are formed, how organizations interact with each other, how actors operate differently within an organization and inter organizations, etc.

## *4.4 Algorithms for Measuring Closeness Centralities*

In order to analyze how the metrics introduced above are measured by computer techniques, in the following sections, we will discuss existing popular algorithms which are commonly employed in current SNA software tools or SNA researchers. In this section, we will focus on algorithms for measuring closeness centralities. Closeness centrality measurement is based on calculating all-to-all geodesic distances. There are many algorithms developed for this problem. In this document, we focus on two of the most widely used algorithms: Dijkstra's algorithm [Dijkstra'59] and Floyd's algorithm [Floyd'62].

### a) Dijkstra's algorithm

Dijkstra's algorithm [Dijkstra'59] is one of the most popular graph algorithms. This algorithm is a type of greedy approach. It has been proved that Dijkstra's algorithm can find the shortest paths (and their geodesic distances) between a source vertex and all other vertices for a graph with positive real edge weights. The proof can be found in [BaaseG'00]. The pseudo-code of Dijkstra's algorithm

is shown in Figure 4-4. In this figure, s represents the source vertex. This algorithm calculates the shortest distances between the source vertex to all other vertices in the graph.

```
1. initialization:
    a): for each node v: set d_v as infinity;
    b): set all nodes status as unseen;
    c): initialize a set of nodes, Q, as {s};
    d): set d_s as 0;
2. while Q is not empty
3.      find the node v in Q which has the smallest distance to s, remove it
        from Q;
4.      for each node u which is not finished and adjacent to v
5.          get D = d_v+w(v,u);
6.          if u's status is seen
7.              if D<d_u
8.                  d_u =D ;
9.                  update u's information in Q;
10.         if u's status is unseen
11.             d_u =D, set u's status as seen ;
12.             put node u into Q;
13.     set v's status as finished
```

**Figure 4-4.** Pseudo-code of Dijkstra's algorithm.

From the pseudo-code we can see that in each loop, most work done in Dijkstra's algorithm is finding the element with smallest distance to the source vertex in the queue *Q*. To our best knowledge, using Fibonacci heaps, Dijkstra's algorithm can obtain the optimum computational time as $O(m+n\log n)$ [BaaseG'00], where *n* is the graph size and *m* is the number of edges contained in the graph. Running Dijkstra's algorithm, we can solve the single-source shortest paths problem. In order to obtain the closeness centrality measurements for all vertices we need to run Dijkstra's algorithm on every vertex in the graph.

Thus, the cost for using Dijkstra's algorithm to measure closeness centralities for all vertices is $O(nm+n^2 \log n)$.

Dijkstra's algorithm is a decent algorithm for closeness centrality measurement. However, it is not suitable for handling large social networks. As we mentioned, when a social network size is large, it is infeasible to put the whole graph on a single processor. The graph needs to be decomposed into smaller parts and each processor will only focus on one part of the graph. However, in order to use Dijkstra's algorithm, it requires the processor to have information of the whole network. Moreover, Dijkstra's algorithm only works for graphs with positive real edge weights. In some applications, edge weights in social networks can be negative. Thus, in order to be able to handle a general graph, we decide to employ other algorithms in our methodology for closeness centrality measurement.

**b) Floyd's Algorithm**

Floyd's algorithm [McHugh'90] is a fundamental and popular algorithm for solving the all-pairs shortest paths problem. Using Floyd's algorithm, we can find geodesic paths (and geodesic distances) between all pairs of vertices. Floyd's algorithm is an iterative method. It tries to incrementally update the distance matrix $D$ by each vertex's connection information. Here, $D$ is an $n$ by $n$ matrix and each element $D_{ij}$ stores the obtained shortest distance for paths connecting from $i$ to $j$. The recurrence of distance matrix updating is formed in [McHugh'90] as:

$$D^0_{i,j} = w_{i,j}$$
$$D^k_{i,j} = \min(D^{k-1}_{i,j}, D^{k-1}_{i,k} + D^{k-1}_{k,j})$$

where $w_{i,j}$ is the weight of the edge connecting from vertex $i$ to vertex $j$. If there is no edge connecting these two vertices, the weight is set as positive infinity. Define *internal vertices* for a path as vertices on the path except the source and the target. By induction, it is not hard to see that at stage $k$, the obtained paths have the shortest distances among all paths whose internal vertices are chosen

from 1, 2 … $k$. When $k$=|$V$|, we can obtain the shortest paths between all pairs of vertices. The pseudo-code of Floyd's algorithm is shown in Figure 4-5:

```
1. initialization:
       copy adjacent matrix A into D
2. for(k=0; k<n; k++)
3.       for(i=0; i<n; i++)
4.             for(j=0; j<n; j++)
5.                   D_{i,j} = min(D_{i,j}, D_{i,k}+D_{k,j})
```

**Figure 4-5.** Pseudo-code of Floyd's algorithm.

By running Floyd's algorithm, we can obtain the shortest paths for all-pairs of vertices for a broader range of graphs which can have negative real edge weights but without negative cycles. The proof of the correctness of Floyd's algorithm can be found in [McHugh'90]. The computational cost of this algorithm is the same as matrix multiplication, which is $O(n^3)$.

Comparing with Dijkstra's algorithm, we can see that Floyd's algorithm is slower which makes Dijkstra's algorithm seem to be better. However, Floyd's algorithm has its own significant advantages. First, Floyd's algorithm can work on graphs with negative edge weights. Moreover, Floyd's algorithm is more suitable for a parallel/distributed computational framework. For example, Distance Vector Routing (DVR) algorithm [KuroseR'01] is a modified version of Floyd's algorithm on a distributed framework. This algorithm is one of the most frequently used algorithms in the network routing problem which is similar as the one-too-all shortest path problem if we take each router as a vertex in a graph. On each router, DVR algorithm only focuses on the connections of local router to its neighbor routers. Each router iteratively tries to update its shortest distances to all other routers based on the distance information of its neighbors. DVR algorithm also can effectively handle the graph dynamism. When a change of network connection happens, it will be adopted by the routers which are incident

on this edge. Then, the effects of the dynamic change will propagate through the whole network as a ripple-effect. Thus, we can see that DVR algorithm seems to be more suitable for our methodology. However, DVR algorithm cannot be directly applied in our anytime-anywhere methodology. This is because, in the DVR algorithm, each processor only contains a single vertex. As we mentioned that large social networks have a great amount of vertices, it is impractical for us to employ so many processors in our parallel/distributed computational framework. Moreover, DVR algorithm is an asynchronous algorithm which means that it will be very hard to tell when this algorithm converges and estimate the quality of partial results generated by DVR algorithm. In order to address these problems, in our anytime-anywhere methodology, we implement a modified DVR algorithm for measuring closeness centralities. In later chapters, we will provide details about the design, implementation, performance analysis, and anytime property for our modified DVR algorithm.

## *4.5 Algorithms for Measuring Ego-Betweenness Centralities*

In this section, we will briefly introduce two typical algorithms which are commonly used in current SNA researches for measuring ego-betweenness centralities. They are Everett's algorithm [EverettB'05] and a modified Dijkstra's algorithm. In following paragraphs, we will discuss the workflow, the computational cost, and pros & cons for each algorithm.

### a) Everett's Algorithm: A Straightforward Approach

A simple and fast algorithm for measuring the ego-betweenness is proposed in [EverettB'05]. This algorithm is based on manipulation of the network adjacent matrix. In an ego-network generated from an unweighted graph, the geodesic distance between any pair of vertices is either 1 or 2. Adjacent alters do not contribute to the betweenness of the ego. The ego-betweenness is determined by the paths of length 2 for non-adjacent pairs of alters. This information can be

obtained in the square of the adjacent matrix *A*. To avoid counting the adjacent alters, we can generate a new matrix *B* where

$$B_{i,j} = A^2_{i,j} \cdot (1 - A_{i,j}),$$

where $A_{i,j}$ is the element on the *ith* row and *jth* column of *A*. $A^2$ is the square of the adjacent matrix. The ego-betweenness is the sum of the reciprocal of all non zero elements in the matrix *B*.

From the algorithm design we can see that most of the work is done in matrix multiplication (calculating the square of adjacent matrix). The dimension of adjacent matrix is *deg*(*ego*) by *deg*(*ego*). Based on the common knowledge of matrix multiplication, the computational time of Everett's algorithm is the $O(deg^3(ego))$. For the whole large network, the algorithm's work load is bounded by $O(n\Delta^3)$ where $\Delta$ is the maximum degree, and *n* is the graph size.

This algorithm is easy to understand and simple to implement. It was employed in the initial implementation of our methodology [SantosPAP'06]. However, this algorithm is designed for unweighted graphs. It has vital problems on handling weighted graphs. When edges have different weights, elements in the square of the adjacent matrix do not represent the number of paths with 2 hops anymore. Also, the distance of the path with 2 hops may be equal to any real positive value, instead of 2. Moreover, the shortest path connecting two actors may contain more than 2 hops. Thus, Everett's algorithm is not employed in our methodology in this dissertation.

**b) Modified Dijkstra's Algorithm**

Another popular algorithm for measuring ego-betweenness centrality is a modified version of Dijkstra's algorithm [BrandesE'05]. We have already introduced the original Dijkstra's algorithm in the previous section. We know that using Dijkstra's algorithm we can obtain the shortest paths (and their distances) which connect from the source vertex to all other vertices in a graph with real positive edge weights. There may be multiple shortest paths for a pair of vertices.

According to the definition of ego-betweenness, we can see that in order to calculate ego-betweenness centrality we need the information about all shortest paths in the ego-network. We can employ Dijkstra's algorithm to achieve these information by modifying it in the ways as shown in [BrandesE'05]. The modification of Dijkstra's algorithm is the addition of a mechanism to record the number of all the shortest paths between each pair of vertices ($u$,$v$), and the number of those shortest paths which connect ($u$,$v$) and go through the ego, $o$. The pseudo-code of the modified Dijkstra's algorithm is shown in Figure 4-6. In this algorithm, we maintains three elements for each vertex $v$:

1. The distance $d_v$ to the source vertex $s$.
2. The number of shortest paths connecting $s$ to $v$. We use $g_v$ to represent this number.
3. The number of shortest paths connecting $s$ to $v$ and going through the ego $o$. We denote it as $g_v(o)$.

1. initialization:
    a): for each node *v*: set $d_v$ as infinity and set $g_v$ & $g_v(o)$ as 0;
    b): set all nodes status as *unseen*;
    c): initialize Q, a set of nodes, as {*s*};
    d): set $d_s$ as 0, set $g_s$ as 1;
2. while *Q* is not empty
3. find the node *v* in *Q* which has the smallest distance to *s*, remove it from *Q*;
4. if *v*=*o*, set $g_v(o)=g_v$;
5. for each node *u* which is *not finished* and adjacent to *v*
6. $D = d_v+w(v,u)$;
7. if *u*'s status is *seen*
8. if $D<d_u$
9. $g_u=g_v$; $g_u(o)=g_v(o)$; $d_u =D$
10. update *u*'s information in Q
11. else if $D=d_u$
12. $g_u=g_u+g_v$; $g_u(o)=g_u(o)+g_v(o)$
13. update *u*'s information in Q
14. if *u*'s status is *unseen*
15. $g_u=g_v$; $g_u(o)=g_v(o)$; $d_u =D$; set *u*'s status as *seen*
16. put node *u* into Q
17. set *v*'s status as finished

**Figure 4-6.** Pseudo-code of modified Dijkstra's algorithm for ego-betweenness measurement

Running our modified Dijkstra's algorithm for a specific source vertex *s*, we can obtain two vectors

$$\mu=(g_1, g_2, \ldots, g_k),$$

and

$$\mu'=(g_1(o), g_2(o), \ldots, g_k(o)),$$

where *k* is size of the ego-network. Each element $\mu_i$ is the number of shortest paths connecting source vertex *s* and vertex *i*. Each element $\mu'_i$ represents the

number of shortest paths connecting source vertex *s* and vertex *i* and go through the ego vertex *o*. The contribution of vertex *s* to the ego's ego-betweenness value is:

$$e_s = \sum_{i=1}^{k} \frac{\mu_i'}{\mu_i'}.$$

According to formula (4.3), taking every vertex as a source and add up all their contributions, we can obtain the exact ego-betweenness value. This is formulated as

$$C_B(v_i) = \frac{1}{2}\sum_{i=1}^{k} e_i$$

An ego-network's size is the value of the degree of the ego, *deg*(*o*). The modified Dijkstra's algorithm has the same computational cost as the original Dijkstra's algorithm. Assume an ego-network *Z* has $|E_z|$ edges. From the previous section, we know that the work load for the modified Dijkstra's algorithm is bounded by $O(|E_z|+deg(o)\log(deg(o)))$. In order to calculate ego-betweenness centrality value, we need to run the modified Dijkstra's algorithm on all vertices in the ego-network. Thus, the work load for calculate ego-betweenness for vertex *o* is $O(deg(o)|E_z|+deg^2(o)\log(deg(o)))$. For measuring ego-betweenness centralities for all vertices in a network with size *n*, the work load is bounded by $O(n\Delta|E^*_z|+n\Delta^2\log\Delta)$ where $\Delta$ is the maximum degree, $E^*_z$ is the maximum edge set among all ego-networks in the network. Since the modified Dijkstra's algorithm can effectively handle weighted graph and has decent computational cost, we implemented this algorithm in our methodology for measuring ego-betweenness centralities for large and dynamic social networks.

## *4.6 Algorithms for Maximal Clique Enumeration*

One of the most significant tasks for SNA is finding cohesive groups of actors contained in a social network. Usually, cliques are fundamental elements for forming and identifying these groups. Finding all cliques also has significant

applications in many other fields, such as biology, electronic circuit design, etc. There are numerous algorithms developed for the problem of maximal clique enumeration. Due the space limitation, in this section, we will introduce the most frequently referred algorithm and the newly proposed algorithm.

**a) BK Algorithm**

One of the most fundamental and frequently referred algorithm for finding all maximal cliques contained in a graph is BK algorithm [BronK'73] which is published in 1973. This algorithm employs a recursive branching strategy to traverse all cliques on a search tree based on three dynamically changing sets:

- *compsub*: a global set containing the clique which is being processed;
- *candidates*: a local set consisting of all vertices which will eventually be added to current *compsub*;
- *not*: a local set holding vertices that have already been added to current *compsub*. In other words, any extension of *compsub* containing any vertex in *not* has already been generated.

The essential idea of BK algorithm is recursively extending *compsub* based on *candidates* so as to generate its all extensions which do not contain any vertex in *not*. The basic mechanism of BK algorithm can be found in [BronK'73]. Based on the specific strategies for selecting different types of elements in *candidates* to expand *compsub*, there are several improved versions [BronK'73, Jonston'76] of the BK algorithm. The worst case for BK algorithm has been proven to be $O(3^{n/3})$ [TomitaTT'04].

Finding all maximal cliques is a very hard task. In fact, maximal clique enumeration is an NP-harp problem. In the worst case, the computational cost is in an exponential order of the graph size. For some extreme graphs, even when their sizes are not so large, it still takes formidable long time to find all maximal cliques. Thus, we can see that for this type of problems a methodology with anytime-anywhere property becomes more desirable. BK algorithm can be interrupted during processing and generate some partial results. However, BK

algorithm tends to traverse all maximal cliques with a pseudo-random order. It is hard to estimate the quality of the obtained partial results. It seriously degrades the usefulness of the partial results. Thus, we say that BK algorithm has a poor anytime property.

**b) Zhang's Algorithm**

Recently, there is a new type of approaches [ZhangABCLS'05, KoseWLF'01] developed for maximal clique enumeration. This type of approaches is designed based on the fact that every clique with size *k* (or *k*-clique) is generated from cliques with size *k-1* (or (*k-1*)-cliques). One of the most representative and efficient algorithm based on this type of approaches is Zhang's algorithm [ZhangABCLS'05]. In the following paragraphs, we will briefly introduce Zhang's algorithm.

In Zhang's algorithm, cliques are generated in an increasing order of the clique size. Define a clique with size *k* which can be expanded to be a *candidate k-clique*. A candidate k-clique contains two parts, a *k*-clique and its common neighbors. Taking the nodes which are connected by an edge as cliques with size 2, the algorithm first identifies the set of all maximal cliques with size 2. Then, it puts expandable cliques with size 2 into a list of candidate 2-cliques. Next, it tries to expand each candidate 2-clique to generate cliques with size 3. Repeat this process on *k*-cliques until there are no candidate cliques. In order to avoiding exploring the same clique multiple times, Zhang's algorithm keeps all *k*-cliques and candidate *k*-cliques in non-repeating canonical order. Taking the graph shown in Figure 4-7 as an example, let us show how Zhang's works.

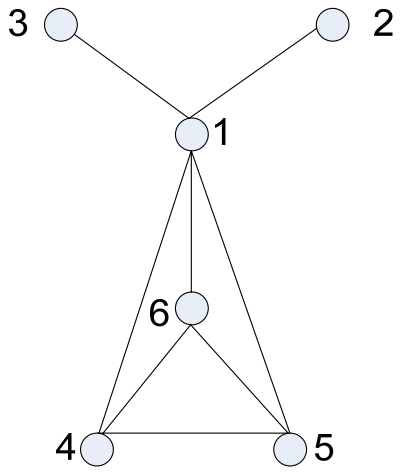



**Figure 4-7.** An example graph of maximal clique enumeration problem

Based on this graph's connection information, we can get maximal 2-cliques and candidate 2-cliques as:

maximal 2-cliques: {1,2}, {1,3}
candidate 2-cliques: {1,4|5,6}, {1,5|4,6}, ~~{1,6|4,5}~~, {4,5|1,6}, ~~{4,6|1,5}~~, ~~{5,6|1,4}~~

where the crossed elements in candidate cliques are the cliques which cannot be expanded according to canonical order. Expanding candidate 2-cliques, we can find that there are no maximal 3-cliques and candidate 3-cliques are:

candidate 3-cliques: {1,4,5|6}, ~~{1,5,6|4}~~, ~~{4,5,6|1}~~

Finally, expanding candidate 3-cliques we can obtain the maximum clique {1,4,5,6} and stop the program since there are no candidate 4-cliques.

The cost of Zhang's algorithm is still an exponential order of the graph size. As we mentioned that maximal clique numeration is an NP-hard problem, large computational cost cannot be avoid. However, from the design and the example we can see that Zhang's algorithm has the ability to provide useful partial results during the processing. Zhang's algorithm generates cliques in an increasing order of size in stages. At the end of stage *k*, partial results which consist of all cliques within size *k* are returned to the user. From this point of view, Zhang's algorithm is a kind of anytime approaches. However, this algorithm cannot be

directly employed in our methodology since it does not have the ability to handle graph's dynamic changes. In later chapters, we will present details how we modify Zhang's algorithm in order to fit in our anytime-anywhere methodology.

## *4.7 Comparisons and Summarizations of Selected SNA Metrics*

We have discussed three SNA metrics: ego-betweenness centrality, closeness centrality, and maximal cliques. For each metric, we introduced its corresponding algorithms which are broadly employed in current SNA software tools or by SNA researchers. In order to analyze how well current approaches are suitable for handling large social network, we need to study them from at least the following 4 aspects:

- required information for calculation
- computational cost
- anytime property
- anywhere property

First, required information for calculate is important for large social network analysis. When social networks are large, it is infeasible to put the whole network on a single processor. In a parallel/distributed framework, each processor will only focus on a part of the large network. Second, algorithm's computational cost is a significant factor which should always be considered. Usually, we not only want to comprehensively analyze large networks but also want to obtain the analysis results as soon as possible. Furthermore, according to the challenges of long processing time and graph's dynamism, anytime and anywhere properties are critical concerns for algorithms to effectively handle large social networks.

### a) Required Information

From definitions of ego-betweenness centrality and maximal clique we can see that these two metrics are focused on the first-order zone graphs. Ego-

betweenness centrality is measured on ego-networks which only consider the interactions between the ego and its alters and among alters. Maximal cliques also focus on direct connections between vertices. Taking a sub-graph as an ego (called as *super-ego*), we can build a super-ego-graph which contains the interactions within the super-ego, and interactions between super-ego and its alters and among its alters. A super-ego-graph is just one step farther away from the sub-graph generated in the Domain Decomposition Phase. Ego-betweenness centrality and maximal clique can both be measured based on a super-ego-graph. Thus, we say that measuring these two metrics only requires local graph information.

Measuring closeness centrality is different. From the definition of closeness centrality we can see that it is measured based on the geodesic paths between pairs of vertices. Finding the shortest paths between a pair of vertices may use connection information of all vertices. Some types of approaches, such as Dijkstra's algorithm, require the knowledge of the whole network's connection information. Other types of algorithms, such as DVR algorithm (a distributed version of Floyd's algorithm), can focus on local sub-graph information and incrementally refine it local solution by the results obtained from other sub-graphs.

**b) Computational cost**

Computational cost is an important parameter to indicate the bound of time cost for an algorithm to solve a specific problem. We have already presented them for each algorithm in previous sub-sections. Some algorithm's computational cost depends on the maximum degree or number of graph edges, such as Dijkstra's algorithm and Everett's algorithm. In the worst case (when every vertex has connections to all other vertices), these two parameters can be $n-1$ and $n(n-1)$ respectively. For the worst cases, the algorithms for these metrics are ranged in an increasing order of computational cost as: closeness centrality, ego-betweenness centrality, maximal clique enumeration.

In the worst case, the algorithms for ego-betweenness centrality will cost longer period of time than closeness centrality algorithms. This is not surprising since that under this case each ego-network is the whole network. We repeatedly process the whole network *n* time. However for large social networks, due to limitations on social power, it is almost-impossible for a vertex to have connections as a linear order of the graph size. From the discovered special properties (discussed in section 2.2) we can see that it seems that vertex's degree distribution for real-world large social networks follows a power-law. Normally, the degree of most vertices or the number of edges in a large social network is usually on the logarithm order of the network size, O(log*n*). The work load of using Dijkstra's algorithm to measure closeness centralities for normal large social networks is $O(n^2 \log n)$. Everett's algorithm and modified Dijkstra's algorithm for ego-betweenness will both have computational cost as $O(n \log^3 n)$ for processing normal large social networks.

Computational cost for the maximal clique enumeration problem can also have upper bound on large social networks with power-law distribution. It has been presented in [DuWXWP'06] that the work load of BK algorithms on real-world large social networks is:

$$O(\Delta \cdot M_C \cdot Tri^2)$$

where $M_c$ is the maximum clique's size. Limiting the maximum degree as O(log*n*), we can get the bound of BK algorithm's computational cost as

$$O(\log n \cdot M_C \cdot Tri^2)$$

By similar analyses in [DuWXWP'06] we can get that Zhang's algorithm has the same computational work load bound as BK algorithm.

**c) Anytime and Anywhere Properties**

Anytime and anywhere properties are important for effectively analyzing large and dynamic social networks. In common approaches for SNA, usually there is no consideration of the anywhere property. As we mentioned, current software

tools handle graph dynamism in a quite naive way that when a network is changed, they just discard the obtained results and re-analyze the whole network from scratch.

Not all approaches for SNA metrics have the anytime property. Some specific SNA metrics are very simple, such as ego-betweenness centrality and degree centrality. These metrics are defined only on a small part of the graph and can be obtained with very low computational cost for normal social networks. Analyzing them do not requires the anytime property. However, for most SNA metrics, the anytime property is critical. Although some of algorithms for these metrics can be interrupted and present some partial results, it is vital to check if there is a mechanism to estimate the quality of the returned results.

**d) Summarization**

Based on the discussion of selected SNA metrics we can see that ego-betweenness centrality seems to be the easiest one to measure. In the definition, ego-betweenness centrality is only focused on the first-order zone of an ego vertex. From the computational work load, normally measuring ego-betweenness centralities will relatively take a very short period of time. In fact, ego-betweenness centrality is chosen as a fundamental test case to primarily check if there is any flaw in our design which will degrade our methodology's performance. Closeness centrality is a representative problem with middle difficulty. The time cost for this metric is about the third order of graph size. We chose maximal clique enumeration problem as our test case for the hardest problems because in the worst case, finding all maximal cliques will take exponential costs on both time and memory.

Properties of selected SNA metrics and their corresponding algorithms are summarized in Table 4-1.

**Table 4-1** Summary of selected SNA metrics.

| Algorithm Properties | Ego-betweenness | | Closeness | | Maximal Clique | |
|---|---|---|---|---|---|---|
| | Everett's algorithm | Modified Dijkstra's algorithm | Dijkstra's algorithm | Flody's (or DVR) algorithm | BK algorithm | Zhang's algorithm |
| Required information | Local | Local | global | Local* | Local | Local |
| Time cost in worst case | $O(n\Delta^3)$ | $O(n\Delta\lvert E^*_z\rvert + n\Delta^2 \log\Delta)$ | $O(nm + n^2 \log n)$. | $O(n^3)$ | NP-hard | NP-hard |
| Time cost for NLSNs | $O(n\log^3 n)$ | $O(n\log^3 n)$ | $O(n^2 \log n)$. | $O(n^3)$ | $O(M_c \log n\, Tri^2)$ | $O(M_c \log n\, Tri^2)$ |
| Anytime property | N | N | N | Y* | N | Y |
| Anywhere property | N | N | N | N | N | N |

Note: NLSN stands for Normal Large Social Network. The *local** in Floyd's algorithm represents that the algorithm can work on local information. However, in order to achieve correct results, it needs to information shared from other processors. The *Y** in Floyd's algorithm represents that the algorithm does not have anytime property for local sub-network. But, it can be modified to have anytime property for generating closeness centrality for the whole network.

## *4.8 Summary*

In order to validate the effectiveness and evaluate the performance of our anytime-anywhere methodology, we decide to implement and study our methodology on three selected SNA metrics, ego-betweenness, closeness centrality, and maximal cliques. In this chapter we briefly introduce definitions, significances and popular algorithms for these SNA metrics. The selected metrics not only are indispensable for general SNA applications but also cover a broad range of difficulties, according to both computational complexities and different types of required graph information. We believe that evaluating our approaches on these selected SNA metrics can provide comprehensive study and strong validation for the effectiveness of our methodology. In what follows, we will provide details about our design, implementation, and theoretical performance analyses of the approaches designed based on our methodology for these three selected SNA metrics.

