Graph Clustering using Treaps and Deaps

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering
in
Computer Science and Engineering

Submitted By
Dharya Arora
(Roll No. 801032008)

Under the supervision of:
Dr. Shalini Batra
Assistant Professor

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004

June 2012

CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled, "*Graph Clustering using Treaps and Deaps*", in partial fulfillment of the requirements for the award of degree of Master of Engineering in *Computer Science and Engineering* submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of *Dr. Shalini Batra* and refers other researcher's work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

Dharya Arora

(Dharya Arora)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

(Dr. Shalini Batra)
Assistant Professor,
Computer Science and Engineering Department,
Thapar University, Patiala

Countersigned by

(Dr. Maninder Singh)
Head
Computer Science and Engineering Department
Thapar University
Patiala

(Dr. S. K. Mohapatra)
Dean (Academic Affairs)
Thapar University
Patiala

Acknowledgement

No volume of words is enough to express my gratitude towards my guide **Dr. Shalini Batra**, Department of Computer Science & Engineering, Thapar University, Patiala, who has been very concerned and has aided for all the materials essentials for the preparation of this thesis report. She has helped me to explore this vast topic in an organized manner and provided me all the ideas on how to work towards a research-oriented venture.

I am also thankful to **Dr. Maninder Singh**, Head of Computer Science & Engineering Department and **Dr. Inderveer Channa**, P.G. Coordinator, for the motivation and inspiration that triggered me for the thesis work.

I would also like to thank the staff members and my colleagues who were always there at the need of hour and provided with all the help and facilities, which I required, for the completion of my thesis work.

Most importantly, I would like to thank my parents and the almighty for showing me the right direction out of the blue, to help me stay calm in the oddest of the times and keep moving even at times when there was no hope.

Dharya Arora

(801032008)

Abstract

With the advent of the Internet, Social networks have grown enormously and Social Network Analysis (SNA) has come up as an important field for research. Social networks are represented as graphs and the fundamental component of SNA is the relationship defined by linkages among units or nodes in the network. Since graphs in social networks comprise of large number of vertices and edges, adjacency matrix is considered to be an effective and efficient technique to represent them. The intent of this thesis is to cluster the graphs, optimize the graph storage and mapping without using a large adjacency matrix to represent a large graph.

A special data structure Treap, a combination of binary search tree and heaps has been used as a replacement to a large adjacency matrix. It has been experimentally evaluated that the proposed approach significantly improves the space occupied by adjacency matrix and helps the graph to grow dynamically without affecting the current data structure.

Once the graph or social network is optimally stored, clusters are generated, based on some probabilistic mathematical models and heuristic approaches. After the graph clustering is done efficiently, the next task of this thesis is to store and map the clustering information in an efficient format so that storage space is optimized and access time is less.

A special kind of data structure ClusteredDeap inherited by Deaps data structure is used, in which dynamically generated array of linked list is properly mapped. It has been experimentally proved that access time for various operation such as insert, delete and traverse is significantly reduced using such data structure.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

The enormous growth of Internet and the development of new applications and services have dramatically increased the number of users, resulting in increased storage size [1].
A social network is a set of relationships among interacting social entities. Although the size of the social networks has grown exponentially, no standard method has been designed for efficient mapping of a graph or social network onto a compatible data structure. As internet in itself is a social network graph comprises of nodes and edges so with the increased size of graph day by day directly affect the storage being used, such as adjacency matrix [2, 3]. Data structures used for adjacency matrices usually have two components an array that stores all the entries of the matrix and pointer to an array which would take care of increased size of entries in it [4].

1.1 Introduction to Network

Networks can range from small sociograms such as those introduced by Moreno [5] with only a few units to huge networks with practically billions of units, for example, a network of all computers connected to the Internet or a network of all people and their relationship or attributes. Units can be people, organizations, countries, words *etc.* and for each of these units there are a number of possible ties between pairs of these units. Commonly available networks are networks among people (friendship or communication), trade networks among organizations, countries, citation networks, genealogies, organic molecules in chemistry, ties among words in text, transportation networks *etc.*

1.2 Introduction to Social Network

Social network is a map of all of the relevant ties between all the nodes being studied, defined by graphs representing social relationships between people or organizations. These concepts are often displayed in a social network diagram, where nodes are the points and ties are the lines. Example includes email communication networks [6], instant messenger networks [7], mobile call networks [8], and friendship networks [9]. Other

forms of complex network, like co authorship or citation networks [10], biological networks, metabolic pathways, genetic regulatory networks, food web and neural networks etc. are also examined and demonstrate similar patterns [11]. Each node also called an actor or vertex in a graph represents an individual person or a group of person. An edge connecting two nodes called a tie represents relationship between the objects represented by these two nodes as shown in Figure 1.1.