# 5. Domain Decomposition & Initial Approximation Phases Implementation

As we mentioned in Chapter 3, our anytime-anywhere methodology are consisted of three main phases (modules), Domain Decomposition (DD) phase, Initial Approximation (IA) phase, and Recombination (RC) phase. The anytime and anywhere properties of our methodology are mainly manifested in the Recombination phase. Thus, we use a separate chapter to discuss our design and implementation of the Recombination phase. In this chapter, we will focus on the implementation of our Domain Decomposition phase and Initial Approximation phase.

## *5.1 Domain Decomposition Phase Implementation*

The DD phase in our methodology is proposed for partitioning a large social network into smaller parts which fit for being handled on single processors. This phase is very important and has significant influences on our methodology's performance. The results obtained from small sub-networks, which are generated in the DD phase, will be taken as an initial approximation of the analysis of the whole network. How well the network is decomposed in the DD phase will directly affects the quality of the initial approximation. Also, in the RC phase, in order to achieve the exact or accurate enough results of some SNA metrics, such as closeness centrality, each processor may need to communicate with others and refine its local results based on the results obtained elsewhere. The work remaining in RC phase is also affected by how networks are decomposed.

Reiterate that in a general point of view, the DD phase essentially is a constrained multi-objective optimization process. The multiple objectives are

related to the quality of initial approximation and the work load remained in the recombination. The constraints are generated by the specific requirements of real applications and users. The task of the DD phase is using relatively small time to finding a solution which has optimal, or at least optimized, objectives and satisfies all constraints. The DD phase's architecture is shown in Figure 5-1.

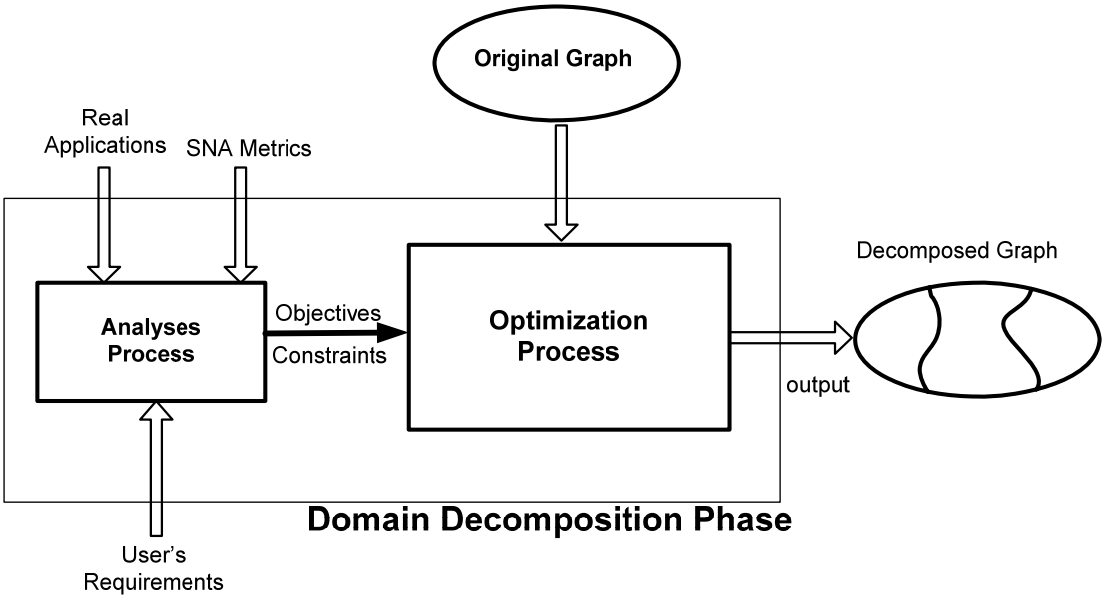



**Figure 5-1.** The architecture of the Domain Decomposition phase**.**

As we discussed in section 3.3, according to various SNA metrics and applications, there are many factors which may affects the quality of graph domain decomposition. How to generate proper objectives and constraints for graph decomposition is a big research topic which will take a long time to study. In this dissertation, my interest is to primarily design and validate our methodology for large social network analysis. Thus, in the initial implementation, we only focus on the most fundamental factors. Since each phase in our methodology can be taken as a plug-and-play module, we can easily adjust the currently employed objectives and constraints, even the architecture of the whole DD phase.

As defined in Chapter 4, *cut-size* is the sum of edges whose endpoints belong to different sub-graphs. Cut-size is one of the most fundamental and universal

factors affecting DD phase's quality. This is because that for most, if not all, cases the less information corrupted in the graph decomposition, the more accurate the results obtained in the IA phase and the less work remained in the RC phase. In our initial work, we use the cut-size to direct graph domain decomposition. In order to achieve better efficiency in a parallel/distributed computational framework, sub-graphs obtained in DD phase should have similar size. The task of DD phase is to partition a large graph into balanced small sub-graphs with optimized cut-size.

**a) Graph Domain Decomposition Architecture**

Generally, graph partitioning (decomposition) is a NP-Complete problem. It is not practical to partition large graphs with a global optimal cut-size. In fact, researchers usually employ some heuristics to achieve optimized cut-size. Multilevel graph partitioning algorithm together with heuristic refinements on each level is a popular and effective method [Hendrickson93, BarnardS'94, KarypisK'99, KayehR'00, SoperWC'04]. The essential idea of this approach is collapsing nodes with strong connections to coarsen the large graph into a smaller one level by level, then generating good initial partition on the smallest graph according to the objective function, finally mapping and refining the graph partition back to the original graph level by level. The brief procedures of this approach are shown in Figure 5-2.

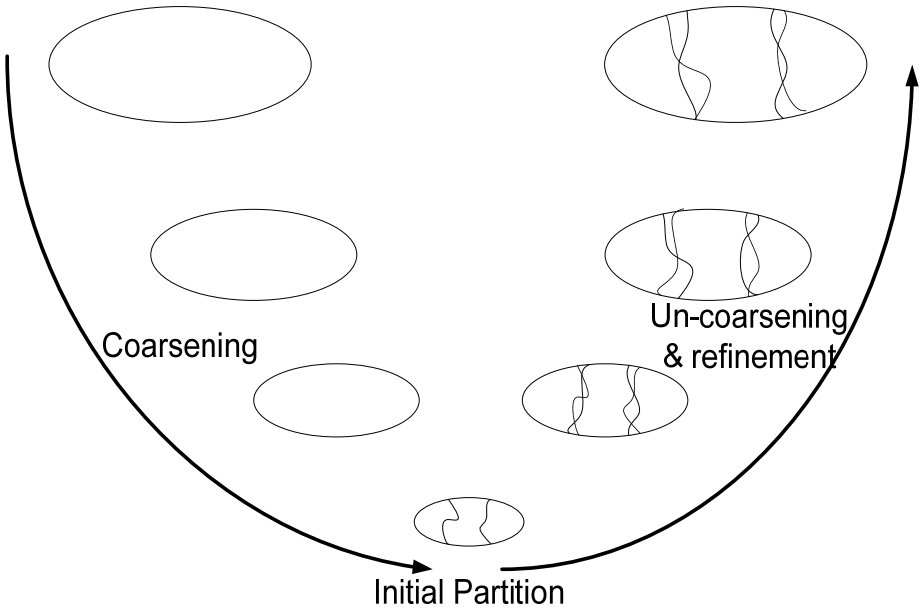



**Figure 5-2.** The structure of graph domain decomposition approach

Usually, this kind of approach consists of three phases: Coarsening, Initial Partition, and Un-coarsening & Refinement. Taking cut-size as the optimization objective, our graph partitioning algorithm is designed based on the multilevel graph partitioning algorithm proposed in [Karypis98, KarypisK'99]. According to SNA applications, we made some modifications of this algorithm.

### b) Coarsening Phase

We can first coarsen the graph step by step down to a graph with a few hundred nodes by generating a sequence of smaller graphs $G(i)$. For each coarsening step, we choose the nodes that are highly related to each other to collapse into a super-node. This process can be formally defined in terms of matching. Since the goal of collapsing nodes is to decrease the graph size, a maximal matching is desired. In our SNA application, the network uses edge weights to indicate the strength of the connection between nodes. In most cases two nodes with strong connection may be highly related or similar. Thus, we would like the matching to have heavy edge weight. In our coarsening phase, we use a heavy-edge matching algorithm [KarypisK'99] to build the maximal matching for each step in

the coarsening phase. The idea behind the algorithm is the following: randomly choose an un-collapsed node *a* in *G*(*i*), check its neighbor, collapse it with the neighbor *b* who is un-collapsed and has the strongest connection with this node, and use a super-node *x* in *G*(*i+1*) to represent the combination of these two nodes. The edges between *a* and *b* will disappear and edges connecting *a* and *b* to other nodes will be merged together. This process is repeated until the graph has been reduced to a manageable size, as defined by the user.

**c) Partition Phase**

After coarsening the graph, we get a small graph, *G*(*s*), with a few hundred nodes. We can generate a good graph cut for this small graph within a short amount of time. Here we need to do *k*-way graph partitioning. The parameter *k* is determined the by structural characteristics of graph. However, these features are just what the SNA tools used to measure or analyze,. Before cutting the graph, we usually do not know what the proper value of *k* is. Thus, in our implementation, we ask the user to specify a threshold for cut-size. Within this cut-size, we cut via bisection recursively, trying to partition the initial graph into as many parts as possible until the threshold is reached. In partition phase, we use a simple breadth-first growing algorithm to increase the size of a partition which originates from a single seed of high degree. We chose this approach to minimize the chance that vertices of high degree will end up near the edge of a partition.

**d) Un-coarsening Phase**

Each vertex *v* in graph *G*(*i+1*) contains a distinct subset of vertices in graph *G*(*i*). When we project the graph back from *G*(*i+1*) to *G*(*i*), we will have more degrees of freedom. It is very likely to be able to find a smaller cut-size for *G*(*i*) than *G*(*i+1*). This means after finding the graph partition *P*(*i+1*) which is minimized on *G*(*i+1*), we should perform some refinement on partition *P*(*i*) to get the minimized cut-size for *G*(*i*).

Fiduccia and Mattheyses (FM) algorithm [FiducciaM'82] is a good candidate algorithm for this refinement. It is an improvement of the Kernighan-Lin (KL) algorithm [KernighanL'70]. It has low computation cost and good performance in practice. However, the original FM algorithm cannot be directly applied to more than two partitions. There are several variants of the KL algorithm for *k*-way partition refinement, such as Generalized KL, Greedy Refinement (GR) and Global Kernighan-Lin Refinement (GKLR) [KarypisK'98]. These algorithms try to achieve minimized cut-size while maintaining balanced partition size—making them ideal for distributing data among processors. GKLR approach seems to have good performance [KarypisK'98]. We employed GKLR in our graph decomposition algorithm.

## *5.2 Initial Approximation Phase Implementation*

The task of the IA phase is to comprehensively analyze the sub-graph stored locally at each processor and use the analysis results as an initial approximation of the whole network. It is apparently helpful to employ current SNA techniques in the IA phase. However, we implemented our own approaches for the IA phase. The reasons we do so are as follows. First, our parallel methodology is built on a cluster of processors with Linux system. Many SNA software tools do not support Linux operation system right now. Second, most SNA software tools do not provide API interfaces. It is difficult to encapsulate them into our system. Finally, few of current commercial SNA software provide the function to measure ego-centric betweenness centrality for every node. Usually, they only perform ego-centric analysis on specified actors. We believe that, as commercial SNA software evolves, there is the potential that they can be utilized in our IA phase.

Currently, in the IA phase, we employed modified Dijkstra's algorithm introduce in section 4.5 for measuring ego-betweenness centrality. For closeness centrality, we use Floyd's algorithm [McHugh'90]. Maximal clique enumeration problem is an NP-hard problem. In some extreme cases, even when sub-graphs are

relatively small (several thousands nodes), it may cost a very long time to generate the complete results. Thus, on each processor, we run Zhang's algorithm [ZhangABCLS'05] on the locally stored sub-graph and present all cliques with size 2 (maximal 2-cliques and candidate 2-cliques) as the initial approximation of the whole network.

## *5.3 Summary*

In this chapter, we provided the realization of the DD phase and the IA phase for the SNA metrics chosen for our methodology's validation. All the work we presented in this chapter is an initial implementation of these two phases. In our current work, especially for the DD phase, we only focus on the most fundamental and universal concerns which we believe to be sufficient for our methodology's primary validation. There is, of course, much space for the refinement and improvement on the design. Fortunately, our methodology for large social network analysis is designed on a modular framework in which each phase is a plug-and-play module. Without changing our methodology's framework, we can further study and refine our phase design in the future.

# 6. Recombination Phase Implementation

In the DD phase and IA phase, within a relatively small time, the original large social network is decomposed into smaller parts and each part is distributed to and analyzed at a single processor. These results are not accurate or complete because that they only focus on the separated sub-graphs (i.e. for closeness centrality) or a small portion of the original problem (i.e. for maximal cliques). In order to achieve complete results, we should either further analyze the locally stored sub-graph (i.e. for maximal cliques) or refine local results by the results obtained in other processors (i.e. for closeness centrality). This is one of the main tasks for the RC phase. Another main task for the RC phase is handling the graph's dynamism. As we mentioned, most, if not all, social networks are dynamic. There will be edge/vertex changes during the analyses process. These dynamic changes will be effectively handled in the RC phase.

In this document, according to different requirements of SNA metrics, we designed and implemented various recombination algorithms for them. In this chapter, we will first introduce the general analyses of anytime-anywhere approaches for large social network analysis. Then, we will present details about recombination algorithms for each selected SNA metrics respectively.

## *6.1 General Anytime Recombination Algorithm's Design*

In this document, anytime algorithms are defined as those approaches which can partially and incrementally process SNA metrics and present useful partial results to the user during the processing. In order to effectively and efficiently generate partial results, the SNA metric should have the property that it can be recursively

defined or can be calculated by dynamic programming algorithms. This means that the results for next stage can be generated from obtained partial results.

There are mainly four characteristics which make anytime algorithms different from normal approaches. These characteristics are: quality measurements, predictability, interruptability, and monotonicity. In order to obtain an effective anytime algorithm, all these characteristics should be considered. During the analysis process, an anytime algorithm can be interrupted in middle and can present obtained partial results. Also, in order to make better use of obtained partial results, there should be a mechanism to measure the quality of these results. Partial results' quality must be non-decreasing over time. Moreover, we should have the ability to estimate or bound the time cost for achieving partial results.

According to characteristics of anytime approaches, in the initial implementation of our methodology, the anytime recombination algorithm takes the form of

$$f(X_i) = F\big(f(X_{i-1})\big)$$
$$Quality_i = Q\big(f(X_i)\big) \quad where\ i = 1\ to\ k$$
$$and \quad \Psi(X_0) \approx f(X_k)$$

where $f(X_i)$ is the collection of results returned at the $i^{th}$ stage, $X_0$ is the set of initial inputs, $\Psi(X_0)$ is the correct results, and $Quality_i$ is the quality of the results returned at $i^{th}$ stage. In our initial implementation, the partial results for next stage are generated from the obtained results on the current stage. According to the characteristics of anytime approaches, the *Quality* function is monotonically non-decreasing. The results finally returned to users are either correct results or good approximation.

## *6.2 General Anywhere Recombination Algorithm's Design*

Social networks are dynamic. There may be different types of changes on a social network. Vertices or edges can be added or removed. Edge weights can

be increased or decreased. However, all these changes can be generalized as changes on edge weights. Adding an edge can be treated as decreasing this edge's weight from infinity to a real value. Removing an edge can be taken as increasing the edge weight to infinity. Adding/removing a vertex, in fact, is a set of edge addition/removal. Therefore, in our methodology validation, we only focus on dynamic graphs with edge weight changes.

Anywhere recombination algorithms are focused on those approaches which can effectively handle graph dynamic changes during the processing. Handling dynamic graphs is a very hard task. The simplest and most straightforward way to deal with dynamic graphs is discarding obtained results and re-analyzing the graph from scratch. However, as we mentioned, this type of approaches has a formidable overhead and are not suitable for handling large social networks.

In order to effectively and efficiently adopt graph's dynamic changes during analysis process, anywhere algorithms should at least have the following abilities:

1. Identifying the range of dynamic changes' effects. Usually, when a dynamic change happens, not all obtained results will be affected. There will be a lot of elements which are still correct in the result set. Recalculating these valid results will be a horrible waste. Also, in order to achieve good efficiency, it is better to calculate the affected elements' values based on the results we have already obtained. Thus, it is critical to identify the range of the effects of dynamic changes. Anywhere algorithms should have the ability of identifying either un-changed elements or potentially affected elements in the obtained result set.
2. Identifying the way that dynamic changes affect on the result set. We know that dynamic changes in graphs will affect obtained results. However, according to different SNA metrics, the ways in which these dynamic changes affect are different. For some metrics, such as ego-betweenness centrality and closeness centrality, the affected elements will still be contained in the final results but with new values. For other metrics (i.e.

maximal cliques), elements which are affected by graph's dynamic changes may become invalid anymore and will be removed from the result set. In order to incorporate dynamic changes, the anywhere approach should be able to find in which way dynamic information affects the obtained results.

3. Recalculation based on obtained results (or partial results). In dynamic graph processing, usually we have already obtained some partial results (or the final results) when some graph changes happen. With the ability of identifying effect range of dynamic changes, we can process the affected (or potentially affected) elements in two ways: recalculate from scratch but only focusing on affected elements, or refine the affected elements based on the obtained unaffected results. Usually, using obtained results we can avoid redundant work and finish the dynamic change adoption faster.

In following sections of this chapter, we will present how we design our anywhere approaches for each selected SNA metric.

## *6.3 Ego-Betweenness Centrality Recombination Approach*

Among the three selected SNA metrics for our methodology validation, ego-betweenness centrality seems to be the easiest one to handle. The definition of the ego-betweenness centrality shows that this metric only focuses on the first-order zone of a vertex. From this point of view, in the process of measuring the ego-betweenness centrality, the large social network is automatically decomposed into small parts and the direct results can be achieved with a small time and memory cost. We implemented the approach for ego-betweenness centrality in our methodology as a sanity test to check if there is any flow in the design which will degrade our methodology's performance on large social network analysis. In this chapter, we will present details about how ego-betweenness centrality is incorporated in our anytime-anywhere methodology.

As we mentioned in section 4.7, not all SNA metrics require the anytime property. Some SNA metrics, such as degree centrality, ego-betweenness centrality, etc, are so simple that they can be directly measured within a very short period of time. There are no needs for partial and incremental processing of these types of SNA metrics. Ego-betweenness centrality is directly measured in the IA phase by the modified Dijkstra's algorithm introduced in section 4.5. In the RC phase, we only implement an anywhere approach for ego-betweenness centrality measurement.

### a) The Anywhere Approach for Ego-Betweenness Centrality

In this section, we will present details about how we design the anywhere algorithm for ego-betweenness centrality measurement according the concerns of general anywhere approaches discussed in section 6.2.

*Effect Range*: When an edge's weight is increased or decreased, not all vertices' ego-betweenness values will be changed accordingly. An edge change only affects those vertices whose ego-networks contain this edge. According to the definition of ego-network, these vertices consist of the dynamic edge's endpoints and their common neighbors. This is shown in Figure 6-1.

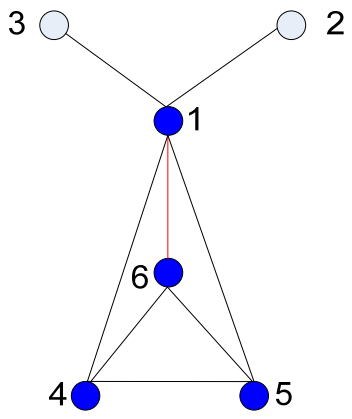



**Figure 6-1.** An example of the ego-betweenness dynamic change's effect range

In this figure, the red edge *e*(1,6) represents the dynamic edge in the graph. The vertices whose ego-networks contain this edge are the blue vertices,. Only these

vertices' ego-betweenness centrality values may be affected. Ego-betweenness centrality values of vertices 2 and 3 will keep unchanged.

*Effect Way*: As we mentioned, in the initial implementation of our methodology, we only focus on edge weight changes in anywhere approach design. For ego-betweenness centrality, after an edge's weight changes, we still need to present ego-betweenness centrality value for each vertex. The affected elements in the results set will be replaced by new values.

*Recalculation*: Ego-betweenness centrality is a simple SNA metric which can be directly measured within a very short period of time. In our anywhere approach for ego-betweenness centrality, we just recalculate the affected vertices based on their new ego-networks.

*The Approach*: Combining these concerns together, the pseudo-code of our anywhere approach for ego-betweenness centrality measurement is shown in Figure 6-2. It is easy to get that the work load of this algorithm is bounded by $O(\Delta^2|E^*_z|+\Delta^3\log\Delta)$. For normal large social networks, the work load is $O(\log^4 n)$.

```
1. initialization:
      a): get the new edge weight w'(a,b);
      b): initialize Q, a set of vertices, as {a,b};
2. get all common neighbors of a & b into Q;
3. while Q is not empty
4.       pick the first vertex v contained in Q;
5.       generate the new ego-network of vertex v;
6.       calculate v's ego-betweenness value;
7.       remove v from Q;
```

**Figure 6-2** Algorithm I: the anywhere recombination approach for ego-betweenness centrality measurement

## *6.4 Closeness Centrality Recombination Algorithm I – The Anytime Approach*

Closeness centrality measurement requires the knowledge of All-Pair Shortest Paths (APSP) distances. Solving the APSP problem for large graphs will cost a long period of time and great amount of computational resources, such as memory. Researchers have already tried to develop parallel algorithms for the APSP problem, such as the parallel version of Floyd's algorithm [Quinn'03] and the specific algorithm presented in [HanPR'97]. These algorithms can more-or-less improve the processing speed of APSP problem. However, they do not fit for analyzing large social networks due to the absent of anytime property. They can not provide usable partial results. Moreover, these approaches are designed for static graphs and cannot adopt dynamic changes of networks during processing.

In this section, we will introduce our anytime approach for the closeness centrality measurement. Our approach is designed based on the Distance Vector Routing (DVR) algorithm [KuroseR'01], which is a distributed version of Flody's Algorithm and has been frequently used in network routing. This algorithm works based on the fact represented by the following formula presented in [KuroseR'01]:

$$D^x(Y,Z) = c(X,Z) + \min_w \left\{ D^z(Y,w) \right\}$$

where $D^x(Y,Z)$ is the shortest path distance from $X$ to $Y$ via $X$'s direct neighbor $Z$, $c(X,Z)$ is the distance between $X$ and $Z$, and $min_w$ term is taken over all of $Z$'s direct neighbors. In our approach, we use a vector to store the geodesic distance information of a node $v_i$, which is denoted as $DV(v_i)$. In $DV(v_i)$, the element $DV_j(v_i)$ represents the obtained geodesic distance from node $v_i$ to $v_j$. Recall that our anytime approach will provide usable partial results for analyst during processing. The elements contained in DV*s* may be immediate results, in stead of the exact geodesic distance which can be obtained by complete analysis of the whole graph. These immediate results will be refined step by step.

The main idea of DVR algorithm is updating local optimal route by the information obtained from neighbors. In this algorithm, each router/computer is a node. Each node maintains a distance table which contains its neighbors' shortest distances to all other nodes. A node tries to identify its most efficient route to the target by checking its neighbors' distances to the target. Our RC phase's task is similar to this network routing situation. In our system, after the DD phase, the large graph is decomposed into interrelated sub-graphs/parts. Taking each sub-graph as a super-node, we can build a super-graph accordingly, as shown in Figure 6-3. In this figure, the original graph is decomposed into 6 parts which are represented by dashed circles. Red nodes stand for boundary nodes. Unlike the Distance Vector Routing problem in which each node only represents one router/computer, in our combination problem, each "node" is in fact a super-node which contains a group of nodes and their connections.

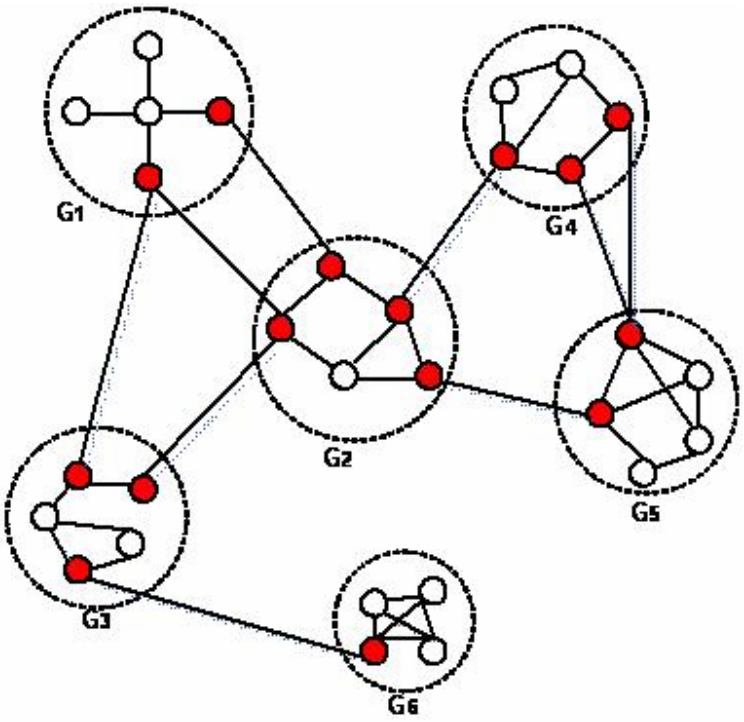



**Figure 6-3.** An example of super-graph based on decomposition of original graph

**a) The Approach**

Each processor only handles a part of the original network. We call the set of nodes which are contained in the locally stored sub-graph as *local nodes* and the set of nodes belonging to other sub-graphs as *outside nodes*. Define *global nodes* as the combination of these two sets. Each processor $P_i$ stores a *distance matrix* $D_i$ whose elements are the obtained shortest distances from local nodes to global nodes. In the DD phase, each sub-graph is assigned to a separate SNA processor. After the IA phase, each agent obtains the shortest distances from their local nodes to all other nodes through paths which consist of only the edges contained in the local sub-graph. In other words, the shortest paths obtained in the IA phase at each processor are generated from an edge pool which only contains edges in the agent's local sub-graph.

From Figure 6-3, we can see that boundary nodes are bridges connecting local connections with connections contained in other processors. Only through boundary nodes outside processors' information can affect local results. Thus, in our recombination approach, each agent maintains a table of distance information of boundary nodes' outside neighbors. This table is denoted as *outside distance table*.

In order to easily estimate the quality of partial results obtained at each agent, synchronous algorithms are employed. In each stage, each processor will first gather all related information from its neighbor processors into its outside distance table. For a processor, the *neighbor processors* are those whose local graphs have ties connecting with its boundary nodes. Then, each processor refines its boundary nodes' DVs based on the new information contained in the outside distance table. After this, all local nodes' DVs are updated. The update is accomplished in the similar way as Floyd's algorithm. The only difference is that, in stead of using all local nodes' information, the update is only based on the new information contained in boundary nodes. Finally, each processor will send its new results to its neighbors. The working procedures of our anytime approach for

closeness centrality measurement are shown in Figure 6-4. In this figure, $p$ represents the number of sub-graphs which the original graph is decomposed into, $P_i$ is the $i^{th}$ processor which handles the sub-graph $G_i$, $C_i$ is the set of cut-edges of $G_i$, and $D^i_j$ is the distance matrix in processor $P_j$ at stage $i$.

1. Initialize the RC phase: set combination step index *ind* as 1; treat the distances from local nodes to external nodes as infinity and generate $D^0_i$ based on the local all-pair geodesic distance information obtained in the IA phase; set each local boundary node's DV as new DV.
2. Propagate new information: go through local boundary nodes, prepare and send new DVs to all direct neighbors $P_j$.
3. Gather new information, update outside distance table and local boundary nodes: receive new DVs from all direct neighbors, update local boundary nodes DVs based on $C_i$ and the new DVs received from other processors.
4. Update local information: calculate $D^{ind}_i$ based on new local boundary nodes DVs, and inform new results to SNA users
5. Checking convergence of the algorithm: if *ind* = *p-1*, then terminate the combination. Otherwise, go to next step.
6. Identify new information: set all local boundary nodes DVs which have been changed in step 4 as new DVs.
7. Synchronization: wait until all processors finish step 6, then go back to step 2.

**Figure 6-4.** Algorithm II: the anytime recombination algorithm for closeness centrality measurement.

### b) Validity of the Approach

The proof for our anytime algorithm's correctness is similar to the one for Distance Vector Algorithm [GoodrichT'02]. Here, we use an example to demonstrate the validity of our algorithm. Assume after the DD phase, the original graph is partitioned as shown in Figure 6-3. If we treat each sub-graph as

a super-node, the original graph can be transformed into a super-graph shown in Figure 6-5.

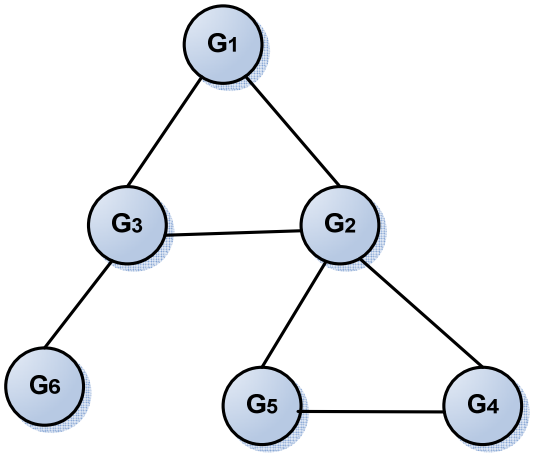



**Figure 6-5.** The super-graph obtained based on partitioning of the example graph.

Taking $P_1$ as the root in our example, the edge pool for shortest paths at $P_1$ after each stage is shown in Figure 6-6. Initially, after the IA phase, the pool on $P_1$ includes all edges contained in $G_1$. After the first stage, the pool on $P_1$ consists of the edges contained in $G_1$, $G_2$, $G_3$, $C_{1,2}$, and $C_{1,3}$, where $C_{i,j}$ is the connection between sub-graphs $G_i$ and $G_j$. After the second iteration, the pool contains edges in $G_1$, $G_2$, $G_3$, $C_{1,2}$, $C_{1,3}$, $G_4$, $G_5$, $G_6$, $C_{2,4}$, $C_{2,5}$ and $C_{3,6}$. Generally, taking $P_i$ as the root and building a breadth first search tree, the shortest paths at $P_i$ after the $x^{th}$ iteration are generated from a pool which contains edges in the super-nodes whose distance to $P_i$ is less or equal to $x$ and connections between these super-nodes except connections between super-nodes on level $x$. If the depth of the breadth first tree is $d$, the processor will achieve the correct shortest paths from local nodes to all nodes in the original graph after $d+1$ iterations. In this example, after the third iteration, the pool on $P_1$ will contain all edges in the original graph. Thus, we can see that the convergence of our combination algorithm depends on the depth of the breadth first trees. The worst case is that the super graph is a line. The algorithm will converge within $p-1$ iterations when the super-graph contains $p$ super-nodes.

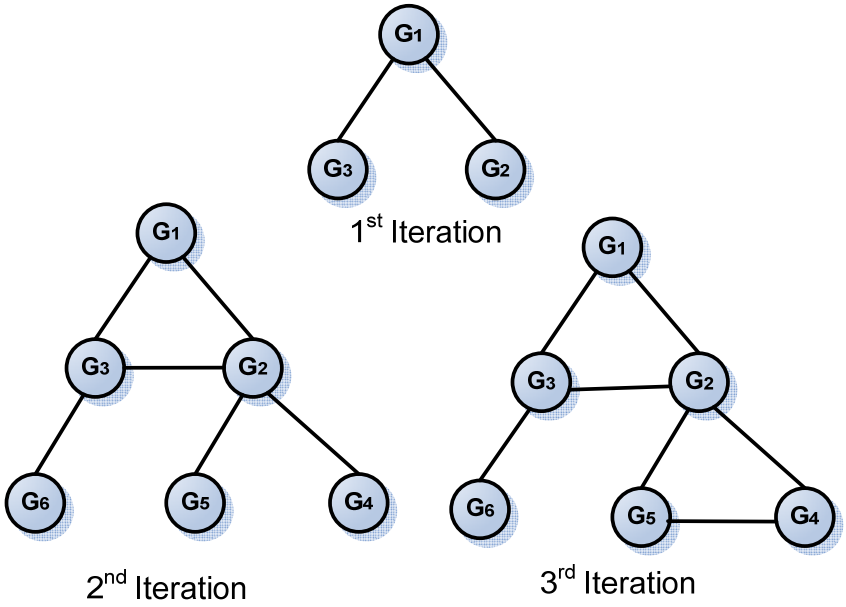



**Figure 6-6.** Edge pool for shortest paths at $P_1$

### c) Anytime Property of the Approach

Boundary nodes are bridges connecting local nodes to external nodes. For each processor, the edge pool for generating shortest path can be expanded only through boundary nodes. At the beginning of each stage (step 2 in Algorithm II), boundary nodes connection information is updated by results obtained from neighbor processors. This means that the local edge pool is expanded by edges contained in the neighbor processors' pools. Thus, we can see that information is propagated through a ripple-effect. In each stage, shortest paths are generated from a larger edge pool which covers edges contained in farther away processors. Closeness centrality measurements become more and more accurate since the quality of partial solutions is determined by how many edges are contained in the edge pool. By the synchronous scheme, the pool size can be easily estimated from the structure of the super-graph. The algorithm will stop when the pool covers all edges in the original large network. The number of stages for our algorithm to converge is limited by the diameter of the super-graph.

**d) Approach's Performance**

In the performance analysis, we mainly focus on the computation load. Omitting the communication cost, most work done in Algorithm II is at steps 3 and 4. Assume the value of graph size $|V|$ is $n$. For processor $i$, updating boundary nodes' DVs costs $O(n \cdot |C_i| \cdot \gamma_i)$ where $\gamma_i$ is the maximum number of cut-edges connecting to a boundary node in sub-graph $i$. Calculating $D^{ind}$ takes $O(n \cdot |V_i| \cdot |C_i|)$. Since we separate the original graph into $p$ sub-graphs with balanced size, we can approximate $|V_i|$ by $n/p$. Thus, after the synchronization step of each stage, the computation cost is:

$$O\left(n \cdot |C_i| \cdot \gamma_i + n \cdot |V_i| \cdot |C_i|\right) = O\left(n \cdot |C_i| \cdot \gamma_i + n^2 \cdot |C_i| / p\right) \quad (6.3)$$

In the worst case, the algorithm will converge after *p-1* stages (the diameter of the super-graph cannot larger than *p-1*). Thus, the running time for the whole algorithm is:

$$O\left(n^2 \cdot \gamma_i + n^2 \cdot |C_i|\right)$$

Values of $|C|$ and $\gamma$, in fact, depend on how well the graph is decomposed. Thus we can see that the performance of our anytime recombination algorithm for closeness centrality measurement is affected by how well the DD phase is designed. Researches of large networks [Newman'03, AlbertB'02] have shown that the distribution of node degree in a real-world large network follows a power-law. A quantity follows power law can be represented as $p(x)=cx^{-\alpha}$ where $p(x)$ is the fraction of a quantity with value of $x$, $c$ and $\alpha$ are positive constants. In real world large networks, $\alpha$ usually lies in the range from 1.6 to 3.0. For a network with large size, i.e. more than $10^5$, it is reasonable to limit $\gamma_i$, the maximum number of cut-edges connecting to a boundary node in sub-graph $i$, as

$$\gamma_i \leq n / \log n .$$

Also, $|C_i|$, the number of boundary nodes in a sub-graph, is limited by the sub-graph size

$$|C_i| \leq n / p .$$

Therefore, the running time can be formed as

$$O\left(\frac{n^3}{p}+\frac{n^3}{\log n}\right) \tag{6.4}$$

As we mentioned, each SNA processor updates its local information based on Floyd's algorithm. Therefore, our anytime approach is able to handle real weights (positive or negative), although our dissertation is only focused on graphs with positive real weights. To the best of our knowledge, the running time for the fastest serial algorithm for APSP problem with real edge weights is $O(n^3\sqrt{\log\log n}/\log n)$ [Zwick'04]. Thus, we can see that when O(log*n*) processors are employed, our parallel anytime approach can outperform the serial one, even when the original large graph is not decomposed well.

## *6.5 Closeness Centrality Recombination Algorithm II – The Anywhere Approach*

Graphs are dynamic. All graph dynamic changes can be generalized as edge weights' changes. There are two types for partial dynamism: decrease only and increase only. A fully dynamic APSP algorithm is designed to handle both types of partial dynamisms. The main objective of dynamic APSP algorithms is to calculate the shortest distances faster than beginning from scratch when a dynamic change happens. Solving Dynamic All-Paris Shortest Paths (DAPSP) problem is a hard task. There has been a lot of researches done on this problem. Many researchers developed DAPSP algorithms for special graphs with constraints to achieve better performance. Some algorithms are designed for analyzing planar graphs [HenzingerKRS'97, FakchareompholR'01]. Some approaches are focused on graphs with integer or limited number of different weight values [AusiellolMN'91, King'99]. In [King'99], an algorithm with O($n^{2.5}$(*S*log*n*)) worst case running time is proposed where *S* is the number of different edge weight values that appear in the calculation. A remarkable breakthrough was made by Demetrescu and Italiano [Demetrescu I'03] who

proposed a fully dynamic algorithm to solve a general digraph with non-negative real edge weight. The worst case running time of this algorithm is $O(n^2 \log^3 n)$. To our best knowledge, this is the fastest algorithm to solve the fully dynamic APSP problem. However, the design of the algorithm is focused on an assumption that all shortest paths are unique in the graph, which is un-realistic in the real world. In order to break multiple shortest paths, the algorithm requires additional storage in memory, which can be $O(n^3)$. It significantly limits its applicability on analyzing large networks. Thus, in order to handle general graphs, we developed our own anywhere approach for closeness centrality measurement.

**a) The Anywhere Approach for Decreased Edge Weights**

When an edge weight is decreased or increased, our approaches for adopting them in the graph analysis are different. In this section, we first focus on the decreased edge weights. Our anywhere approach for handling increased edge weights will be discussed in next section. Assume the weight of an edge $e(a,b)$ (directing from $a$ to $b$) is decreased from $w(a,b)$ to $w'(a,b)$.

*Effects Range*: When an edge's weight is decreased, there is no way to directly know which shortest distances will change. We need to check all elements in the distance matrix $D$ to see if they are affected by the new edge weight. For an element $d(u,v)$, it is affected by the new edge weight if there is a shortest path going through the edge $e(a,b)$. In other words, the new shortest path connecting from $u$ to $v$ will be a concatenation of the shortest path connecting from $u$ to $a$, the edge $e(a,b)$, and the shortest path connecting from $b$ to $v$. The changed elements can be identified by checking if they satisfy the following condition

$$d(u,v) < d(u,a) + w'(a,b) + d(b,v) \tag{6.5}$$

*Effect Ways*: For closeness centrality, after an edge's weight is decreased, we still need to present closeness centrality value for each vertex. Affected elements in the results set will be replaced by new values.

*Recalculation*: It is straightforward to obtain the new values of affected nodes' closeness centralities. We only need to replace the distance of old shortest path with the new smaller one.

$$d(u,v) = \min\big(d(u,v), d(u,a) + w'(a,b) + d(b,v)\big) \quad (6.6)$$

*The Approach* :\When the weight of an edge *e*(*a*,*b*) (directing from *a* to *b*) is decreased, from *w*(*a*,*b*) to *w'*(*a*,*b*), the algorithm for closeness centrality measure is shown in Figure 6-7.

```
1. for(i=0; i<n; i++)
2.     for(i=0; i<n/p; i++)
3.         D_{i,j} = min(D_{i,j}, D_{i,a}+w'_{a,b}+D_{b,j})
```

**Figure 6-7.** Algorithm III: the anywhere recombination algorithm for closeness centrality measurement when edge weight is decreased.

### b) The Anywhere Approach for Increased Edge Weights

When an edge's weight is increased, the anywhere algorithm is more complicated. In this section, we will discuss details about the design of our anywhere recombination algorithm for increased edge weights.

*Effect Range* :If the weight of an edge *e*(*a*,*b*) (directing from *a* to *b*) is increased, from *w*(*a*,*b*) to *w'*(*a*,*b*), the geodesic distance between a pair of node may be affected only if this edge stays on one of the shortest paths connecting this pair of nodes. Identifying whether edge *e*(*a*,*b*) is on the shortest path connecting nodes *u* and *v* can be achieved by check if the following condition is satisfied:

$$d(u,v) = d(u,a) + w(a,b) + d(b,v) \quad (6.7)$$

where *d*(*u*,*v*) is the geodesic distance between nodes *u* and *v*. We refer the pair of nodes which satisfy this condition as a *potentially affected pair* (PAP) of the dynamic edge *e*(*a*,*b*). For each PAP of *e*(*a*,*b*), there is at least one shortest path which connecting PAP and goes through this edge. Here, we use the term

"potentially affected" because that there may be multiple shortest paths which do not go through edge *e*(*a*,*b*) but have the same distance as the shortest path containing *e*(*a*,*b*). For these pairs, the geodesic distance will not change with the new edge weight.

*Effect Ways* :The same as decreased edge weights, after an edge's weight is increased, we still need to present closeness centrality value for each vertex. The affected element in the results set will be replaced by a new value.

*Recalculation*: When an edge weight is increased, the dynamic adoption is relatively complicated. It is nearly-impossible to directly find the new true geodesic distances of PAPs. This is because that PAPs may have alternative shortest paths which do not go through this dynamic edge. However, the new geodesic distances for PAPs can be upper-bounded as

$$Upper\big(d(u,v)\big) = d(u,v) + \big(w'(a,b) - w(a,b)\big) \tag{6.8}$$

This upper bound can be taken as an over-estimation of the geodesic distances of PAPs. Then, we will refine these over-estimated values by obtained unchanged results stage by stage.

The recalculation is designed based on two phases: overestimation and correction. At the beginning, all processors gather the information of dynamic changes, identify all PAPs and overestimate their geodesic distances as the upper bound shown in formula (6.8). In order to obtain the accurate results, each processor will first try to correct the overestimation based on the local graph connection and results previously obtained at other processors. The corrections are done by Floyd's algorithm on geodesic distance information contained in local nodes' DVs and the outside distance table. Then, similar to our anytime approach, each processor will update their new information (corrections) to its neighbor processors and corrections will propagate through the whole graph in a ripple-effect. In the worst case, it will require *p-1* steps to obtain the final correct answer, where *p* is the number of sub-graphs. In our methodology, our anytime

approach works as an iterative algorithm. Only after each stage, the partial results with a clearly defined quality are provided to the analysts. Thus, we gather all dynamic changes during each stage, and handle all gathered dynamic changes together at the end of each stage to avoid frequently interrupting the program and decrease the additional cost for processing each individual change.

### c) The Fully Dynamic Anywhere Approach

Combining the anywhere approaches for decreased and increased edge weights together, the procedures of our fully dynamic anywhere recombination algorithm for closeness centrality measurement are shown in Figure 6-8.

1. Identify and propagate new information: set all local boundary nodes DVs which have elements with changed value as new DVs, and send them to the relative neighbor agents
2. Gather all related external information, update local outside distance table and boundary nodes DVs
3. Update local information: update un-converged PAPs' geodesic distances by the external information
4. Gather all dynamic edges' information: DVs of both end points of the dynamic edge
5. Dynamic change local adoption: for each dynamic edge: if edge weight is decreased, update each geodesic distance by Algorithm III; if edge weight is increased, identify the PAPs and do the overestimation
6. Overestimation local correction: recalculate the PAPs' geodesic distances obtained in step 5 based on local nodes' DVs
7. Check convergence of the algorithm: if *converged*, then terminate. Otherwise, go to next step.
8. Synchronization: wait until all processors finish step 7, then go back to step 1.

**Figure 6-8.** Algorithm IV: the fully dynamic anywhere recombination algorithm for closeness centrality measurement

**d) Validity of the Approach:**

There are two types of edge weight dynamisms: decreased weights and increased weights. When an edge weight is decreased, the new shortest paths must go through this edge and have smaller length than the old ones. The smallest length of paths which go through edge *e*(*a*, *b*) can be identified by:

$$d_{a,b}(u,v) = d(u,a) + w'(a,b) + d(b,v)$$

Therefore, we update the geodesic distance between each pair of node by:

$$d(u,v) = \min\begin{cases} d(u,v) \\ d_{a,b}(u,v) \end{cases}$$

where $d_{a,b}(u,v)$ is calculated based on the new edge weight of *e*(*a*,*b*). Essentially, this formula is the same as (6.6).

When an edge weight is increased, only *potentially affected pair*'s (PAP) geodesic distance may change. Recall that for a specific dynamic edge *e*(*a*,*b*), its PAPs are pairs of nodes which have at least one shortest path going through this edge. If after *e*(*a*,*b*)'s weight is increased a new shortest path has a smaller length value than that obtained in the overestimation, it means this new shortest path does not go through *e*(*a*,*b*). We use *p* to present this path. For a potentially affected pair of nodes (*u*,*v*), the set of its shortest paths which go through the dynamic edge *e*(*a*,*b*) is denoted as $P_{e(a,b)}$. The new shortest path *p* must either be an edge connecting *u* and *v* or contain some nodes which vary from the nodes contained in $P_{e(a,b)}$. For the first case, based on the local graph connection information (if *u* and *v* are in the same sub-graph) or outside distance table (if *u* and *v* belong to different sub-graphs), the correct geodesic distance can be found at the step 6 (if *u* and *v* are in the same sub-graph) or step 3 (if u and v belong to different sub-graphs) in Algorithm IV. For the second case, assume node *z* is one of the nodes which lie on the new path *p* and is not contained in $P_{e(a,b)}$. Denote a shortest path connecting node pair (*u*, *z*) as $p_{(u,z)}$. Then, we can get that $p_{(u,z)}$ does not go through edge *e*(*a*,*b*). Otherwise, node *z* will be contained in $P_{e(a,b)}$. It is the same for $p_{(z,v)}$, a shortest path connecting node pair (z,v). This means that

the node pairs ($u$,$z$) and ($z$,$v$) are not PAPs of the dynamic edge $e$($a$,$b$). During the dynamic change adoption, the lengths of $p$($u$,$z$) and $p$($z$,$v$) keeps unchanged. When $z$ is in the same sub-graph as $u$, this new shortest path can be found at the overestimation local correction step in Algorithm IV. If $z$ and $u$ are not in the same sub-graph, the geodesic distance information of node z will arrive to the processor which contains u at some stage and the shortest path can be identified at the step 3 in Algorithm IV in that stage.

As we mentioned in part b, several dynamic changes may be grouped together to be handled. This will not affect the validity of our dynamic adoption approach. During dynamic change adoption, geodesic distances between pairs of nodes may be overestimated. However, the graph connection information is kept consistent with the true graph. Even with overlapped overestimations, the overestimated values can be corrected based on graph connection information.

**e) Anywhere Property of the Approach**

In this manuscript, the term “anywhere” mainly represents the idea of graph’s dynamic information adoption. When an edge weight is decreased, each processor can update their local information by formula (6.6) to directly adopt this change. When an edge weight is increased, the upper bounds of the new geodesic distances are first calculated by the overestimation (6.8). Then, each processor tries to refine these overestimations by local and external information and propagate the refinements through the whole network as a ripple-effect. Our program can effectively adopt both types of dynamic change based on obtained results during its analysis process. Thus, we say that our closeness centrality recombination algorithm is an anywhere approach.

**f) Approach’s Performance:**

In the following paragraphs, we will analyze the time cost for our anywhere recombination algorithm for closeness centrality measurement to adopt edge weight changes. When an edge weight is decreased, the program will update all

elements in local nodes' DVs by formula (6.6). The cost for this is $O(n \cdot \max|V_i|)$. Since $|V_i| \approx n/p$ (as we discussed in the analysis of anytime approach), this bound can be formed as: $O(n^2/p)$ where $p$ is the number of processors used in our system.

When the weight of an edge, $e(a,b)$, is increased, a simple lower bound of the work required to adopt this change will be the number of shortest paths affected by this edge. We define the *edge betweenness* as the number of pairs of nodes which have at least one shortest path going through this edge, and we use $bt$ to represent this value. In other words, $bt$ represents the number of PAPs of $e(a,b)$. Most work for adopting a dynamic edge with an increased weight is done in steps 2, 3, 5, and 6 of Algorithm IV. The work can be classified into two types. Steps 5 and 6 are performed only at the first time when a change happens. For step 5, the work is bounded by identifying PAPs, which requires $O(n \cdot |V_i|)$. In step 6, we will try to refine every overestimated distance by local information. The work load is $O(bt_i \cdot |V_i|)$. Thus, the bound of work load in step 5 and 6 is

$$O(bt_i \cdot |V_i| + n \cdot |V_i|)$$

Steps 2 and 3 are another type of work and used for updating local results and propagating new information. They are performed at each stage until the overestimated geodesic distances are converged. For step 2, as we discussed in anytime approach, its work load is $O(n \cdot |C_i| \cdot \gamma_i)$. In step 3, we will check every overestimated PAP to see if there any improvement based on the new information. The work load for this step is $O(bt_i \cdot |C_i|)$. The number of stages required for the fully correction of overestimated distances is the diameter of the super-graph obtained in the DD phase, which is less or equal to $p$-1. The total work load for steps 2 and 3 is

$$O(p \cdot n \cdot |C_i| \cdot \gamma_i + p \cdot bt \cdot |C_i|)$$

Thus, the total work for adopting an edge with increased weight is

$$O(p \cdot n \cdot |C_i| \cdot \gamma_i + p \cdot bt \cdot |C_i| + bt_i \cdot |V_i| + n \cdot |V_i|)$$

This formula shows that the work for dynamic edge adoption is determined by how well the graph is decomposed and the betweenness value of this edge. As $|C_i|$ is bounded by $n/p$ (discussed in anytime approach), the work load can be transformed into:

$$O(n^2 \cdot \gamma_i + bt \cdot n + n^2 / p) \tag{6.9}$$

Recall that in the anytime approach analysis we have shown that $\gamma_i$ can be bounded as $n/\log n$ in real-world large social networks. Also, we know that, except outside distance table, there are at most $n \cdot |V_i|$ distances are maintained in each processor which means that $bt_i \le n^2 / k$. Therefore, from formula (6.9), we can see that when the edge betweenness is low and graph is not well decomposed, the work load is mainly determined by the graph domain decomposition. When this dynamic edge affects a lot of shortest paths, the work load it affected by this edge's betweenness value. When the graph is well decomposed and the dynamic edge only stays on a few shortest paths, the most work is done in identifying the PAPs of this edge (which is represented by the last term in formula (6.9)). For the worst case, the work load can be

$$O\left(\frac{n^3}{p} + \frac{n^3}{\log n}\right)$$

which is the same as calculating the geodesic distances for all pairs from scratch. In this case, the graph is badly composed and the dynamic edge affects shortest paths between all pairs of nodes. However, for average case, it will only require relatively short period of time for our approach to adopt the dynamic information. In order to accelerate the dynamic adoption for the worst case, we need to store all shortest paths. This will require a formidable amount of memory which is $O(n^3)$ and seems to be impractical when social networks are large.

## *6.6 The Anytime Anywhere Recombination Approach for Closeness Centrality*

In both the anytime approach and the dynamic adoption (anywhere) approach for closeness centrality recombination, the new information obtained at each processor is propagated through the whole network by a ripple-effect. The number of stages required for fully adopting one dynamic change is the limited by the diameter of the super-graph obtained after the DD phase. We can keep track of dynamic changes in a queue. Old changes, which have been fully adopted, will be removed from the queue and new changes, which come in the current stage, will be added. When the queue is empty, the final true results for the graph with all dynamic changes are obtained.

From the design of Algorithm IV, we can see that the dynamic adoption approach couples with the anytime approach tightly. They both work based on updating local results by external information and propagate local new results as a ripple-effect. In fact, dynamic change adoption can be easily incorporated into the anytime approach paradigm. Combining these two algorithms together, the flowchart of our anytime anywhere recombination algorithm for closeness centrality measurement is shown in Figure 6-9.

1. Initialize the RC phase.

2. Check the convergence of the anytime anywhere algorithm: if *converged*, then terminate the program. Otherwise, go to next step.

3. Identify and propagate new information

4. Gather all related external new information & update outside distance table and boundary nodes DVs.

5. If the anytime approach is *converged*, go to step 7. Otherwise, go to next step

6. Update local Information I: update geodesic distances for all local nodes based on the new information contained in the outside distance table. Then, go to step 8

7. Update local Information II: update un-converged PAPs' geodesic distances based on the new information contained in the outside distance table.

8. Gather all dynamic edges' information

9. Dynamic change local adoption

10. Overestimation local correction

11. Synchronization: wait until all processors finish step 10, then go back to step 2.

**Figure 6-9.** Algorithm V: the anytime anywhere recombination algorithm for closeness centrality measurement

In our anytime-anywhere approach for closeness centrality measurement, the nearly-seamless coupling of partial results calculation and dynamic information adoption imbues the program with the ability to adapt dynamic edge changes with obtained partial results. This will potentially decrease the work load for adopting graph's dynamism.

## *6.7 Maximal Clique Enumeration Recombination Algorithm I – The Anytime Approach*

Enumerating all maximal cliques contained in a graph is an NP-hard problem. All known algorithms for solving this problem have both computation and storage costs on an exponential order of the graph size. Our anytime approach for maximal clique enumeration is the similar as Zhang's algorithm introduced in section 4.6. Here, we include the pseudo-code of this algorithm in Figure 6-10. In this algorithm, all elements in a clique (both maximal clique and candidate clique) are stored in canonical order. The anytime property of this approach is demonstrated by the fact that the algorithm provides maximal cliques in an increasing order of clique size. The validity of this algorithm can be found in [ZhangABCLS'05].

1. Initialization:
    obtain all 2-cliques
    generate maximal 2-cliques, and candidate 2-cliques
    set clique size $k$=2
2. while the set of candidate $k$-cliques is not empty
3.  while the set of candidate $k$-cliques is not empty
4.  pickup a candidate k-clique $C_i$
5.  while common neighbor of $C_i$ is not empty
6.  get $C_i$ common neighbor $a$, where a is larger than the last element in $C_i$
7.  expand $C_i$ with a: { $C_i$ , a}, get all its common neighbor $A$
8.  if A is empty
9.  put { $C_i$ ,a} into maximal ($k$+1)-cliques.
10.  else if there is one element of $A$ is larger than $a$
11.  put { $C_i$ ,a} together with $A$ into candidate ($k$+1)-cliques
12.  remove $a$ from $C_i$ common neighbor set
13.  remove $C_i$
14.  $k$++

**Figure 6-10.** Algorithm VI: the anytime recombination algorithm for maximal clique enumeration.

## *6.8 Maximal Clique Enumeration Recombination Algorithm II – The Anywhere Approach*

For maximal clique enumeration problem, there are two types graph dynamisms: edge addition and edge removal. Although the essential ideas for handling these two types of dynamic changes are the same, there are some differences between the anywhere algorithms for added edges and removed edges. In following paragraphs, we will introduce them respectively.

**a) The Anywhere Approach for Edge Addition**

Assume after we obtain all *k*-cliques (both maximal *k*-cliques and candidate *k*-cliques) there is an edge *e*(*u*,*v*) added in the graph.

*Effect Range*: It is easy to see that if a clique (either a maximal clique or a candidate clique) includes neither *u* nor *v*, it is not affected by this edge change. Only those cliques which contain either *u* or *v* may be affected by this edge addition. There is no cliques contains both *u* and *v* since there is no edge between them in the original graph.

*Effect Ways*: If a maximal clique contains *u* (or *v*) and *v* (or *u*) is a common neighbor of this clique after the edge addition, this clique will not be maximal anymore. It should be deleted from the maximal clique set. If a candidate clique contains *u* (or *v*) and *v* becomes its common neighbor after the edge addition, this clique is still a candidate clique. However, its common neighbor should be updated.

*Recalculation*: When an edge *e*(*u*,*v*) is inserted, there may be new cliques (both maximal and candidate) that contain both *u* and *v*. Some of these cliques may be generated by an expansion of obtained cliques, but not all of them can be handled in this way. We use Figure 6-11 as an example to demonstrate why this happens.

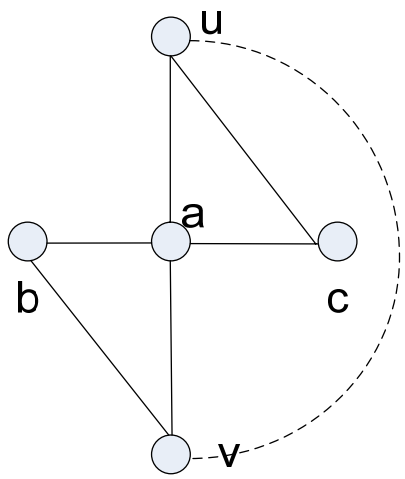



**Figure 6-11.** Example for finding maximal cliques with edge addition

In this example, after running the maximal clique enumeration algorithm, we will obtain 2 maximal 3-cliques, {u, a, c} and {a, b, v}. When the edge $e(u,v)$ is added, there is a new maximal 3-clique {u, a, v}. However, this maximal clique cannot be generated from the obtained results. This is because that neither {a, v} nor {u, a} is a maximal clique. These two cliques belong to candidate 2-cliques. However, since we only maintain the candidate cliques with the largest clique size, these two candidate cliques are eliminated in the iteration for finding 3-cliques.

In order to incorporate an added edge $e(u,v)$ at the $k$th iteration in Algorithm VI, we need to re-enumerate all cliques which contain both $u$ and $v$, up to size $k$. The recalculation for maximal clique enumeration should perform the following three tasks:

1. update the common neighbor set of those candidate $k$-cliques which contain node $u$ or $v$
2. generate all maximal cliques containing both $u$ and $v$ up to size $k$
3. generate all candidate $k$-cliques whose clique sets contain both $u$ and $v$

*The Approach*: When a new edge $e(u,v)$ is added, the anywhere approach for adopting this change for maximal clique enumeration is shown in Figure 6-12. In

this figure, $k$ represents the largest size of obtained cliques and $cn(A)$ is the set of common neighbors of clique $A$.

1. Initialization:
   Assume $u>v$, add edge $e(u,v)$
   get all maximal cliques which contain either $u$ or $v$ into a queue $Q_m$
   get all candidate $k$-cliques which contain either $u$ or $v$ into a queue $Q_c$
2. for every element $M$ in $Q_m$
3. if $u$ is not contained in $M$ but is a common neighbor of $M$
4. remove $M$ from the maximal clique set
5. if $u>\max(M)$ & $\text{size}(M)=k$
6. put $\{M|u\}$ into candidate $k$-clique set
7. if $v$ is not contained in $M$ but is a common neighbor of $M$
8. remove $M$ from the maximal clique set
9. for every element $C$ in $Q_c$
10. if $u$ is contained in $C$ and $v$ is a common neighbor of $C$
11. add $v$ into $C$'s common neighbor set
12. if $v$ is contained in $C$ and $u$ is a common neighbor of $C$
13. add $u$ into $C$'s common neighbor set
14. build a candidate 2-clique $E$ as {u, v | cn({u, v})}
15. enumerate all cliques expanded from $E$ up to size $k$, put maximal cliques and candidate $k$-cliques into corresponding sets.

**Figure 6-12.** Algorithm VII: the anywhere recombination algorithm for added edge for maximal clique enumeration.

Maximal clique enumeration is an NP-hard problem. In the worst case, the number of maximal cliques in a graph is an exponential order of the graph size. It is infeasible to store all maximal cliques in memory. We must either store them in files or in a database. Storing cliques in files will make it hard to find related cliques in graph dynamic change adoption. In order to easily index cliques, we use the MySQL database to maintain all maximal cliques and the candidate cliques with the largest clique size.

**b) The Anywhere Approach for Edge Removal**

Adopting a removed edge can be handled in a similar way as the edge addition case. Assume after we obtain all *k*-cliques (both maximal *k*-cliques and candidate *k*-cliques) there is an edge *e*(*u*,*v*) removed from the graph. For maximal cliques, if it does not contain *u* or *v*, it will not be affected. In other words, those maximal cliques which contains both u and v will become invalid if we remove edge *e*(*u*,*v*). Assume a maximal clique *M* is one of this type's cliques. If we remove *u* from *M*, it will form a new clique, {*M/u*}. This clique is not certainly maximal. We need to check if it has any common neighbor. If it has, this clique has already been contained in other maximal cliques or candidate cliques. We do not need to do anything. If not, this is a new maximal clique and needs to be added in the corresponding maximal clique set. It is the same way to handle the clique {*M*/*v*}.

For candidate *k*-cliques, the process is different. According to the removed edge *e*(*u*,*v*), there are three types of candidate *k*-cliques, type I: containing neither *u* nor *v*; type II: containing either *u* or *v* but not both; and type III: containing both *u* and *v*. After the edge removal, *u*'s and *v*'s connections to other vertices except *v* or *u* are unchanged. For type I candidate *k*-cliques, they are not affected. If a candidate *k*-clique contains only one of vertices *u* and *v*, say *u*, it may be affected if its common neighbor set contains the other one, say *v*. For this type of candidate k-clique, we need to update its common neighbor set by removing vertex *v*. After the updating, the candidate clique may become a new maximal clique (empty common neighbor set), a new candidate clique, or invalid (common neighbor set is not empty but the clique cannot be expanded according to canonical order). For type III candidate cliques, we first remove one endpoints of the removed edge, say *u*, from the clique set, getting {*M*/*u*}. Then, we generate this new clique's common neighbor set *cn*({*M*/*u*}). Expanding {*M*/*u*} with its common neighbor, we can obtain either a new maximal *k*-clique or candidate *k*-clique. For obtained new maximal cliques, we put them into the maximal *k*-clique set. For generated new candidate *k*-cliques, we just discard them since they have already been generated in the process for type II candidate cliques.

The pseudo-code of the anywhere recombination approach for adopting removed edges for maximal clique enumeration is shown in Figure 6-13. In this figure, $k$ represents the largest size of obtained cliques and $cn(A)$ is the set of common neighbors of clique $A$.

```
1. Initialization:
        Assume u>v, remove edge e(u,v)
        get all maximal cliques which contain both u and v into a queue Q_m
        get all candidate k-cliques which contain either u or v into a queue Q_c
2. for every element M in Q_m
3.      remove M from maximal clique set
4.      if {M/u} (or {M/v}) is maximal
5.              put {M/u} (or {M/v}) into the corresponding maximal clique set
6. for every element C in Q_c
7.      if C contains u but not v (or contains v but not u)
8.              update C's common neighbor set
9.      if C contains both u and v
10.             get cn({C/u}), and cn({C/v})
11.             expand {C/u} and {C/v} by one common neighbor respectively
12.             put obtained maximal cliques into maximal k-clique set.
```

**Figure 6-13.** Algorithm VIII: the anywhere recombination algorithm for added edge for maximal clique enumeration.

### c) The Fully Dynamic Anywhere Approach

Combining the two approaches discussed before, our anywhere recombination approach for both edge addition and removal for maximal clique enumeration is shown in Figure 6-14.

1. Initialization:
    get all edge changes into a queue *E*
    get all maximal cliques up to size *k*
    get all candidate *k*-cliques
2. while *E* is not empty
3.  get the first element *e*(*u*,*v*) from *E*
4.  if *e*(*u*,*v*) is edge addition
5.  call Algorithm VII
6.  else if *e*(*u*,*v*) is edge removal
7.  call Algorithm VIII
8.  remove *e*(*u*,*v*) from *E*.

**Figure 6-14.** Algorithm IX: the fully dynamic anywhere algorithm for maximal clique enumeration.

**d) Approach Performance:**

For edge addition, we use Algorithm VII to adopt graph's dynamic changes. Assume edge *e*(*u*,*v*) is inserted in the graph when our algorithm obtains all cliques up to size *k*. The work for adopting this edge is mainly done in updating obtained cliques containing either *u* or *v* (steps 2 to 13 in Algorithm VII) and enumerating cliques including both *u* and *v* (steps 14 and 15).

Use $M_x(y)$ to represent the number of maximal cliques with size *y* and containing vertex *x*, and $\Phi x(y)$ to represent the number of candidate cliques with size *y* and containing vertex *x*. Define

$$M_x^y = \sum_{i=1}^{y} M_x(i) \quad and \quad \Phi_x^y = \sum_{i=1}^{y} \Phi_x(i)$$

Checking if a clique has a common neighbor of vertex *i* will cost O(*Δ*), where *Δ* is the maximum degree. Updating obtained cliques will cost

$$O\left((M_u^k + \Phi_u(k) + M_v^k + \Phi_v(k)) \cdot \Delta\right)$$

The work load to enumerate all cliques (up to clique size *k*) including both *u* and *v* will have time cost as

$$O\left((M_{u,v}^{k} + \Phi_{u,v}^{k}) \cdot \Delta\right)$$

Thus, the total work for adopting added edge $e(u,v)$ is

$$O\left((M_{u}^{k} + \Phi_{u}(k) + M_{v}^{k} + \Phi_{v}(k) + M_{u,v}^{k} + \Phi_{u,v}^{k}) \cdot \Delta\right)$$

When an edge is removed, we will employ Algorithm VIII to handle it. Assume edge $e(u,v)$ is deleted at the time our algorithm obtains all cliques up to size $k$. The main work is done in updating obtained maximal cliques (steps 2 to 5) and candidate $k$-cliques (steps 6 to 12). Updating maximal cliques will have time cost as:

$$O\left(M_{u,v}^{k} \cdot k \cdot \Delta\right)$$

Updating candidate cliques will have time cost as:

$$O\left((\Phi_{u}(k) + \Phi_{v}(k)) \cdot \Delta^{2}\right)$$

Thus, the total time for adopting a removed edge is:

$$O\left(M_{u,v}^{k} \cdot k \cdot \Delta + (\Phi_{u}(k) + \Phi_{v}(k)) \cdot \Delta^{2}\right)$$

In the worst case, the anywhere approach will have the time cost as an exponential order of the graph size. However for average case, usually we only need to handle a small sub-part of the original problem. It will take relatively small amount of time for our approach to adopt the dynamic information.

## *6.9 The Anytime Anywhere Recombination Approach for Maximal Clique Enumeration*

From the design of our algorithms we can see that the anytime approach and the anywhere approach for maximal clique enumeration can seamlessly corporate together. Within the anytime processing of finding maximal cliques, we can naturally adopt graph's dynamic changes by our anywhere approach based on the obtained partial results. The anytime anywhere approach for maximal clique enumeration is shown in Figure 6-15.

1. Initialization:
    obtain all 2-cliques
    generate maximal 2-cliques, and candidate 2-cliques
    set clique size $k$=2
2. while the set of candidate $k$-cliques is not empty
3.  while the set of candidate $k$-cliques is not empty
4.  expand each element $C_i$ according to Algorithm VI
5.  remove $C_i$
6.  gather all happened dynamic edge changes into queue $E$
7.  for each edge change $e(u,v)$ in $E$, call Algorithm IX
8.  empty $E$
9.  $k$++
10. gather all happened dynamic edge changes into queue $E$
11. for each edge change $e(u,v)$ in $E$, call Algorithm IX

**Figure 6-15.** Algorithm X: the anytime anywhere recombination algorithm for maximal clique enumeration

## *6.10 Summary*

In this chapter, we provide detailed discussion about the design and implementation of the Recombination phase for the three SNA metrics chosen for evaluating and validating our methodology. Among these three SNA metrics, Ego-betweenness centrality is the simplest. It can be directly measured within a short period of time. Thus, there is no need for the anytime approach for its measurement. We only design and implement an anywhere approach for this metric to effectively handle graph's dynamic changes. For closeness centrality and maximal clique enumeration, we present the design and theoretical performance analyses for their anytime and anywhere recombination algorithms. For these two metrics, solutions are partially and incrementally built. During the

analysis process, if some dynamic changes happen, our algorithm can effectively and efficiently incorporate these changes into the analysis process based on the obtained results. Both the anytime and the anywhere properties of our methodology are well demonstrated in recombination algorithms of these two metrics.

# 7. Experimental Results and Analysis

In previous chapters, we present details about the design, implementation, and theoretical performance analyses of our anytime-anywhere methodology for large and dynamic social network analysis. In order to further evaluate and validate our methodology for SNA applications, we decide to test our methodology through a set of experiments on the selected SNA metrics, ego-betweenness centrality, closeness centrality, and maximal clique enumeration.

## *7.1 Experiments Setup*

Our anytime anywhere methodology is implemented on a cluster of processors running a version of the Linux operating system. Each machine has 512MB of memory and an Intel Pentium 2.66GHz processor. The machines are connected by a gigabit network backbone.

We used Pajek [BatageljM'04] to generate a series of connected random graphs of size from 5,000 to 30,000 in increasing size with increments of about 5000. For each size, we generate graphs with 3 different types of density:

- Density I: average degree = 4
- Density II: average degree = 8
- Density III: average degree = 16

For each graph, we use our methodology to measure ego-betweenness centrality and closeness centrality and enumerate all maximal cliques by employing 4, 6, and 8 processors respectively. In order to further validate our methodology, we generate another set of random graphs with Density II and test our system on these graphs. In other words, we test the implementation of our methodology on

one set of random graphs with Density I and III respectively, and two sets of random graphs with Density II.

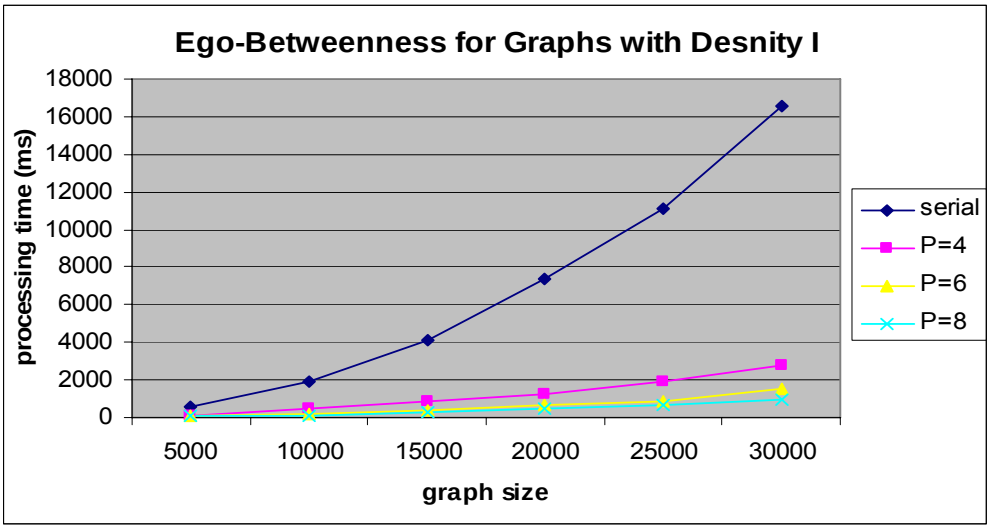
In order to study our methodology's performance on processing dynamic graphs, we generate a set of synthetic random edge changes for each graph and test our system's ability of handling random dynamic changes. As we all know that a system's performance on adopting dynamic changes is affected not only by how many changes happen and when they happen, but also by where (on what types of nodes) they happen. Node degree is an important parameter that should be considered in dynamic change adoption due to two main reasons. First, normally, the more edges a node has the higher chance that a dynamic edge change happens on this node. Second, in many cases, node degree may affect the work load for handling dynamic changes. In order to comprehensively analyze our methodology, we generate and test our system by another set of edge changes which happen on vertices with highest degrees.

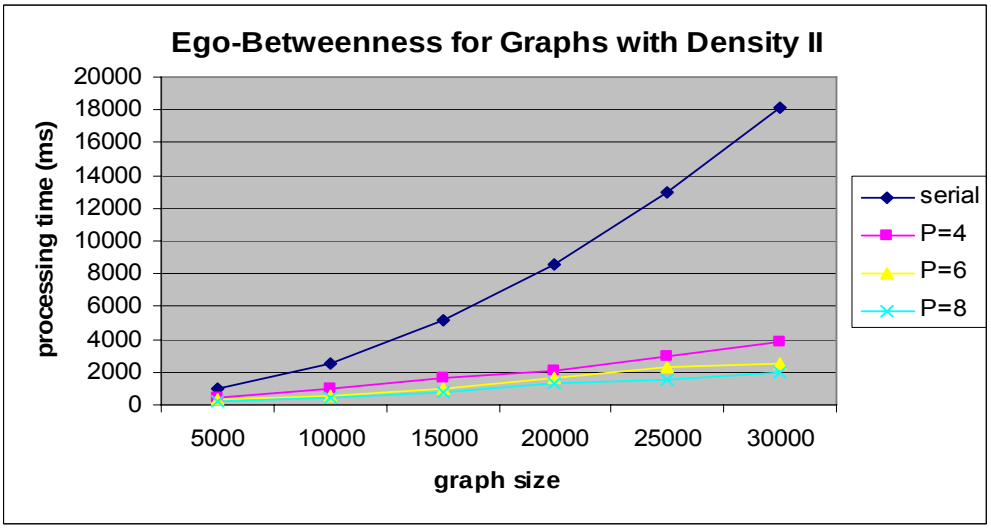
## *7.2 Experiments on Ego-Betweenness Centrality Measurement*

As we mentioned, measuring ego-betweenness centrality is simple and fast. We choose it as a sanity test case for primarily checking if there is any flaw in the design. To validate our methodology, we want to fairly compare the performance of our approach with the performance of current SNA software tools. Most, if not all, current SNA software tools generate ego-betweenness centrality only for unweighted graphs. However our approach focuses on more general graphs which have positive real edge weights. Therefore, we implement a serial algorithm (modified Dijkstra's Algorithm which is introduced in chapter 4.5) which can handle weighted graphs and has similar performance as the algorithm employed in current SNA tools. In our experiments, we try to evaluate our approach by comparing with this serial algorithm.

### a) Serial Algorithm vs. Our Parallel Approach

In this experiment, we study our approach's ability for measuring ego-betweenness centrality for static graphs. In our methodology, we decomposed the graph into 4, 6, and 8 parts respectively. After the decomposition, each part is sent to a single processor and the ego-betweenness centrality is measured locally at each processor. The comparison of our system and serial algorithm based on the running time is shown in Figure 7-1.



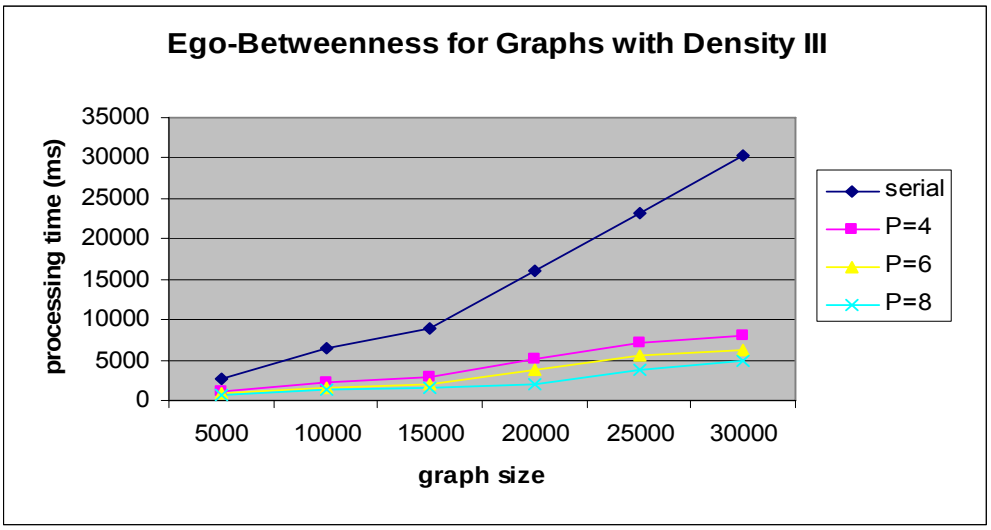



**Figure 7-1.** Ego-betweenness centrality measurement: serial vs. parallel[1].

Figure 7-1 shows that our system can obtain a good speed-up for the ego-betweenness centrality measurement for graphs with all three types of densities. The speed-up of using 4 processors is slightly larger than 4. This is because that in our system, each processor only focuses on a small problem (the local sub-graph). However, the serial algorithm deals the original large network. It introduces additional overheads such as reading discontinuous memories.

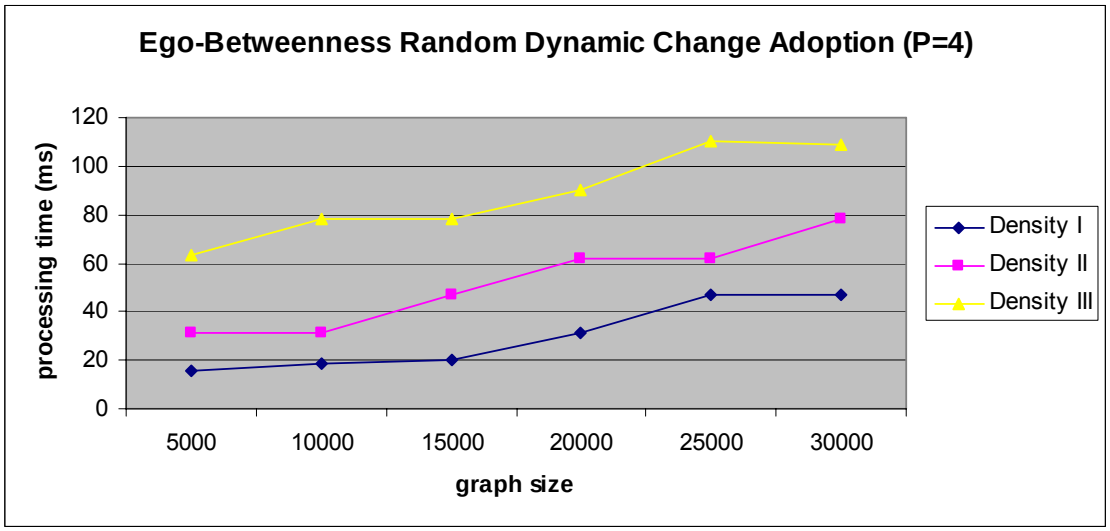
In Figure 7-1, we do not include the cost for graph decomposition. Due to the simplicity of ego-betweenness centrality, the overhead of partitioning graph takes even longer time than serially measuring all ego-betweenness centralities. It seems that the DD phase of our methodology is not useful if we only calculate ego-betweenness centralities. However, our methodology is proposed for comprehensively analyzing large social networks with a broad range of applications. For many complicated but significant SNA metrics, such as closeness centrality, in order to effectively and efficiently analyze them, an effective graph decomposition mechanism is necessary. This will be shown in our experimental results and analyses of closeness centrality measurement.

[1] Notes: in this section, n represent the graph size and P represents the number of processors used in our system to perform network analysis

**b) Anywhere Property - Dynamic Changes Adoption**

We reiterate that there is no need for implementing an anytime approach for ego-betweenness centrality measurement. In our experiment, we study the performance of our anywhere approach for ego-betweenness centrality. Our approach's performance on dynamic change adoption is shown in Figure 7-2, 7-3, and 7-4.

Figure 7-2 shows the time costs of our system to incorporate 64 random changes on edge weights. Intuitively, as graph size increases, the processing time for adopting these dynamic changes will increase. As graph density increases, directly connected nodes may have more common neighbors. Thus, the cost for adopting dynamic changes may also increase. From Figure 7-2, we can see that our system performs exactly the same as what we hypothesized.



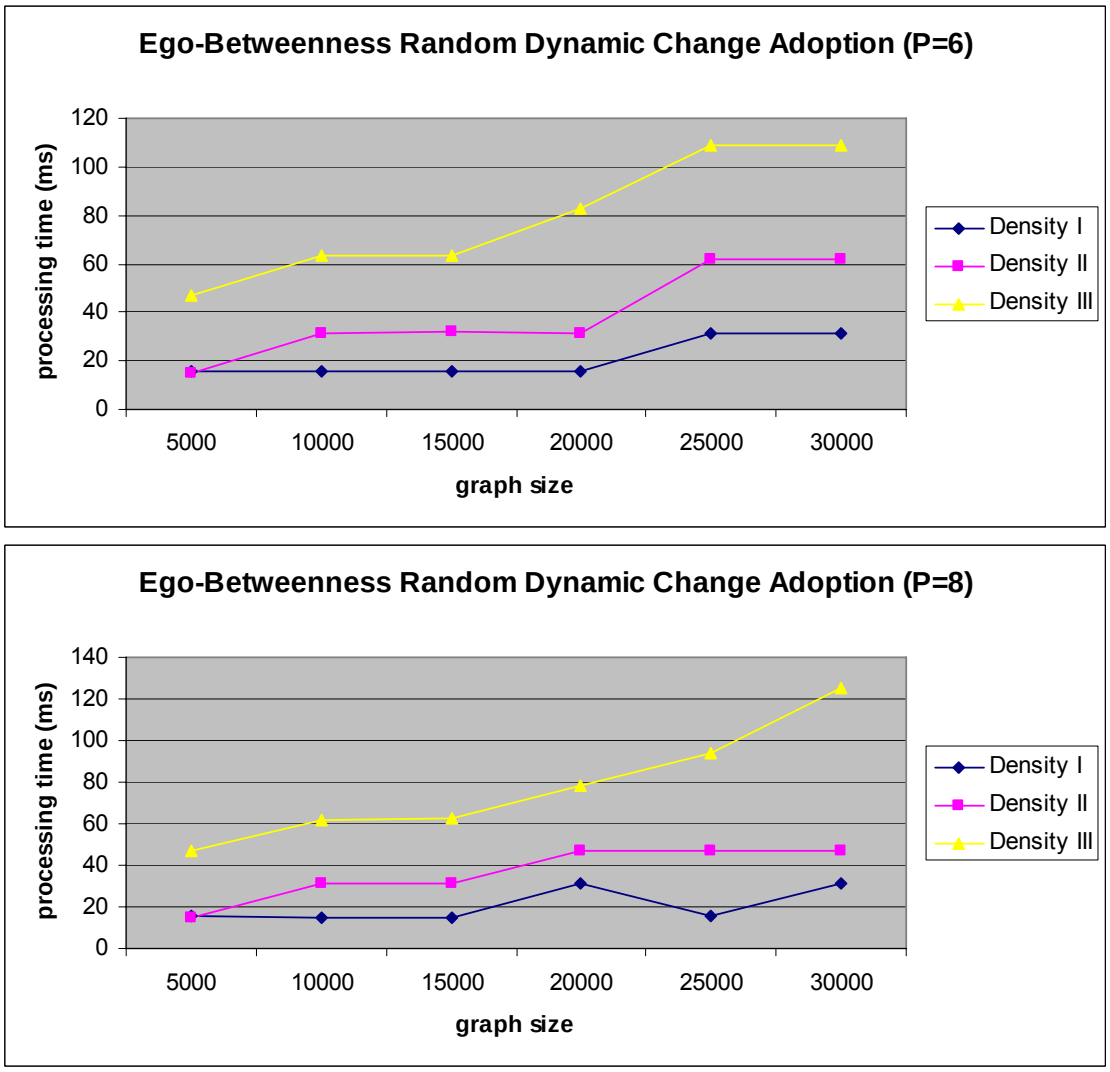



**Figure 7-2.** Time cost for adopting 64 random edge changes for ego-betweenness centrality measurement.

As we mentioned, in order to further study our system's ability of handling graph's dynamism, we also generate a set of dynamic edge changes on nodes with highest degrees. We call this set of dynamic changes as *max degree changes*. The comparison of our system's performance of adopting random dynamic changes vs. performance of adopting max degree changes is shown in Figure 7-3. In this figure, we only present the time costs for our system with 4 processors. The corresponding comparison for 6 (and 8) processors is similar as Figure 7-3. Thus, it is not presented here.

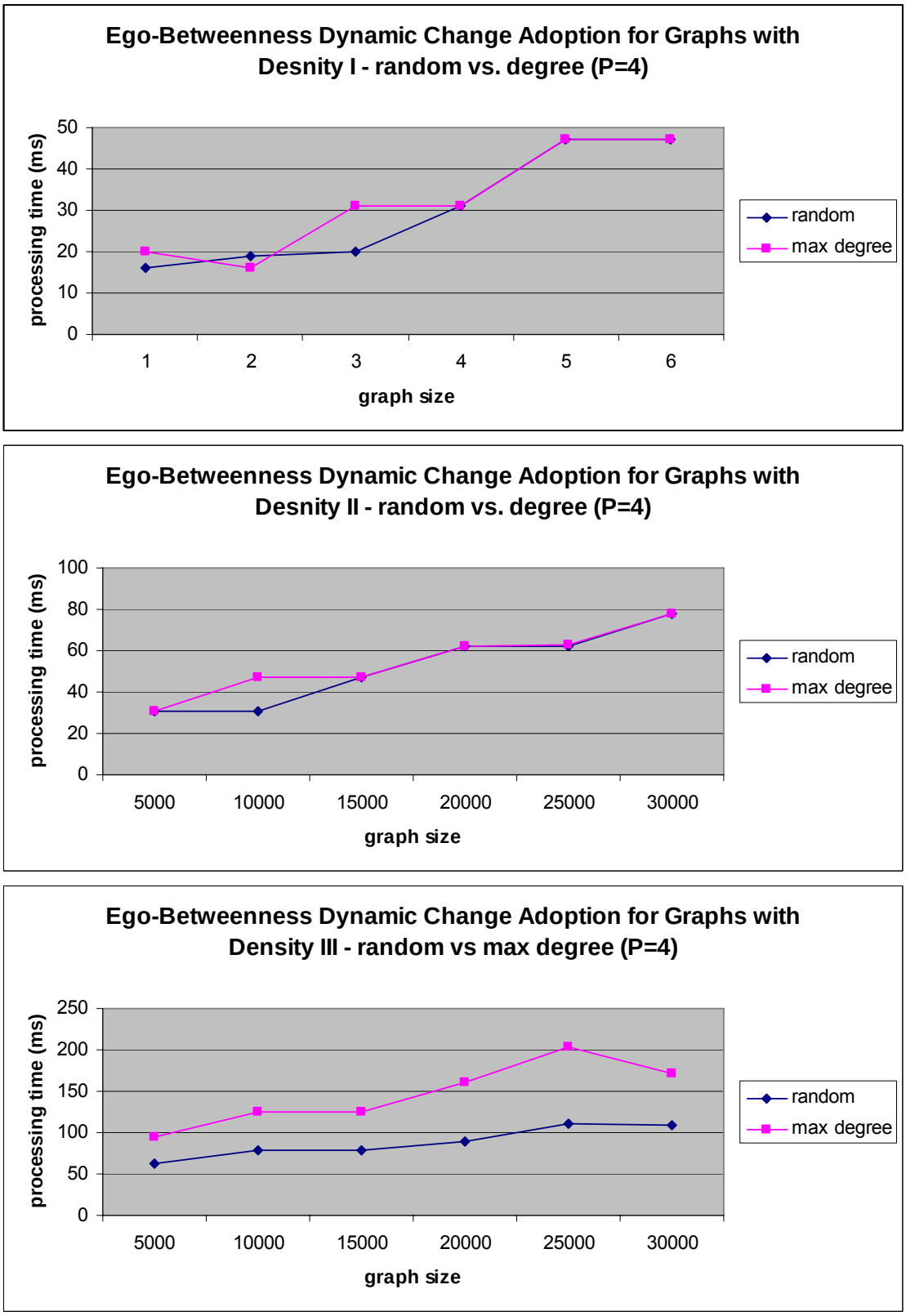



**Figure 7-3.** Performance comparison between handling random changes and handling max degree changes for ego-betweenness centrality measurement.

Figure 7-3 shows that for graphs with Density I and II, our approach performs similarly for random changes and max degree changes. For graphs with Density III, it takes more time for our approach to adopt max degree changes than random changes. In the theoretical analysis of anywhere approach for ego-betweenness centrality measurement we obtained that the work load for adopting an edge change is not determined by the degree of this edge's endpoints, rather depending on the number of their common neighbors. Basically, the higher endpoints node degrees the more potential to obtain large common neighbor sets. This is well demonstrated in the system performance comparison on graphs with Density III. For graphs with Density I & II, the time cost for adopting max degree changes is slightly larger than the cost for adopting random changes. This is because that these two types of graphs have low densities. Even dynamic change happens on endpoints with high degree, there is still little chance for these dynamic edge's endpoint to have large number of common neighbors.

The absolute value of time cost for dynamic edge change adoption is an important factor to evaluate the system performance. However, as shown in Figure 7-2, graph size and density affect time cost. For many cases, it is more useful to how our approach performs relatively to graph size and density. Relative costs for dynamic change adoption for all graphs are shown in Figure 7-4. Each relative cost is calculated as:

$$\mu = \max(r, c) / C$$

where $\mu$ is the relative cost, $r$ is the time cost for adopting random changes, $c$ is the time cost for adopting max degree changes, $C$ is the time cost for calculating each node's ego-betweenness centrality for static graphs by using the serial algorithm. In Figure 7-4 we can see that for types of graph densities although the absolute time cost for adopting dynamic changes grows as graph size increases, the relative cost is decreasing. This means that when the graph size becomes larger and larger, the portion of affected obtained results becomes smaller and smaller. The maximum relative cost for adopting one edge change is about

0.055%. Thus, we can tell that our methodology for ego-betweenness centrality measurement can efficiently handle graph's dynamism.

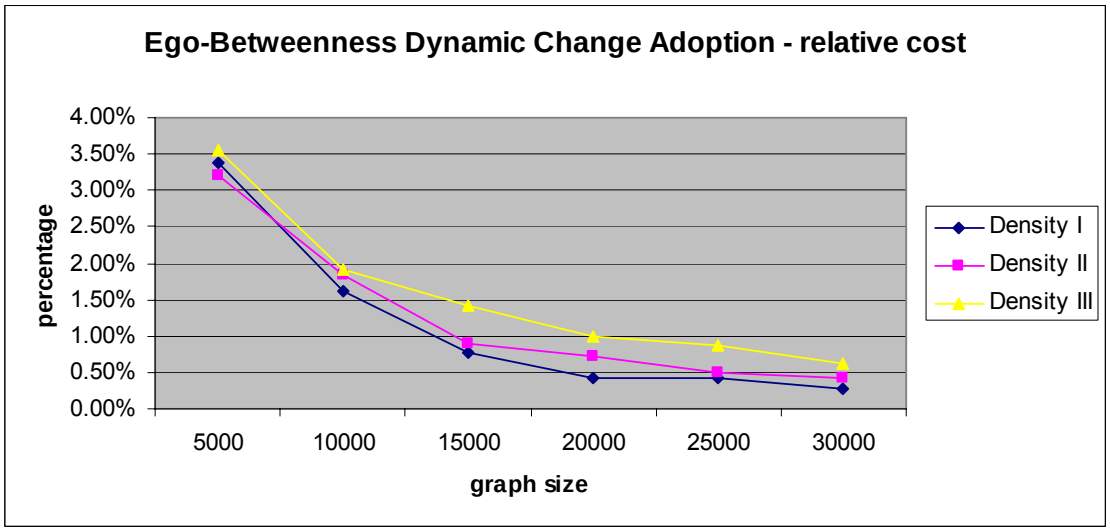



**Figure 7-4.** Relative cost for adopting 64 edge changes for ego-betweenness centrality measurement

## *7.3 Experiments on Closeness Centrality Measurement*

Closeness centrality is measured based on all-pairs shortest paths. Usually, algorithms for this problem require about $O(n^3)$ running time and $O(n^2)$ storage space. Currently software tools cannot provide closeness centrality analysis for large social networks [SantosPAXP'06]. Moreover, these SNA tools work like black-boxes. We do not know how algorithms are implemented in them. There is no way for us to directly compare our approach with the one employed in current SNA tools. Thus, we implement our serial algorithm (Dijkstra's algorithm introduced in chapter 4.4) to compare with. We first study how our serial algorithm performs relatively to popular current SNA tools' performance. Based on this study, the comparison of our anytime anywhere approach to current SNA tools can be inferred from the comparison between our methodology and our serial algorithm.

### a) Comparison between Current SNA Tools and Our Serial Algorithm

UCINet [BorgattiEF'02] is one of the most popular SNA software tools which can provide comprehensive analysis on small-scale interactions. In this experiment, we compare our serial approach with UCINet.by testing them on a set of random graphs with sizes from 500 to 12000 since UCINet program crashes on graphs larger than 13000 nodes [SantosPAXP'06]. We run our serial algorithm and UCINet program respectively to measure the closeness centralities for all nodes contained in the graph. The running time comparison of our serial algorithm to UCINet's algorithm is shown in Figure 7-5. This figure shows that though UCINet outperforms our algorithm, asymptotically they both appear to have the same complexity, and nearly the same runtime. For large social networks, we believe our serial algorithm performs similar as the algorithm employed in UCINet.

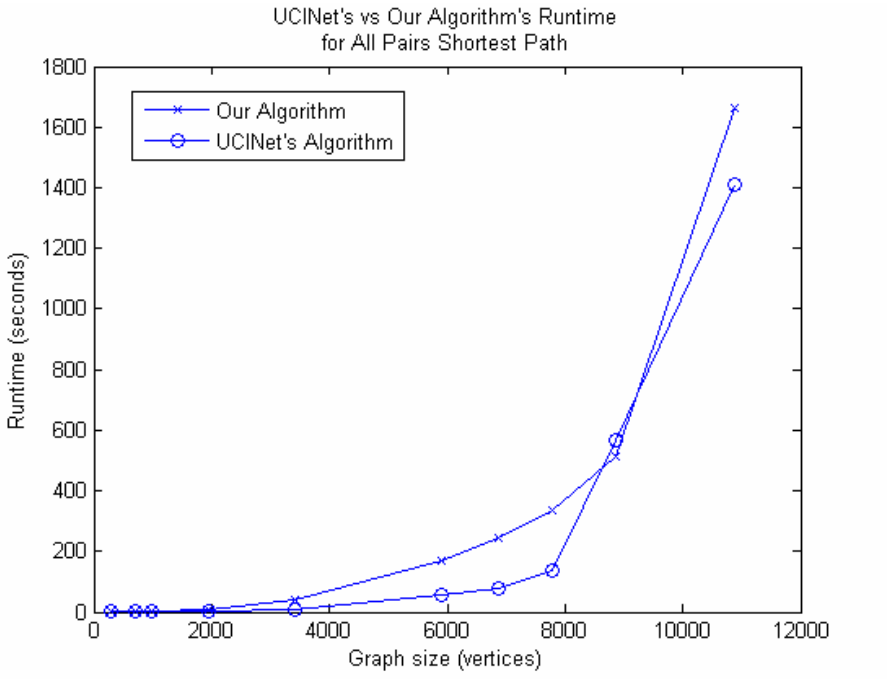
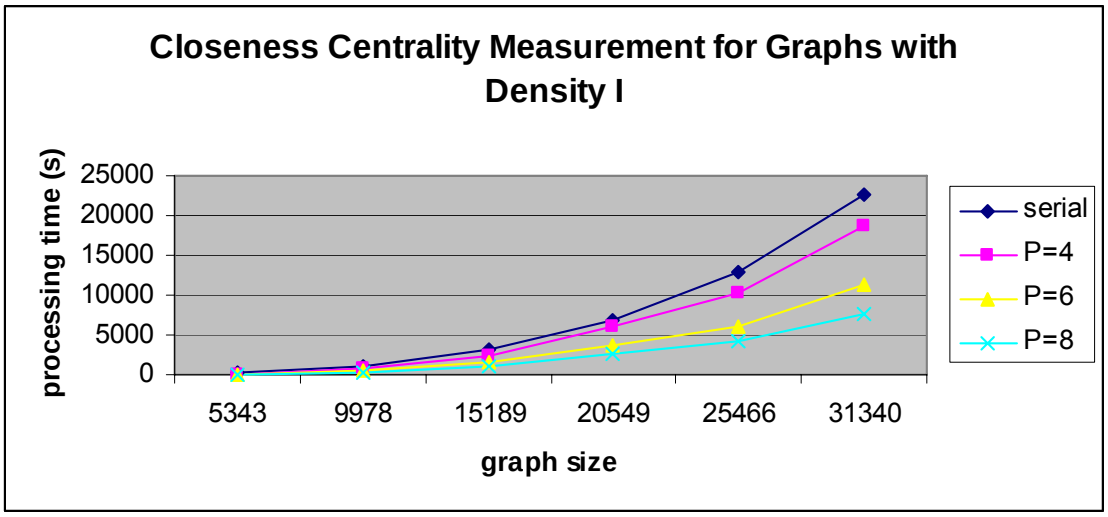
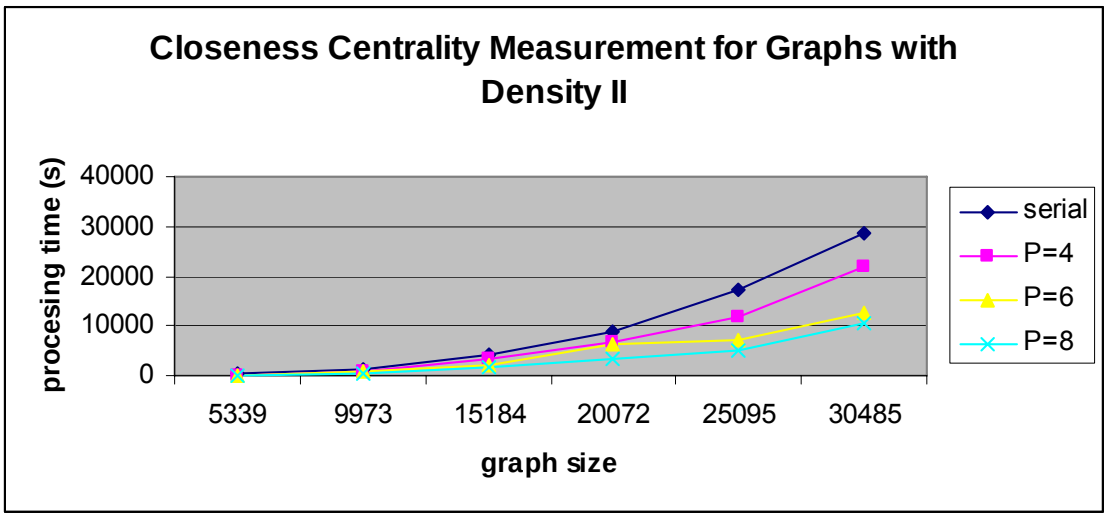



**Figure 7-5.** Running time comparison of UCINet and our serial algorithm for closeness centrality measurement

**b) Serial Algorithm vs. Our Parallel Approach**

In this experiment, all networks are static. We run our serial closeness centrality measurement algorithm on a set of static graphs from size 5,000 to 30,000 with 3 types of densities. We also run our parallel anytime approach on the same set of graphs with 4, 6, and 8 processors respectively. The running time of our parallel anytime approach and serial algorithm is shown in Figure 7-6.



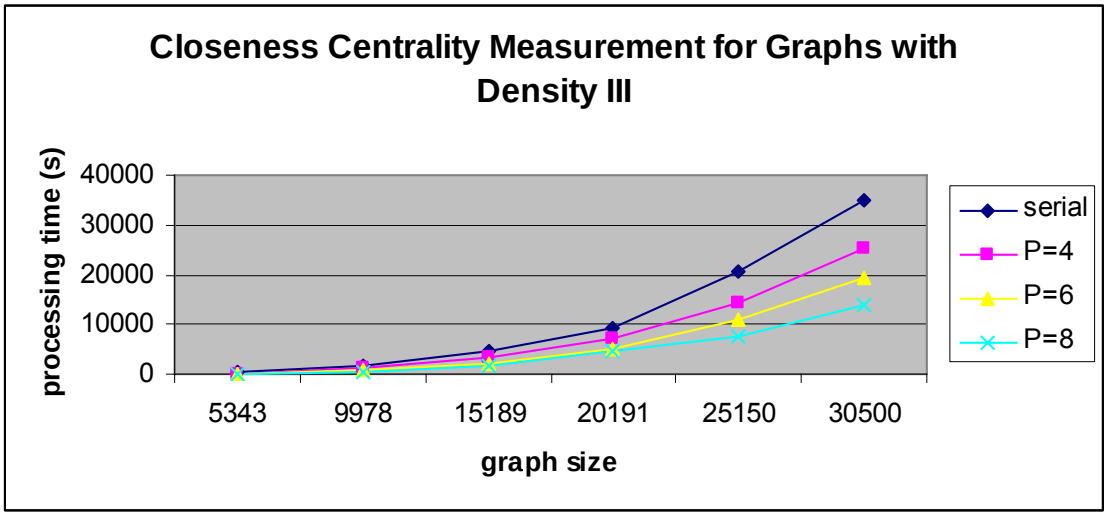



**Figure 7-6.** Closeness centrality measurement: serial vs. parallel.

From Figure 7-6, we can see that, for all graphs (all types of sizes and densities), our parallel approach can achieve the final results faster than the serial algorithm. The speed up of our approach does not close to *P* (the number of processors used in our system) especially when *P*=4. The reason is that the serial algorithm we used here is Dijkstra's algorithm, whose computation complexity is O(|*E*||*V*|). Whereas, our anytime approach for closeness measurement has the computational complexity of about O($|V|^3/k$).

Figure 7-6 also shows that when graphs are partitioned into more parts, our system will achieve better performance. However, one thing should be noticed here is that we cannot cut the original graphs into sub-graphs as small as we want. This is due to two reasons. First, the communication overhead will increase as the number of sub-graphs increases. Second, the analysis results obtained in the IA phase may not be able to approximate the original graph when the sub-graph is tiny. From our experimental experience, keeping the sub-graph size in the range 2000 to 4000 will give good performance.

### c) Anytime Property

Our approach for closeness centrality measurement is an anytime approach. Here, "anytime" is used to represent the idea of partially and incrementally

building solution and presenting useful partial results between the time the analysis process begins to the time it ends. The anytime property of our approach is shown in Figure 7-7.

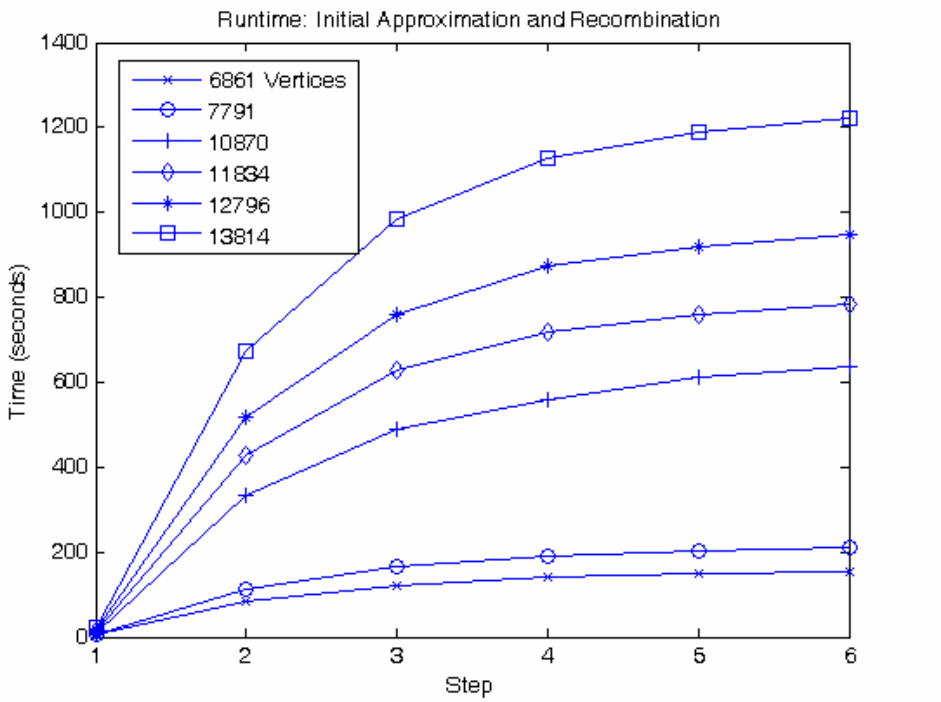


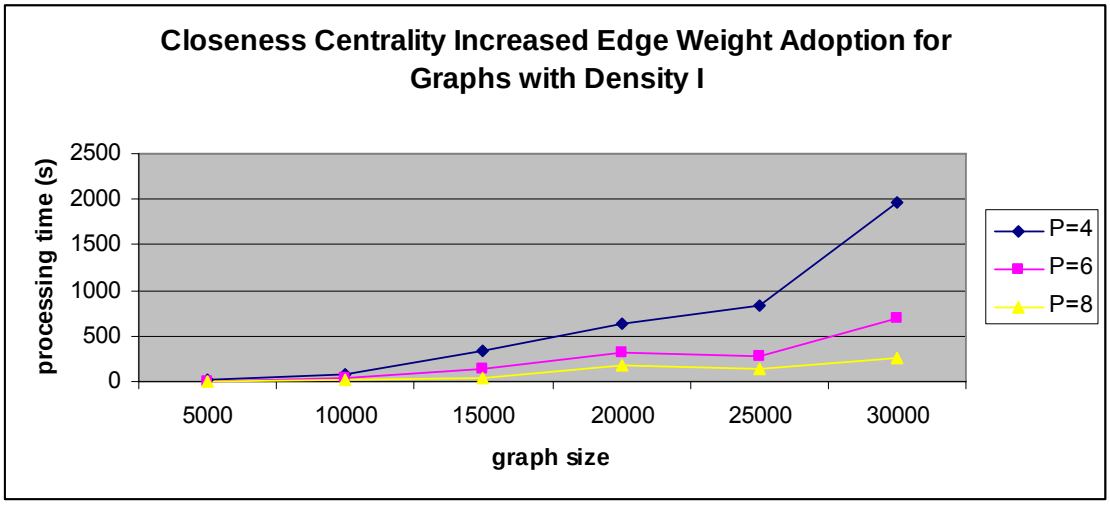

**Figure 7-7.** The anytime property of our approach.

In this figure, each curve represents a graph with a specific size. At each step, there are useful partial results presented to users. Step 1 represents the graph's initial approximation obtained by analyzing local sub-graphs. Step *i*, where *i*>1 represents the (*i*-1)th recombination stage in the RC phase. The vertical axis represents the time when partial results are obtained at step *i*. Figure 7-7 shows that our approach can provide an initial approximation of the network within a very short time, refine its partial solution step by step over time, and finally obtain the correct solutions. Thus, we say that our approach for closeness centrality measurement is an anytime algorithm.

### d) Anywhere Property – Dynamic Change Adoption

There are two types of dynamic changes simulated in our experiments, increased edge weights and decreased edge weights. Our anywhere recombination approach for closeness centrality measurement processes quite differently for these two types of changes. In following paragraphs, we will provide our experimental results for each type of dynamic changes.

*Increased Edge Weights*: When an edge weight is increased, the time for our system to adopt this change is shown in Figure 7-8. In this experiment, we generate a set of 4 edge changes on random nodes. We measure the time cost for our system to adopt each dynamic edge change. In Figure 7-8 figure, the time cost (processing time) is the average value of time costs of these 4 dynamic changes' adoption.



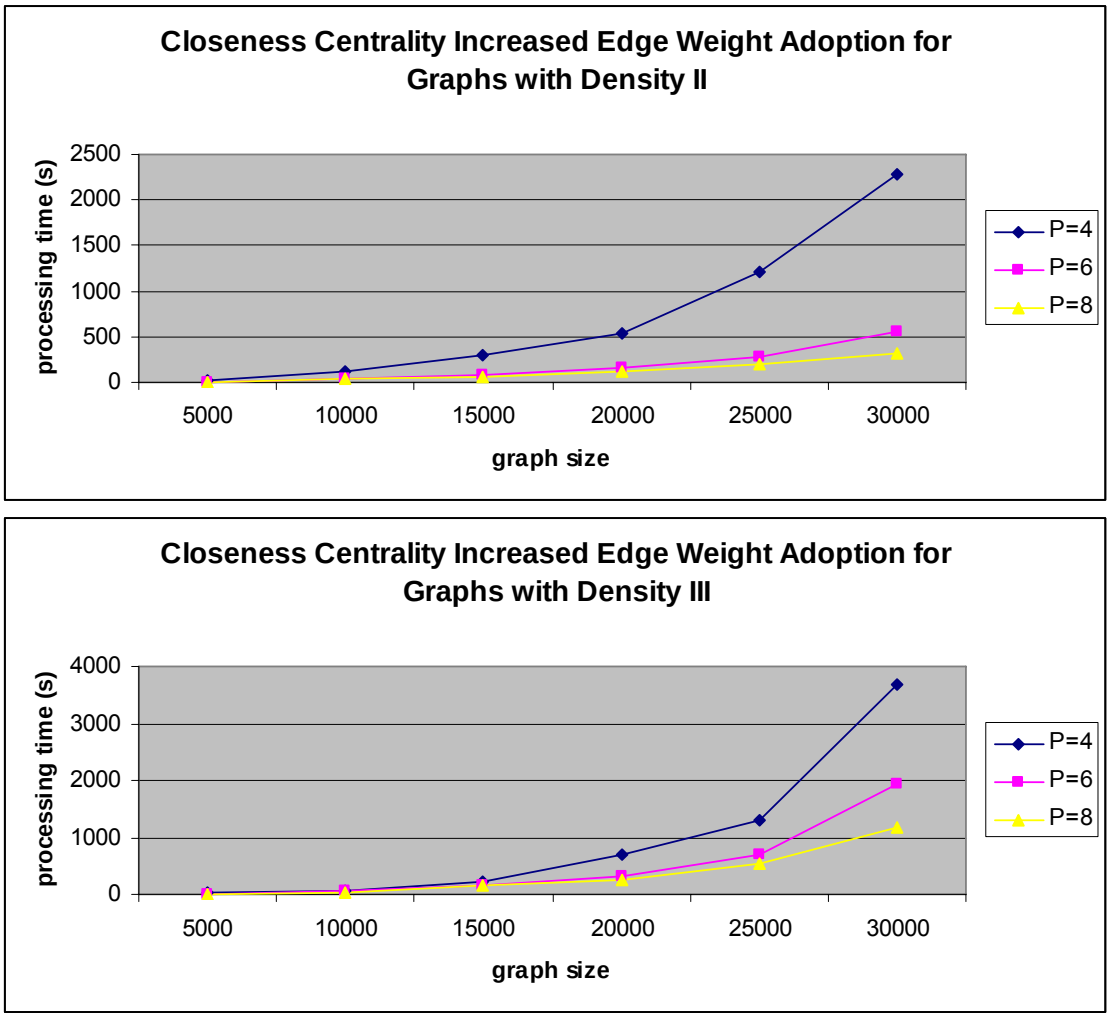



**Figure 7-8.** The time cost of our system to incorporate an edge with decreased weight.

From Figure 7-8 we can see that as graph size grows, the time cost for our system to adopt an increased edge weight increases. For graphs with the same size, when its density increases, the time cost also increases. In order to further study our system's ability of handling graph's dynamism, we generate a set of max degree changes. We record and compare the time costs for adopting random dynamic changes with the costs for handling max degree changes. The comparison for our system with 4 processors is shown in Figure 7-9. The corresponding comparisons for 6 and 8 processors are similar as this figure, thus they are not presented here.

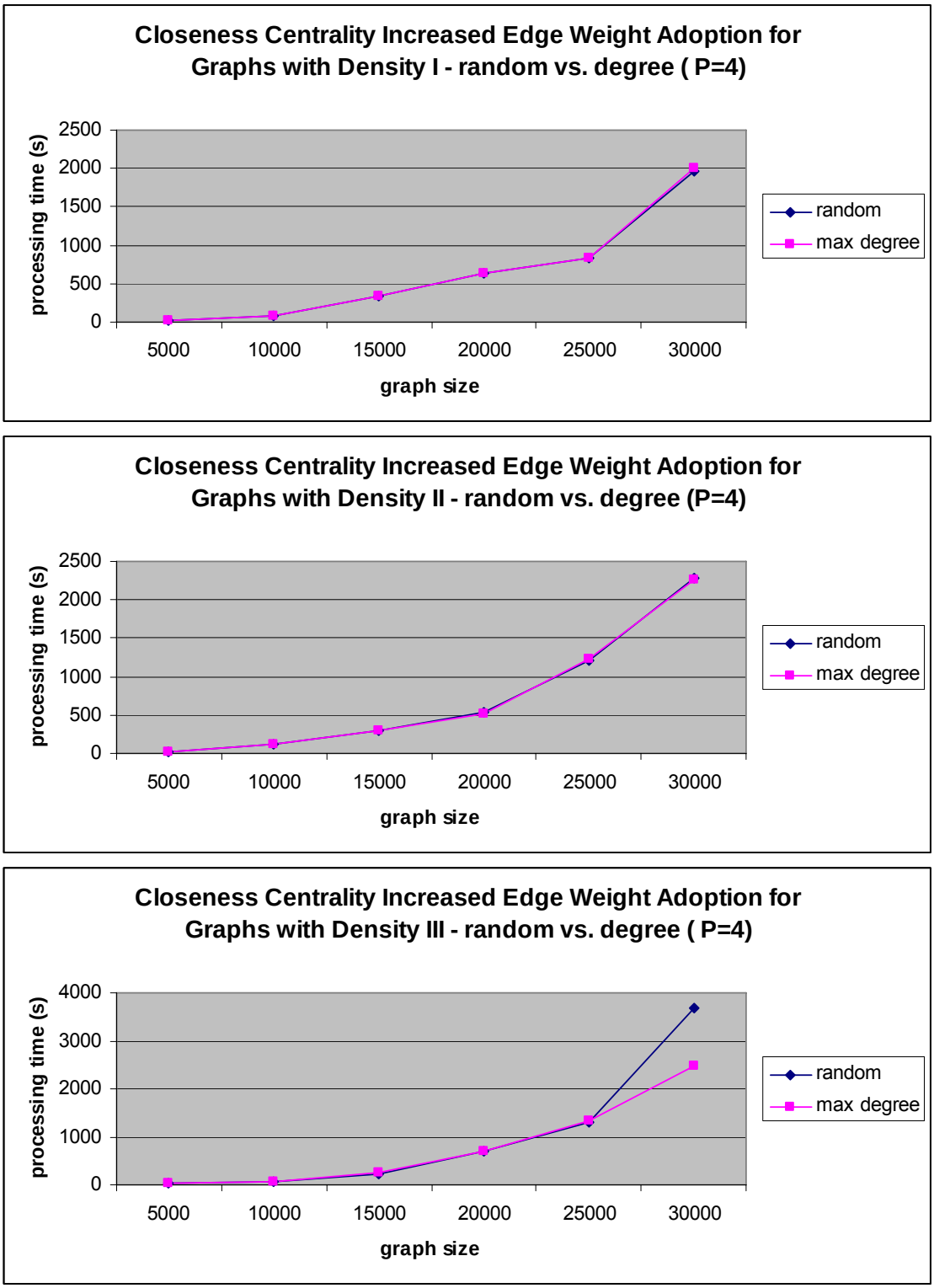
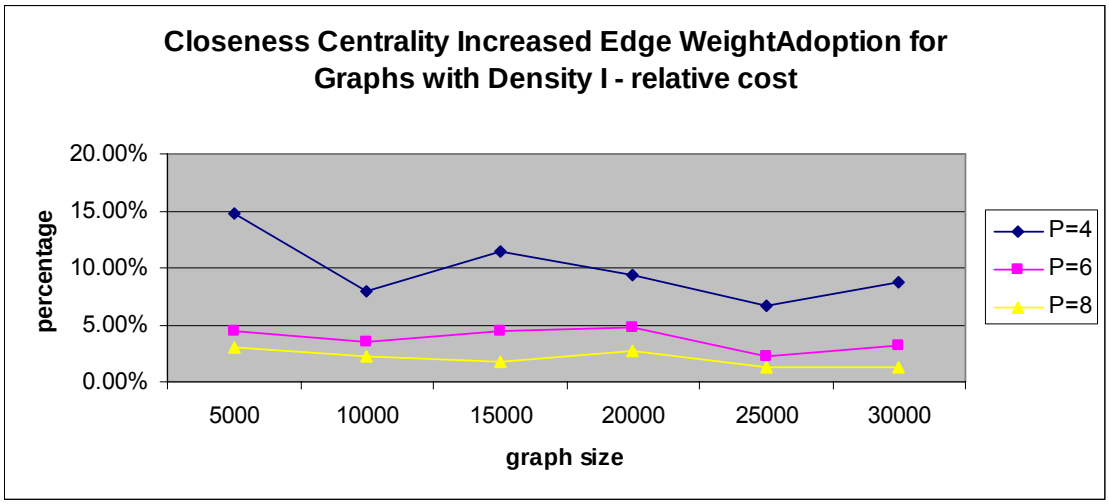



**Figure 7-9.** Performance comparison between handling random increased edge weights and handling max degree increased edge weights for closeness centrality measurement.

Figure 7-9 shows that the time cost for adopting random changes is quite similar to the cost for handling max degree changes. This is because that, as we discussed in the performance theoretical analysis of the anywhere approach for closeness centrality measurement, the work load for adopting an increased edge weight is affected by the edge betweenness value rather than the endpoints node degrees. Our anywhere approach for closeness centrality when an increased edge weight is not affected by the degree of this edge's endpoint.

When graph conditions (graph size and density) change, the time cost for our anywhere approach's performance also change. Only presenting the value of time cost does not provide enough information. We need to study the relative cost for handling dynamic changes. This relative cost is calculated as what we do in the experiment for ego-betweenness centrality dynamic adoption and is shown in Figure 7-10.



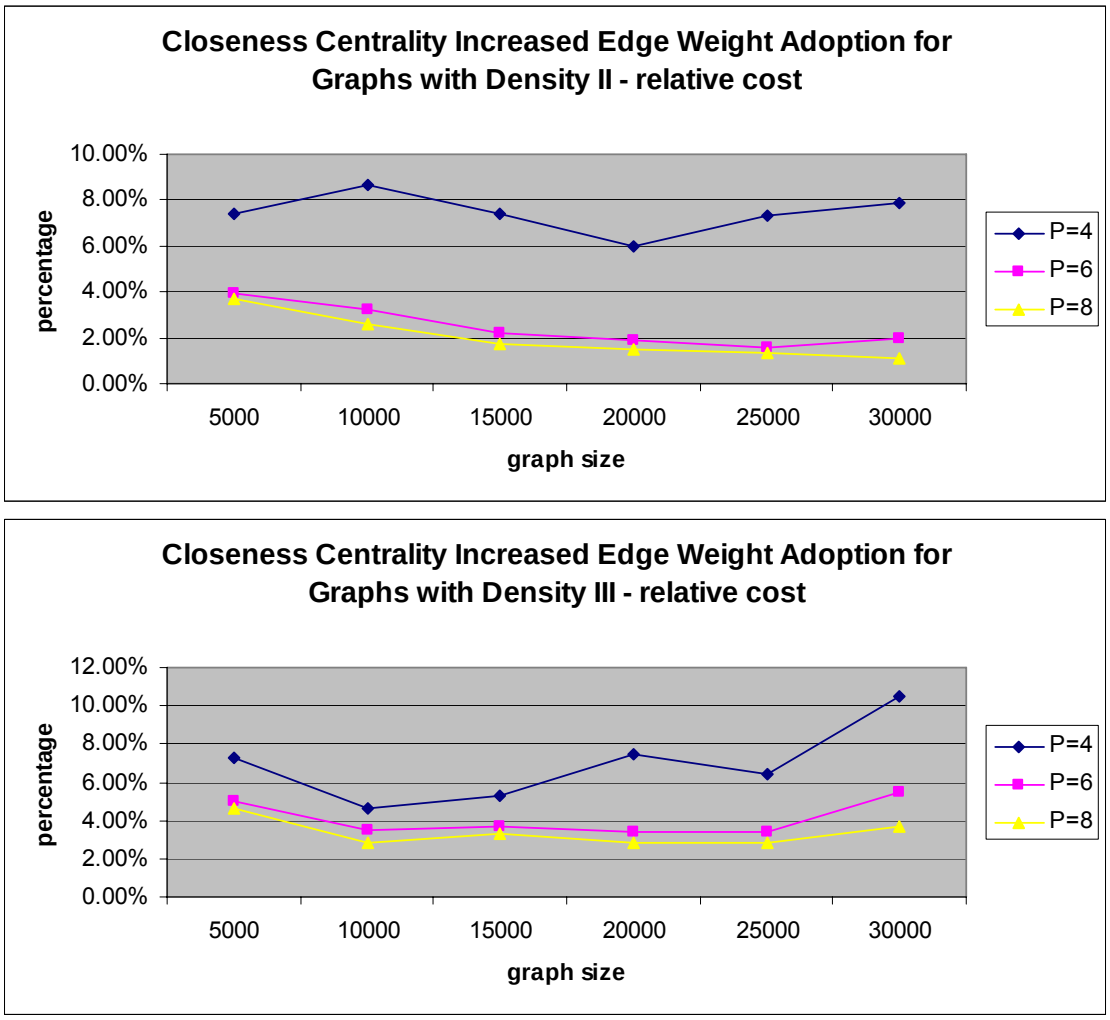
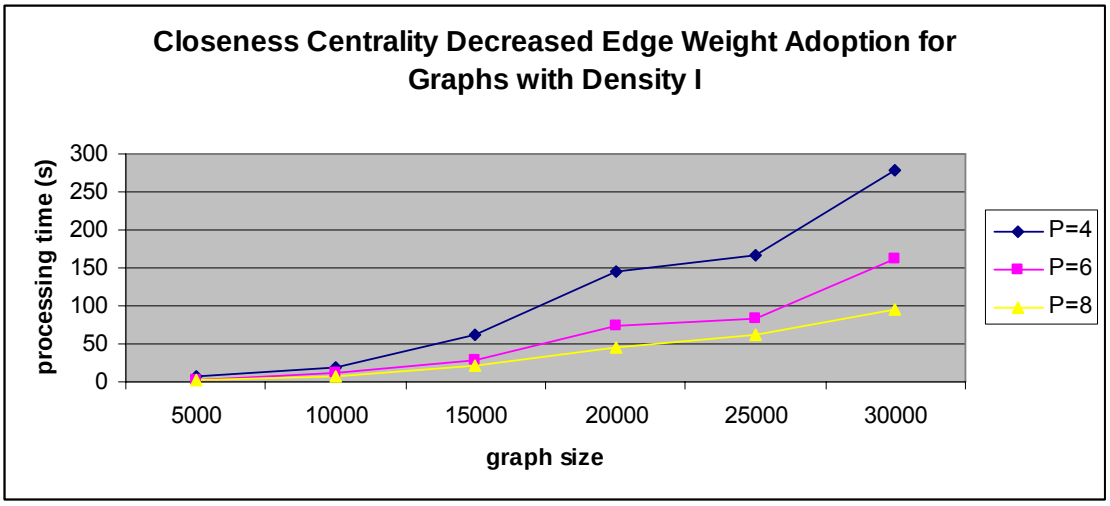
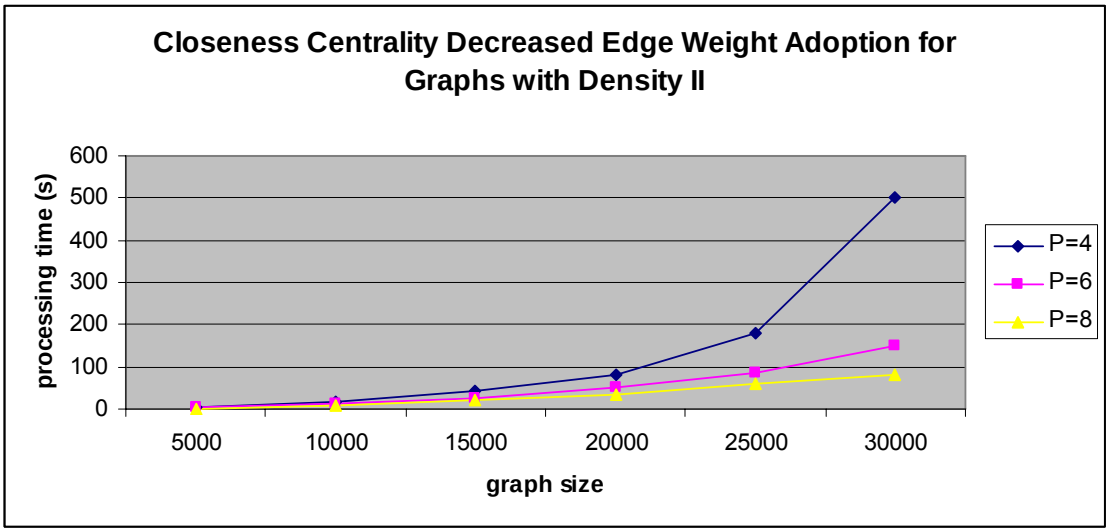



**Figure 7-10.** The relative cost for adopting an increased edge weight for closeness centrality measurement.

From this figure, we can obtain several key points. For all graphs, an increased edge weight can be adopted with a maximum relative cost as 15%. When we use more processors in our system, the relative cost decreases. This is because that, as we show in the system theoretical analysis, the work load for adopting increased edge weights is also affected how well the graph is decomposed. Decomposing graphs into 4 sub-graphs may not fit the real community structure contained in the original graphs. When P is 6 or 8, our system has very low relative costs whose maximum value is about 6%.

*Decreased Edge Weights*: In this experiment, we generate a set of 4 edge changes with decreased weights on random nodes. We measure the time cost for our system to adopt each dynamic edge change. The average time costs for our system to adopt these changes are shown in Figure 7-11.



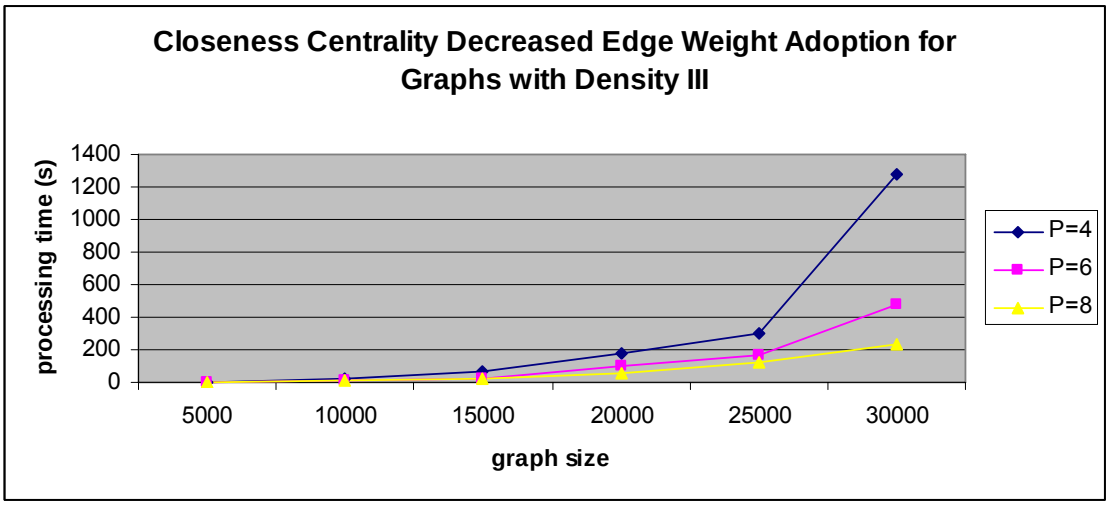
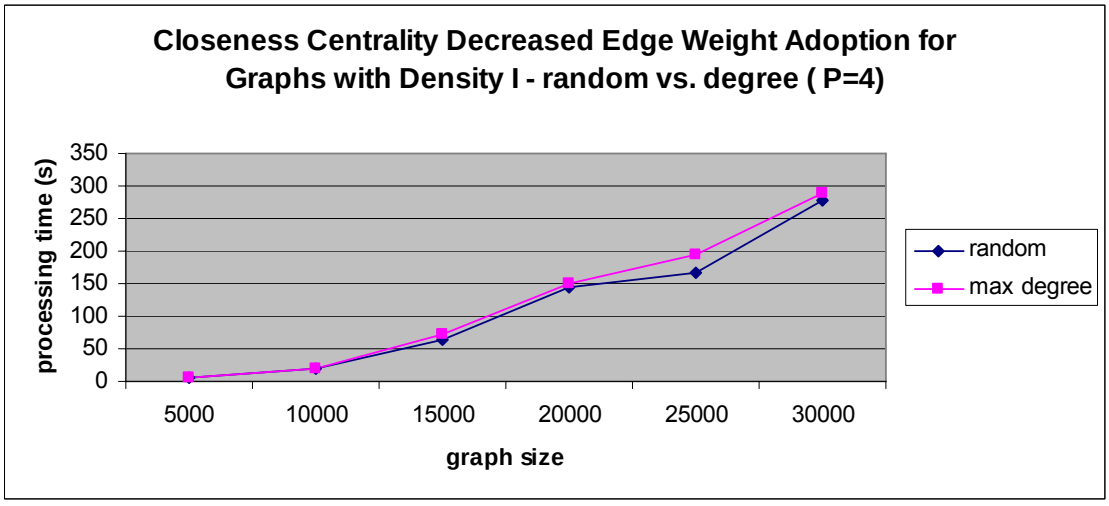



**Figure 7-11.** The time cost of our system to incorporate an edge with decreased weight.

Comparing Figure 7-11 with Figure 7-8 we can see that our system performs similarly on handling decreased weights and increased weights, except that the decreased edge weights require much less time. Similar as what we do for increased edge weights, in this experiment, we also test our system on a set of edge changes which happen on nodes with highest degrees. The comparison of random change adoption with max degree change adoption when P=4 is shown in Figure 7-12.



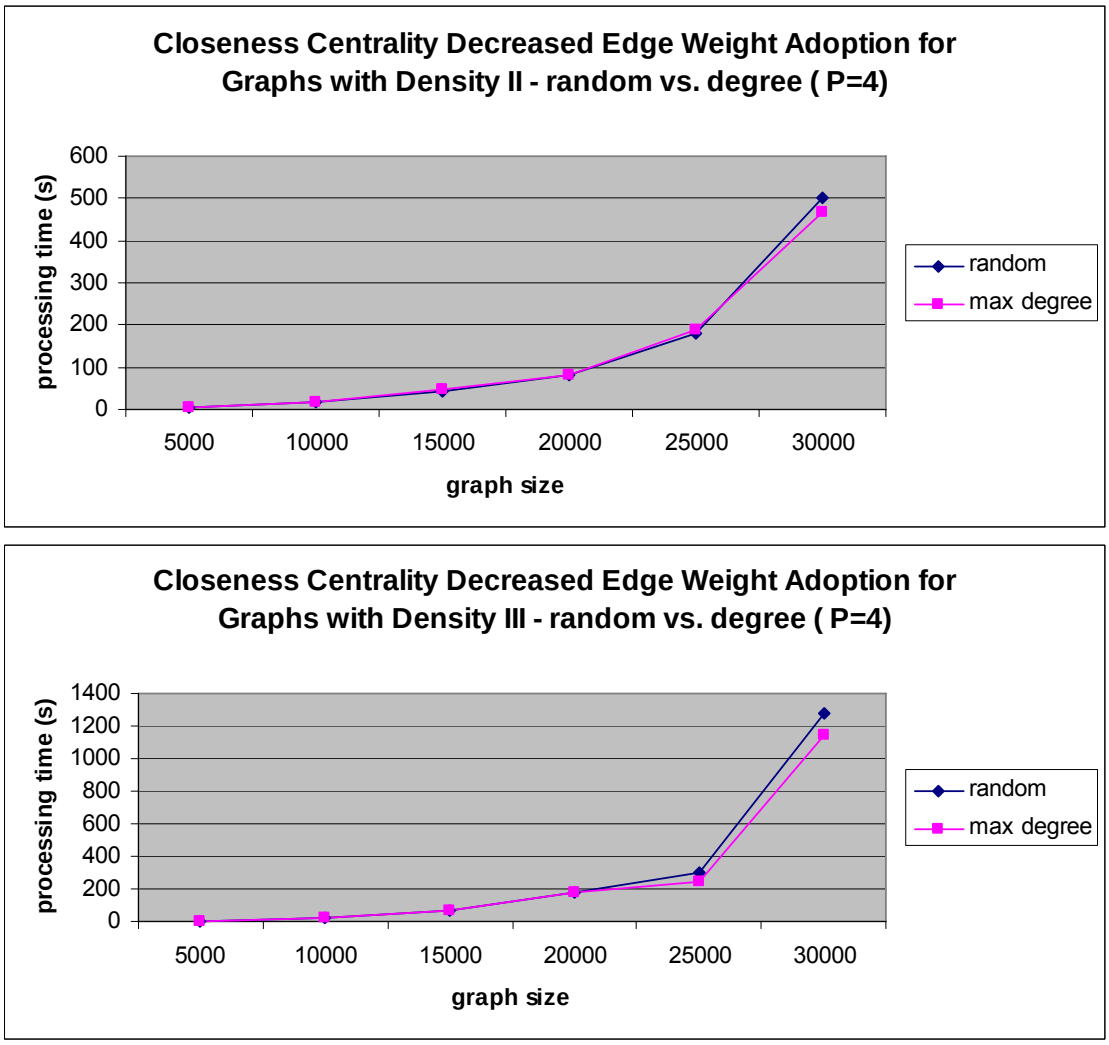
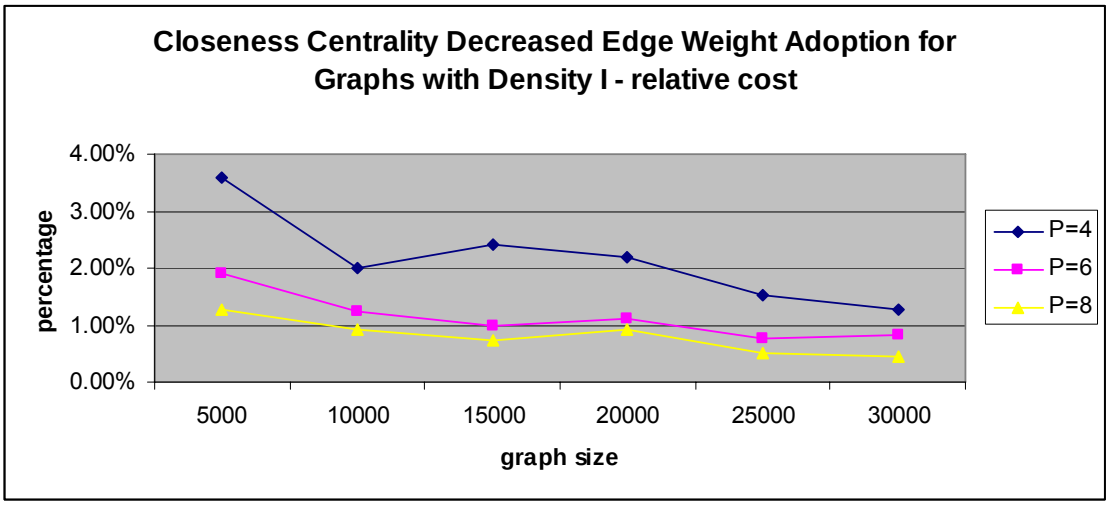


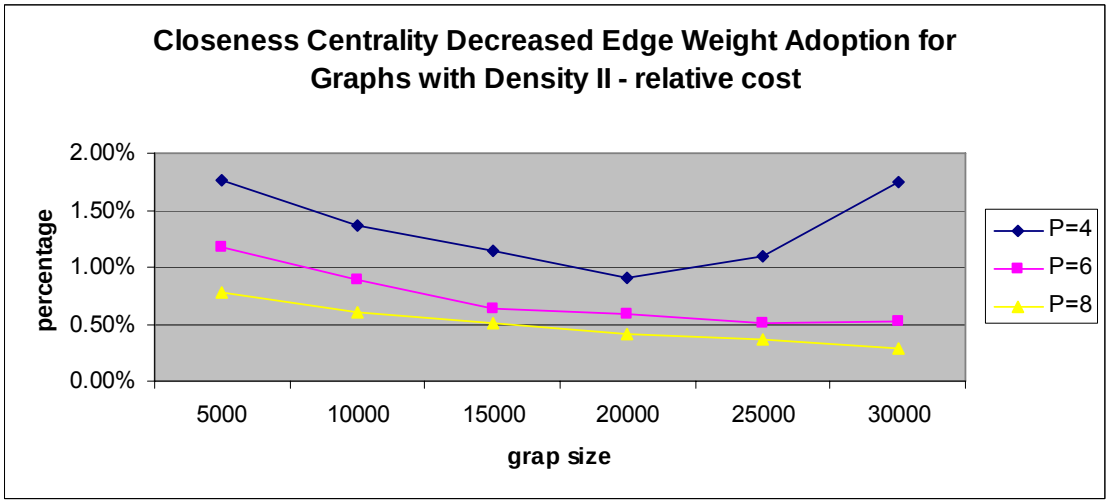

**Figure 7-12.** Relative costs for adopting a decreased edge weight for closeness centrality measurement.

We have similar comparison results for P=6 and 8. We will not present them in this document. From Figure 7-12 we obtain the conclusion that when an edge weight is decreased our anywhere approach for closeness centrality is not affected by the degree of this edge's endpoint.

The relative cost for adopting a decreased edge weight is shown in Figure 7-13. The relative time cost is calculated as what we do for increased edge weights. Figure 7-13 shows that for all graphs the maximum relative cost for handling a decreased edge weight is less than 4%. Similar as handling increased edge

weights, when we use more processors in our system, the relative cost for adopting decreased edge weights decreases. When P is larger than 4, our system has very low relative costs whose maximum value is about 2%.



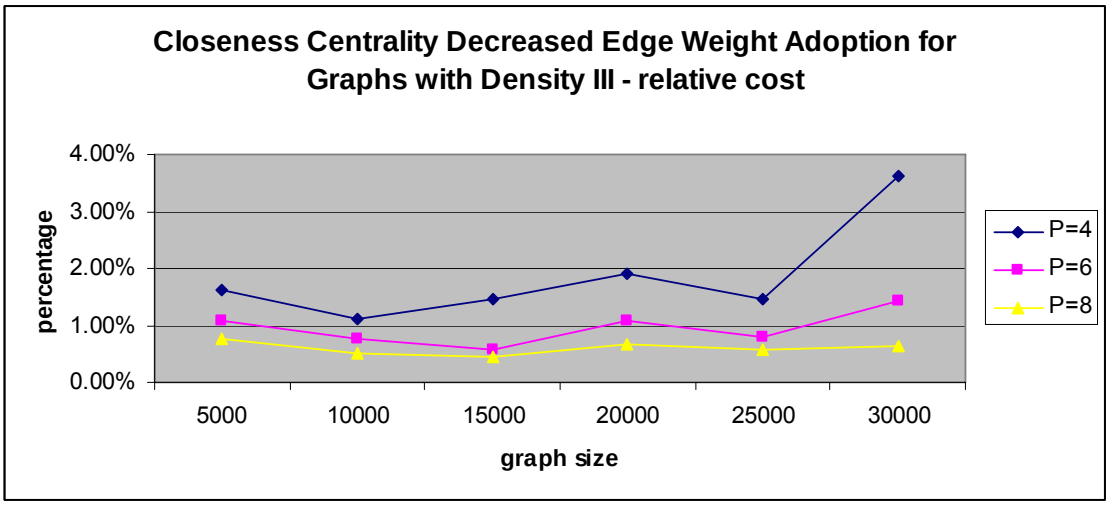



**Figure 7-13.** The relative cost for adopting a decreased weight for closeness centrality measurement.

Based on the experimental results of dynamic change adoption for closeness centrality measurement, we can obtain the following key points. Our approach can effectively handle both increased and decreased edge weights. For both types of dynamic edge changes, there is no evidence that the adoption is affected by the degrees of dynamic edge endpoints. When we use more processors, the relative cost for dynamic change adoption decreases. When an edge's weight is increased, the maximum relative cost for handle this change is about 15%. For decreased edge weight, the maximum relative cost is 4%.

## *7.4 Experiments on Maximal Clique Enumeration*

Maximal cliques are defined for unweighted and undirected graphs. Thus, in our experiments for maximal clique enumeration, all the graphs we use are undirected. Also, we dichotomize graphs by the following formula:

$$w'(u,v) = \begin{cases} 1 & when\ w(u,v) > 0 \\ 0 & otherwise \end{cases}$$

where w' is the new edge weight after the dichotomization.

### a) Serial Algorithm vs. Our Parallel Approach

In order to evaluate our methodology's performance on the problem of maximal clique enumeration, we implement a typical serial algorithm (Zhang's Algorithm introduced in chapter 4.6) to compare with. By running the serial algorithm, we obtain the number of maximal cliques contained in each graph. The number of maximal cliques for each graph is shown in Table 7-1. For all the graphs from size 5000 to 30000 and from Density I to Density III, there is no maximal cliques with size larger than 3.

**Table 7-1** Maximal cliques contained in each graph

| graph size | Density I | Density II | Density III |
|---|---|---|---|
| 5000 | 9975 / 11 | 19738 / 86 | 38332 / 559 |
| 10000 | 19970 / 6 | 39705 / 96 | 77941 / 684 |
| 15000 | 29972 / 5 | 59739 / 66 | 103656 / 443 |
| 20000 | 40955 / 7 | 80746 / 82 | 171706 / 595 |
| 25000 | 51964 / 6 | 104735 / 86 | 202898 / 690 |
| 30000 | 62941 / 11 | 121737 / 85 | 241944 / 677 |

In this table, the value before "/" is the number of maximal 2-cliques contained in the graph. The value after this symbol is the number of maximal 3-cliques. As shown in Table 7-1, graphs with Density I have very small number of maximal 3-cliques because of the low density.

The time costs for finding all maximal cliques contained in each graph by the serial algorithm and our parallel approach are shown in Figure 7-14.

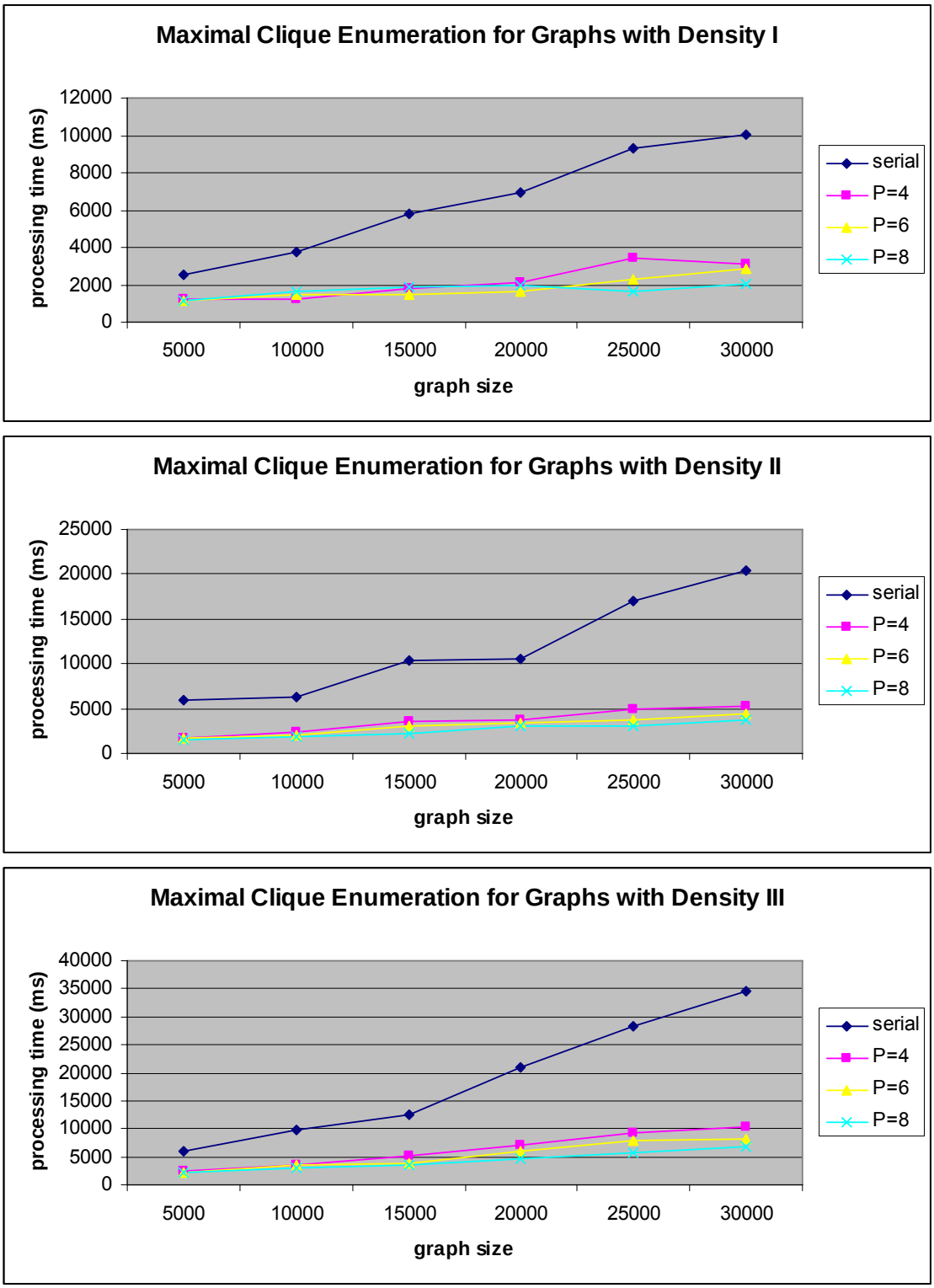
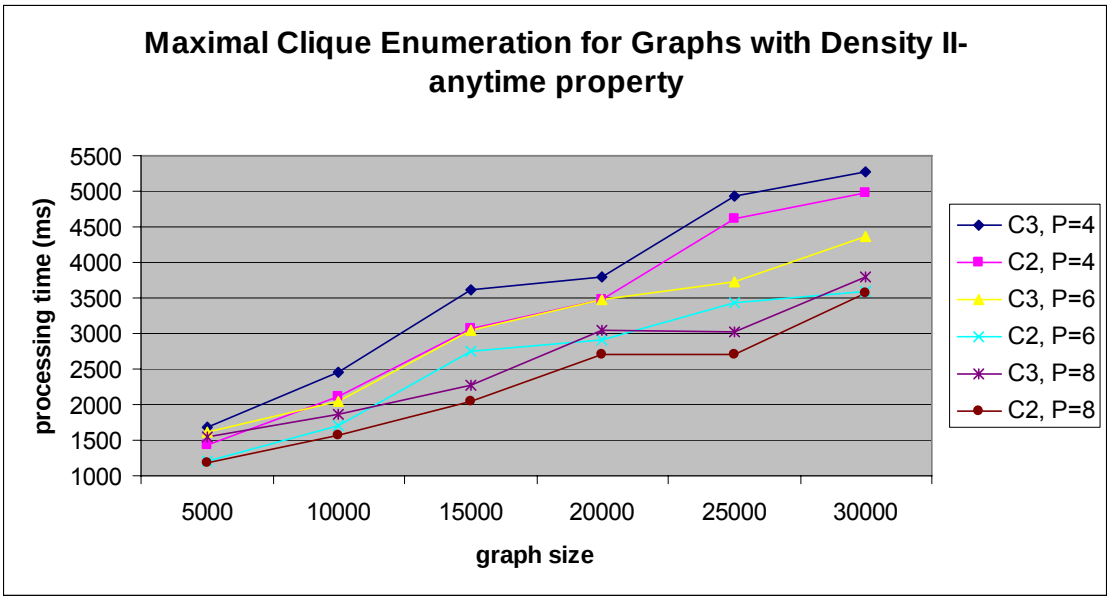



**Figure 7-14.** Time costs for finding all maximal cliques contained in graphs.

This figure shows that as graph size increases, the time for finding all maximal cliques will increase accordingly. For all types of graphs, our parallel approach can solve the maximal clique enumeration problem faster than the typical serial algorithm (Zhang's algorithm). As the number of processors used in our system increases, the time cost for solving the problem decreases.

**b) Anytime Property**

Similar to the typical serial algorithm (Zhang's algorithm) for maximal clique enumeration, our parallel approach generates maximal cliques with increasing size. The anytime property of our approach is demonstrated as that all maximal 2-cliques are first generated. Then maximal cliques with larger and larger size are obtained stage by stage. The anytime property of our approach is shown in Figure 7-15. In this figure, we present the time for generating maximal 2-cliques and maximal 3-cliques for graphs with Density II and III. We do not study the anytime property of graphs with Density I since these graphs have very few (less than 10) maximal cliques with size larger than 2. Almost all the work for Density I graphs is done in finding maximal 2-cliques.



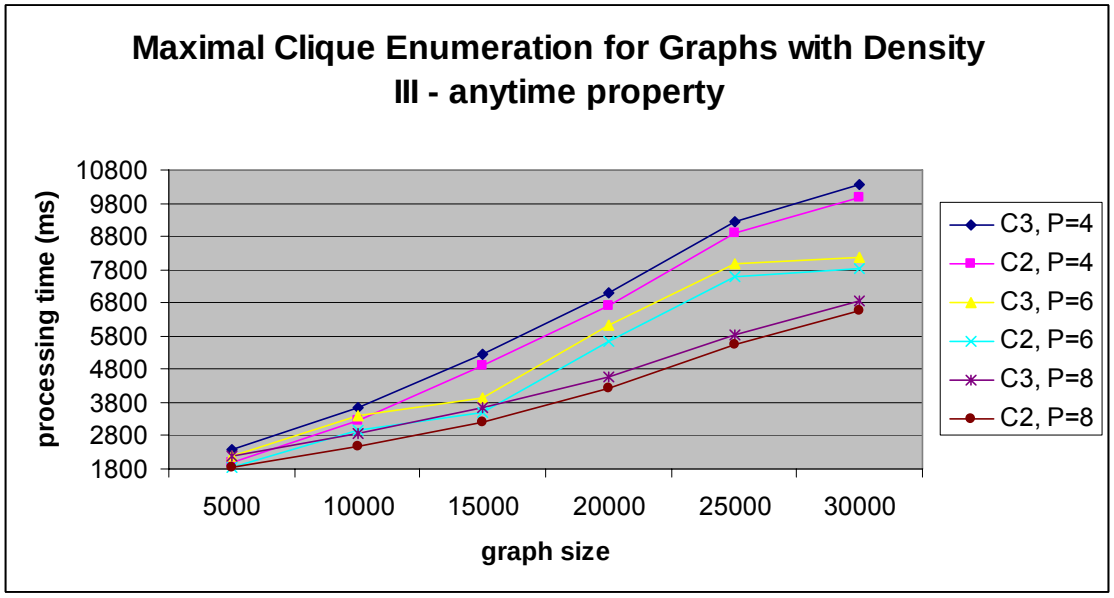



**Figure 7-15.** The anytime property of our approach for maximal clique enumeration

In this figure "Cx, P=y" represents the time cost for our system to find all maximal cliques with sizes no larger than x by using y processors. As shown in Figure 7-15, our approach first finds all maximal 2-cliques. As time evolves, our approach obtains all maximal 3-cliques. The time difference for obtaining all maximal 2-cliques and all maximal cliques whose size is no larger than 3 is not significant. This is because that the number of maximal 3-cliques is much smaller than the number of maximal 2-cliques. When graph density increases, the anytime property will become more preeminent.

In order to further study the anytime property of our approach for maximal clique enumeration, we decide to test our system on highly dense graphs. As we mentioned, dense graphs cannot be large. However, it is quite possible that a small portion of a large social network may have intensive inside connections. Thus, we decide to test our anytime approach on dense but "small" graphs. In this experiment, we generate a set of graphs with high densities and graph size from 400 to 800 in increasing size with increments of about 100. For each graph size, we generate graphs with three types of densities, 10%, 15%, and 20%.

By running the serial algorithm (Zhang's Algorithm), we obtain the information of maximal cliques for each graph. This is shown in Tables 7-2, 7-3, and 7-4 where *n* and *M* represent graph size and maximal clique size respectively.

**Table 7-2**. Maximal cliques contained in graphs with density as 10%

| n \ M | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| 400 | 186 | 6604 | 943 | 7 | 0 |
| 500 | 154 | 11377 | 2364 | 30 | 0 |
| 600 | 86 | 18091 | 4479 | 40 | 1 |
| 700 | 46 | 26342 | 8838 | 123 | 0 |
| 800 | 45 | 36056 | 13118 | 215 | 2 |

**Table 7-3.** Maximal cliques contained in graphs with density as 15%

| n \ M | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| 400 | 5 | 9348 | 7465 | 322 | 1 |
| 500 | 4 | 14075 | 16563 | 854 | 9 |
| 600 | 0 | 17662 | 34032 | 2235 | 11 |
| 700 | 0 | 21566 | 59500 | 4783 | 46 |
| 800 | 0 | 25033 | 95314 | 8348 | 88 |

**Table 7-4.** Maximal cliques contained in graphs with density as 20%

| n \ M | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| 400 | 0 | 5538 | 25039 | 3692 | 67 | 0 |
| 500 | 0 | 5944 | 54132 | 10622 | 240 | 0 |
| 600 | 0 | 5677 | 100269 | 26182 | 678 | 2 |
| 700 | 0 | 5229 | 164267 | 50644 | 1400 | 4 |
| 800 | 0 | 4229 | 151391 | 101710 | 3513 | 15 |

From these tables we can see that the maximal cliques with middle size are the major part. For graphs with density as 10% and 15%, the middle clique size is 3 and 4. For graphs with density as 20%, the middle clique size is 4 and 5. Our anytime approach is expected to put relatively small time to handle all 2-cliques. Then, it will expend a lot of time to expand candidate cliques to generate all middle size cliques. When clique size is large (exceeding middle range), the number of cliques falls down exponentially. Our anytime approach will handle these large cliques within a short period of time. The time costs for our anytime approach to find cliques for all dense graphs are show in Figure 7-16.

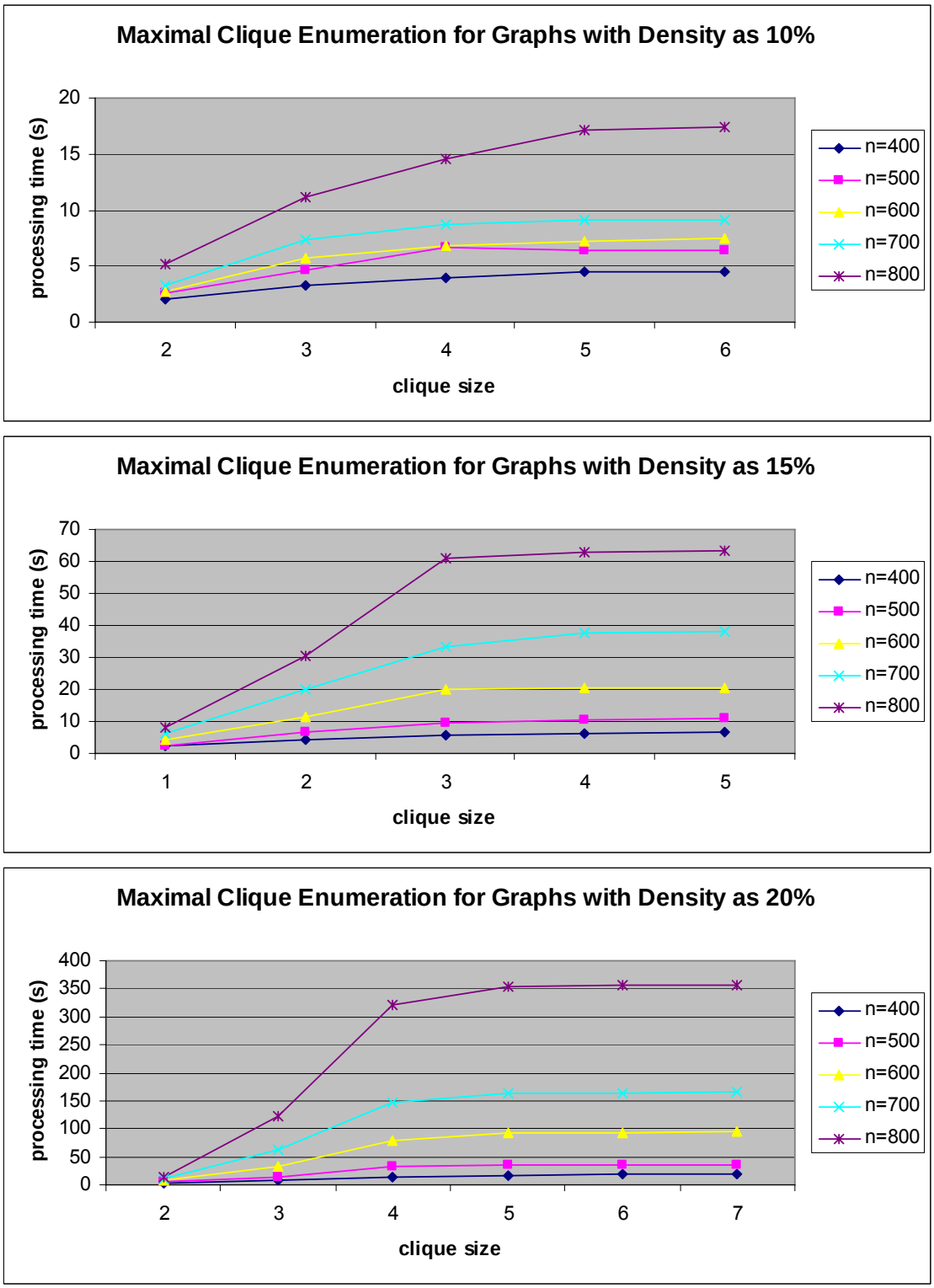



**Figure 7-16.** The performance of our maximal clique enumeration anytime approach for dense graphs.

From this figure we can see that our approach first provides quick analysis results with maximal cliques which have small sizes, and it generates larger and larger maximal cliques stage by stage. Thus we can see that our approach for maximal clique enumeration demonstrates the anytime property well when a social network is dense or there is a part with dense connections in a large social network.

### c) Anywhere Property – Dynamic Change Adoption

In this experiment, we test our anywhere approach for maximal clique enumeration on two sets of 8 dynamic edge changes: random edge changes and max degree edge changes. We measure the time cost of our system to handle each edge change. For each set, we take the average value as the time cost for adopting one dynamic change. We find that there is no obvious evidence that the time costs for adopting these two types of changes for maximal clique enumeration are different. The time costs and relative costs for adopting a random edge change are shown in Figure 7-17 and Figure 7-18 respectively.

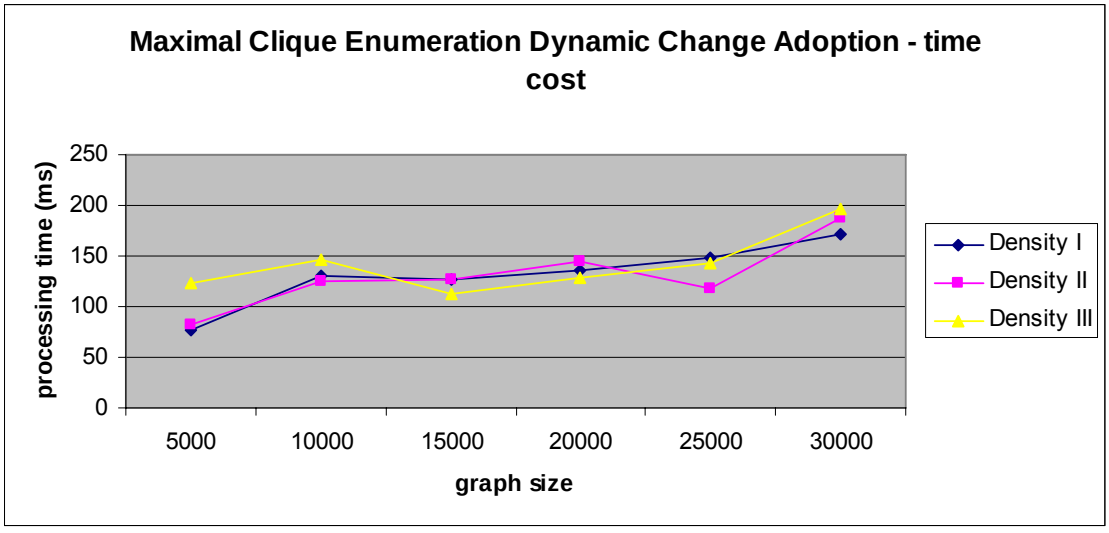



**Figure 7-17.** Time costs for adopting one random dynamic change for maximal clique enumeration.

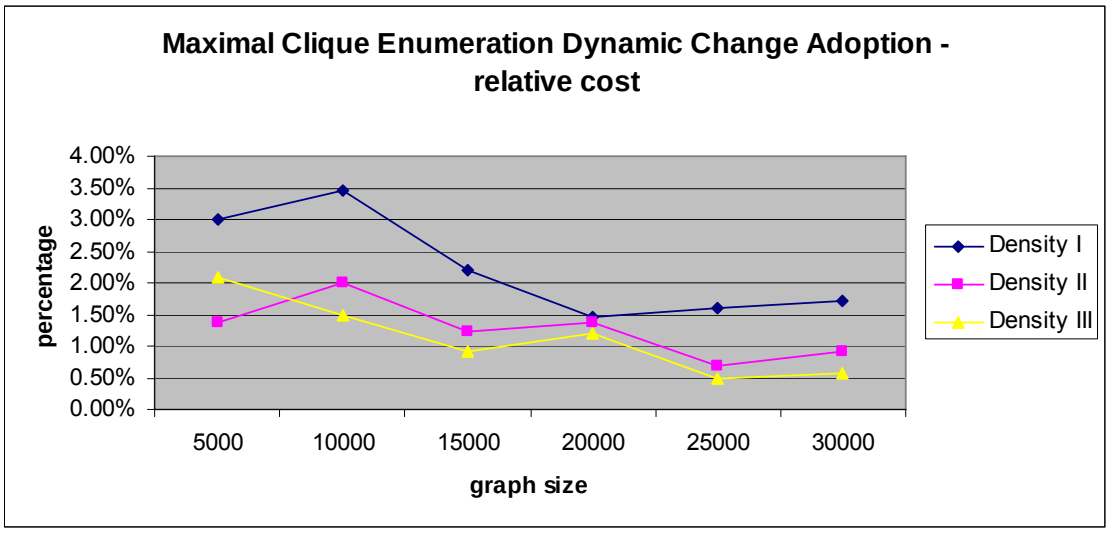



**Figure 7-18.** Relative cost for adopting one dynamic edge change for maximal clique enumeration.

These two figures show that our anywhere approach for maximal clique enumeration can effectively handle dynamic graphs. The relative cost for adopting a random edge change is less than 3.5%.

## *7.5 Summary*

In this chapter, we present experimental results of our anytime-anywhere approaches for the measurement of ego-betweenness centrality, closeness centrality, and all maximal cliques for large and dynamic social networks. For all these three SNA metrics, our anytime anywhere approaches can obtain the final solution faster than the typical serial algorithms used in current SNA tools. For closeness centrality and maximal clique enumeration, experimental results demonstrate our approach's ability to provide useful partial results and refine obtained partial results stage by stage. For all three SNA metrics, experimental results demonstrate and validate the effectiveness and efficiency of our approach on handling graph's dynamism.

# 8. Conclusion & Future Work

Social Network Analysis is an important research topic and has been successfully applied in a broad range of fields, with many SNA techniques and methodologies developed. However, they are designed for small networks and are not suitable for analyzing large networks. Recently, there are numerous large networks with considerable significance and special structural properties that have emerged from extensive research fields. Effectively and efficiently analyzing large social networks is an emergent task which introduces new challenges. The most fundamental and critical challenges are long processing time, large computational resource requirement, and graph dynamism.

In this dissertation, in order to address these challenges, we described an anytime-anywhere methodology for large social network analysis based on a parallel/distributed environment. Our methodology consists of three phases: (1) domain decomposition, (2) initial approximation, (3) recombination. The domain decomposition phase takes the charge of partitioning a large network into smaller ones. A fast and initial approximation of the network analysis is achieved by analyzing these small sub-networks in the initial approximation phase. The approximation will be refined and a set of partial solutions with increasing quality will be provided to analysts as time evolves in the recombination phase. Finally, after the convergence of recombination, the exact or good approximate solution will be obtained. During the analyses process, graph's dynamic change will be naturally adopted based on the obtained partial results.

In order to evaluate and validate our methodology, we design and implement our system for three SNA metrics: ego-betweenness centrality, closeness centrality, and maximal clique enumeration. Based on theoretical performance analysis, we show the advantages of our approaches for analyzing large social networks. We

also test our system on a set of random graphs with various sizes and densities are generated. Experimental results demonstrate that our system overcomes the scalability issues of popular industry SNA tools and effectively handles graph's dynamism. By our experiments, we validate that our methodology is an anytime-anywhere approach with the ability to effectively and efficiently analyze large and dynamic graphs with various densities.

The work discussed in this document presents a fundamental design and validation of our anytime-anywhere methodology for large and dynamic social networks analysis. There are still some topics needing more work. In future, we will further evaluate and validate our methodology by including more SNA metrics in our system. Also, we would like to study in depth the domain decomposition phase to analyze, identify and validate general metrics which are critical for analyzing normal social networks and special metrics which are important for particular networks or SNA applications. Another area for more work is in determining and improving on the accuracy of the initial approximation. We would like to be able to find an upper bound on the how inaccurate an initial approximation can be from the known result. Measuring this empirically can give some insight as to which metrics for domain decomposition allow for the best initial approximation.

# Reference


[Adamic'99] L. A. Adamic, 1999, "The Small World Web", *Proceedings of the Third European Conference, ECDL'99* (Springer-Verlag, Berlin), p. 443.

[AielloCL'00] W. Aiello, F. Chung, L. Lu, 2000, "A Random Graph Model for Massive Graphs", Proceedings of the 32nd ACM Symposium on the Theory of Computing, p. 171.

[AlbertB'02] R. Albert, and A. L. Barabasi, 2002, "Statistical Mechanics of Complex Networks", Review of Modern Physics, vol. 74, p. 47.

[AusielloIMN'91] G. Ausiello, G. Italiano, A. Marchetti-Spaccamela, and U. Nanni, 1991, "Incremental Algorithms for Minimal Length Paths", Journal of Algorithms, vol. 12, p.615.

[BaaseG'00] S. Baase, and A. V. Gelder, 2000, Computer Algorithms: Introduction to Design & Analysis, Pearson Education Pte. Ltd.

[BarabasiJNRSV'02] A. L. Barabasi, H. Jeong, Z. Neda, and E. Ravasz, A. Schubert, T. Vicsek, 2002, "Evolution of the Social Network of Scientific Collaborations", Physica A, vol. 331, (3-4) pp. 590.

[BarnardS'94] S. T. Barnard, and H. D. Simon, 1994, "A Fast Multilevel Implementation of Recursive Spectral Bisection", Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing, pp. 711.

[BatageljM'04] V. Batagelj, and A. Mrvar, 2004, Pajek: Package for Large Networks, Version 1.00. Ljubljana: University of Ljubljana.

[Bauer'04] E. Bauer, 2004, An Overview of the Weblog Tools Market, [Online], Available: http://www.elise.com/web/a/an_overview_of_the_weblog_tools_market.php

[Benta'04] I. M. Benta, 2004, Agna, Version 2.1.1. Cork: University College Cork, Ireland.

[BoerNHSSZ'04] P. Boer, R. de Negro, M. Huisman, T. A. B. Snijders, C. E. G. Steglich, and E. P. H. Zeggelink, 2004, StOCNET: An Open Software System for the Advanced Statistical Analysis of Social Networks, Version 1.5. Groningen: ICS / Science Plus Group, University of Groningen.

[BorgattiEF'02] S. P. Borgatti, M. G. Everett, and L. C. Freeman, 2002, UCINET 6 for Windows: Software for Social Network Analysis, Harvard: Analytic Technologies.

[Borgatti'02] S. P. Borgatti, 2002, NetDraw 1.0: Network Visualization Software, Version 1.0.0.21, Harvard: Analytic Technologies.

[BrandesE'05] U. Brandes, and T. Erlebach, 2005, Network Analysis: Methodological Foundations, Springer-Verlag Berlin Heidelberg.

[BroderKMRRSTW'02] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener, 2002, "Graph Structure in the Web", Computer Networks, vol. 33, p. 309-320.

[BronK'73] C. Bron, and J. Kerbosch, 1973, "Algorithm 457: Finding All Cliques of an Undirected Graph", Proceedings of the ACM, vol. 16, pp. 575-577.

[CanchoS'01] R. F. Cancho, and R. V. Sole, 2001, The Small-World of Human Language, Santa Fe Institute working paper 01-03-016

[CarringtonSW'05] P. J. Carrington, J. Scott, and S. Wasserman, 2005, Models and Methods in Social Network Analysis, Cambridge University Press

[Compbell'04] A. P. Campbell, 2004, "Using LiveJournal for Authentic Communication in EFL Classes", The Internet TESL Journal, vol. 5, No. 9.

[Cyram'04] Cyram, 2004, Cyram NetMiner II, Version 2.4.0, Seoul: Cyram Co., Ltd.

[DeanB'88] T. Dean, and M. Boddy, 1988, "An Analysis of Time-Dependent Planning", Proceedings of the 7th National Conference on Artificial Intelligent, p. 49.

[DemetrescuI'03] C. Demetrescu, and G. F. Italiano, 2003, "A New Approach to Dynamic All Pairs Shortest Paths", Proceedings of ACM Symposium on Theory of Computing, p.159, 2003.

[Dijkstra'59] E. W. Dijkstra, 1959, "A Note on Two Problems in Connexion with Graphs", Numerische Mathematik, vol. 1, S. 269-271.

[DorogovtsevM'01] S. N. Dorogovtsev, and J. F. F. Mendes, 2001, "Evolution of Networks", Advance in Physics, vol. 51, p. 1079.

[DuWXWP'06] N.Du, B. Wu, L. Xu, B. Wang, and X. Pei, 2006, "A Parallel Algorithm for Enumerating All Maximal Cliques in Complex Network", the sixth IEEE International Conference on Data Mining – Workshop.

[ErdosR'59] P. Erdos, and A. Renyi, 1959, "On Random Graphs", Publicationes Mathematicae, vol. 6, p. 290.

[EverettB'05] M. Everett, and S. P. Borgatti, 2005, "Ego Network Betweenness", Social Networks, vol. 27, p. 31.

[FaceBook] http://www.facebook.com

[FakcharoemopholR'01] J. Fakcharoemphol, and S. Rao, 2002, "Plannar Graphs, Negative Weight Edges, Shortest Paths, and Near Linear Time", Proceedings of IEEE Annual Symposium on Foundations of Computer Science, pp. 232, 2002.

[FiduccioM'82] C. Fiduccia, and R. Matteyses, 1982, "A Linear Time Heuristic for Improving Network Partitions", Proceeding of 19th Design Automation Conference, pp. 175-181.

[Floyd'62] R. W. Floyd,. 1962, "Algorithm 97: Shortest Path", Communications of ACM, vol. 5, pp. 570-576.

[Frank'02] O. Frank, 2002, "Using centrality modeling in network surveys", Social Networks, vol. 24 p. 385.

[Fredman'76] M. L. Fredman, 1976, "New Bounds on the Complexity of the Shortest Path Problem", SIAM Journal of Computation, vol. 5, p. 83.

[Freeman'79] L. C. Freeman, 1979, "Centrality in social networks: I. conceptual clarification", Social Networks, vol. 1 p. 215.

[Freeman'02] L. C. Freeman, 2002, The Study of Social Networks, [Online], Available: http://www.insna.org/INSNA/na_inf.html

[Freeman'04] L. C. Freeman, 2004, The Development of Social Network Analysis: A Study in the Sociology of Science, Booksurge LIc.

[FronczakHJS'02] A. Fronczak, J. A. Holyst, M. Jedynak, and J. Sienkiewicz, 2002, "Higher Order Clustering Coefficients in Barabasi-Albert Networks", Physica A, vol. 316, p. 688.

[GoodrichT'02] M. T. Doodrich, and R. Tamassia, 2002, Algorithm Design: Foundations, Analysis, and Internet Examples, New York, Chichester, Wiley, c2002.

[GovindanT'00] R. Govindan, and H. Tangmunarunkit, 2000, "Heuristics for Internet Map Discovery", Proceedings of IEEE INFOCOM 2000, vol. 3, p. 1371.

[GuimeraA'05] R. Guimera, and L. A. N. Amaral, "Functional Cartography of Complex Metabolic Networks", Nature, vol. 433, p. 895.

[HanPR'97] Y. Han, V. Pan, and J. Reif, 1997, "Efficient Parallel Algorithms for Computing All Pair Shortest Paths in Directed Graphs", Algorithmica, vol. 17, p. 399.

[HannemanR'05] R. Hanneman, and M. Riddle, 2005, Introduction to Social Network Methods, [Online textbook], Available: http://www.faculty.ucr.edu/~hanneman/nettext/

[HendricksonL93] B. Hendrickson, and R. Leland, 1993, A Multilevel Algorithm for Partitioning Graphs, Technical Report SAND93-1301, Sandia National Laboratories.

[HenzingerKRS'97] M. Henzinger, P. Klein, S. Rao, and S. Subramanian, 1997, "Faster Shortest-Path Algorithms for Plannar Graphs", Journal of Computer and System Science, vol. 55, p. 3.

[IMDb] http://www.imdb.com

[Johnston'76] H. C. Jonston, 1976, "Cliques of a Graph-Variations on the Bron-Kerbosch Algorithm", International Journal of Computer and Information Sciences, vol. 5, pp 209-238.

[LiljerosEASA'01] F. Liljeros, C. R. Edling, L. A. N. Amaral, H. E. Stanley, and Y. Aberg, 2001,"The Web of Human Sexual Contacts", Nature, vol. 411, p. 907.

[LiveJournal] http://www.LiveJournal.com

[Kadushin'05] C. Kadushin, 2005, "Who Benefits from Network Analysis: Ethics of Social Network Research", Social Networks, vol. 27, p. 139.

[KarypisK'98] G. Karypis, and V. Kumar, 1998, "Multilevel k-Way Partitioning Scheme for Irregular Graphs", Parallel Distributed Computation, vol. 48, p96.

[KarypisK'99] G. Karypis, and V. Kumar, 1999, "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs", SIAM J. on Scienti c Comp., vol. 20, p. 359.

[KayehR'00] A. Kaveh, and B. H. A. Rahimi, 2000,"A Hybrid Graph-Genetic Method for Domain Decomposition", Computational Engineering using Metaphors from Nature, p. 127.

[KernighanL'70] B. Kernighan, and S. Lin, 1970, "An Efficient Heuristic Procedure for Partitioning Graphs", The Bell System Technical Journal, vol. 49, No. 2, pp. 291-307.

[King'99] V. King, 1999, "Fully Dynamic Algorithms for Maintaining All-Pairs Shortest Paths and Transitive Closure in Digraphs", Proceedings of IEEE Symposium on Foundations of Computer Science, p. 81, 1999.

[KoseWLF'01] F. Kose, W. Wechwerth, T. Linke, and O. Fiehn, 2001, "Visualizing Plant Metabolomic Correlation Networks Using Clique-Metabolite Matrices", Bioinformatics, vol. 17, pp. 1198-1208.

[Krebs'06] V. Krebs, 2006, How to do Social Network Analysis, [Online], Available: http://www.orgnet.com/sna.html

[KuroseR'01] J. F. Kurose, and K. W. Ross, 2001, Computer Networking, A Top-Down Approach Featuring the Internet, Addison Wesley Longman.

[Marsden'90] P. V. Marsden, 1990, "Network Data and Measurement", Annual Review of Sociology, vol. 16, pp. 435.

[Marsden'02] P. V. Marsden, 2002, "Egocentric and Sociocentric Measures of Network Centrality", Social Network, vol. 24, p. 407.

[McHugh'90] J. A. Mchugh, 1990, Algorithmic Graph Theory, Prentice-Hall, Englewood Cliffs, New Jersey.

[Milgram'67] S. Milgram, 1967, "The Small World Problem", Psychology Today, vol. 2, p. 60.

[MySpace] http://www.myspace.com

[Newman'00] M. E. J. Newman, 2000, "Models of the Small World", Journal of Statistical Physics, vol. 101, p. 819.

[Newman'01] M. E. J. Newman, 2001, "The Structure of Scientific Collaboration Networks", Proceedings of the National Academy of Sciences of the United States of America, vol. 98, p. 404

[Newman'03a] M. E. J. Newman, 2003, "The Structure and Function of Complex Networks", SIAM Review, vol. 45, p. 167.

[Newman'03b] M. E. J. Newman, 2003, "Ego-Centered Networks and the Ripple Effect", Social Networks, vol. 25, p. 83.

[Newman'05] M. E. J. Newman, 2005, "A Measurement of Betweenness Centrality Based on Random Walks", Social Networks, vol. 27, p. 39.

[NooyMB'05] W. Nooy, A. Mrvar, and V. Batagelj, 2005, Exploratory Social Network Analysis with Pajek, Cambridge University Press.

[NowellNKRT'05] D. L. Nowell, J. Novak, R. Kumar, P. Raghavam, and A. Tomkins, 2005, "Geographic Routing in Social Networks", Proceedings of the National Academy of Sciences of the United States of America, vol. 102, no. 33, p. 11623.

[Price'65] D. J. de S. Price, 1965, "Networks of Scientific Papers", Science, vol. 149, p. 510.

[Quinn'03] M. J. Quinn, 2003, Parallel Programming in C with MPI and OpenMP, McGraw-Hill, Inc.

[Redner'98] Redner, 1998, How Popular is Your Paper? An Empirical Study of the Citation Distribution, European Physical Journal B, vol. 4, p 131

[Ressler'06] S. Ressler, 2006, "Social Network Analysis as an Approach to Combat Terrorism: Past, Present, and Future Research", Homeland Security Affairs, vol. II, Issue 2.

[RichardsS'03] W. D. Richards, and A. J. Seary, MultiNet, Version 4.38 for Windows, Burnaby: Simon Fraser University.

[SantosSW'99] E. Jr. Santos, S. E. Shimony, and E. M. Williams, 1999, "Solving Hard Computational Problems through Collections (Portfolios) of Cooperative Heterogeneous Algorithms", Proceedings of the 12th International FLAIRS Conference, p. 356.

[Santos'01] E. Jr. Santos, 2001, "A Computational Model for Portfolios of Cooperative Heterogeneous Algorithms for Discrete Optimization", Proceedings of the 14th International FLAIRS Conference, p.525-529.

[SantosPA'07] E. E. Santos, L. Pan, and D. Arendt, 2007, "Case Studies for Anytime, Anywhere Methodologies in Social Network Analysis", Tech Rept LCID-07-115, Laboratory for Computation, Information & Distributed Processing, Virginia Polytechnic Institute & State University.

[SantosPAXP'06] E. E. Santos, L. Pan, D. Arendt, H. Xia, and M. Pittkin, 2006, "An Anytime Anywhere Approach for Computing All Pairs Shortest Path for Social Network Analysis", Integrated Design and Process Technology.

[SantosPAP'06] E. E. Santos, L. Pan, D. Arendt, and M. Pittkin, 2006, "An Effective Anytime Anywhere Parallel Approach for Centrality Measurements in Social Network Analysis", IEEE Systems, Man, & Cybernetics Society.

[Scott'92] J. Scott, 1992, Social Network Analysis, Newbury Park CA: Sage.

[SoperWC'04] A. J. Soper, C. Walshaw, and M. Cross, 2004, "A Combined Evolutionary Search and Multilevel Optimization Approach to Graph-Partitioning", Journal of Global Optimization, vol. 29, p. 225.

[Strogatz'01] S. H. Strogatz, 2001, "Exploring Complex Networks", Nature, vol. 410, p. 268.

[TomitaTT'04] E. Tomita, A. Tanaka, and H. Takahashi, 2004, "The Worst-Case Time Complexity for Generating All Maximal Cliuqes", Proceedings, Computing and Combinatorics Conference, 2004.

[WassermanF'94] S. Wasserman, and K. Faust, 1994, Social Network Analysis, Methods and Applications, Cambridge University Press.

[WattsS'98] D. J. Watts, and S. H. Strogatz, 1998, "Collective Dynamics of 'Small-World'Networks", Nature, vol. 393, p. 440.

[WinemillerL'03] K. O. Winemiller, and C. A. Layman, 2003, "Pattern, Process and Scale in the Food Web Paradigm: Moving on the Path from Abstraction to Prediction", Presentation at the 3rd Decade Food Web Symposium, Giessen, Germany, November 2003, Available Online: http://wfsc.tamu.edu/winemiller/lab/Foodweb_Symp.ppt.

[ZhangABCLS'05] Y. Zhang, F. N. Abu-Khazam, N. E. Baldwin, E. J. Chesler, M. A. Langston, and N. F. Samatova, 2005, "Genome-Scale Computational Approaches to Memory-Intensive Applications in Systems Biology", Proceedings of 2005 ACM/IEEE conference on Supercomputing.

[Zwick'04] U. Zwick, 2004, "A Slightly Improved Sub-Cubic Algorithm for the All Pairs Shortest Paths Problem with Real Edge Lengths," Proceedings of the 15th International Symposium on Algorithms and Computation (ISAAC), Lecture Notes in Comput. Sci., 3341, Springer, New York, p. 921.