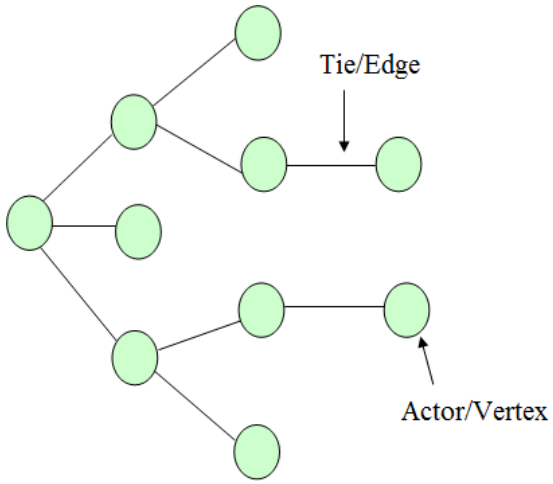

Figure 1.1: A Social Network

Using graphs to represent social data enables social analysts to completely and rigorously describe and analyze the structural information embedded in social relationships.

In general 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, attribute and other information or knowledge processing entities.

1.3 Types of Social Network

Basically two kinds of social networks are investigated by social scientists: egocentric networks and socio-centric networks. Egocentric networks are connected with a single individual (the ego) and his or her social milieu expressed in formal and informal relations like kinship, friends, acquaintances, etc. Socio-centric networks on the other hand also called whole or complete networks are networks defined within a social system

like the friendship relations in a classroom, a network of relations between workers and executives inside an organization; or relations between formal partner organizations. So depending upon the application used, a social network can be categorized into various types. As any real world problem can be mapped to a graph problem so a social network which is nothing but a graph having set of nodes and vertices can have various types. For example, cities can be treated as nodes and the path from one city to another can be treated as edge so the average flow of transport from one city to another can be treated as a weighted edge in a graph. 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. So in short a social network can be weighted, unweighted, homogeneous, heterogeneous and many more. But in this thesis the main focus is given on unweighted and homogeneous relational network.

1.3.1 Example of Social Network

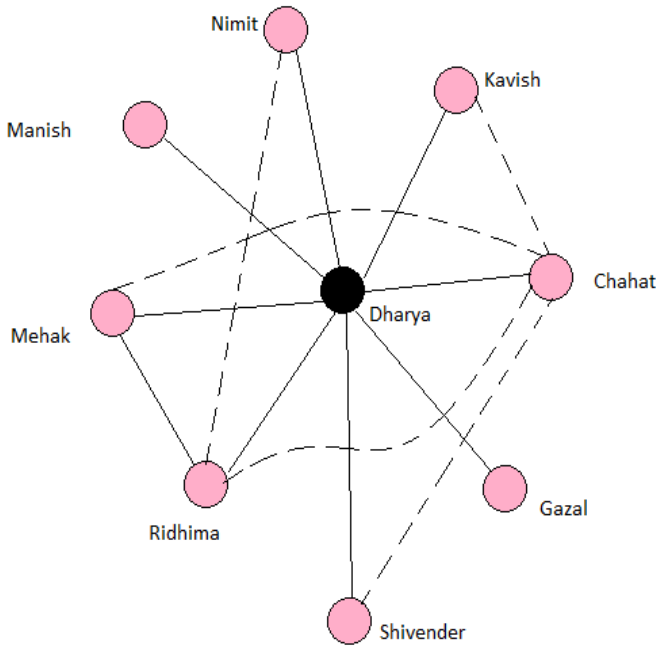

Figure 1.2: A Friendship Network

In Figure 1.2 a real time example of a social networking site has been considered providing a small view of friendship network. As seen in the Figure 1.2, persons Dharya, Chahat, etc. are friends which would be treated as nodes of graph and there relationship is

treated as an edge of graph. A homogeneous relationship with unweighted edges is considered here assuming black node as the central node and dotted arrows show the friendship relationship but it is between those nodes who are friend of Dharya.

1.4 Social Network Analysis

Social Network Analysis (SNA) [12, 13, 14] is the study of relations between individuals including the analysis of social structures, social position, role analysis, and many others. A graph G (V, E) is made of two sets (V: set of vertices E: set of edges) and analyzing the nature of relationships and connections between entities is a key towards understanding a variety of phenomena. A network can be constructed based on the response, with nodes representing individuals and edges the interaction between them. So any social network can be modeled in the form of graph in which the actor or people act as node or vertex of graph and the relationship between actors act as edge of graph. Graph structured instances have no natural representation as a single row of a single fixed-width table. SNA is based on an assumption of the importance of relationships among interacting units or nodes. These relations defined by linkages among units or nodes are a fundamental component of SNA.

Mining problems for graph data includes applying techniques such as frequent pattern mining, clustering [15] and classification [16]. These methods are much more challenging in the graph domain because the structural nature of the data makes the intermediate representation and interpretability of the mining results much more challenging.

Social network analysts argue that causation is not located in the individual but in the social structure. While people with similar attributes may behave similarly, explaining these similarities by pointing to common attributes misses the reality that individuals with common attributes often occupy similar positions in the social structure. Their similar outcomes are caused by the constraints, opportunities, and perceptions created by these similar network positions.

The scientific study of social networks has been ongoing for decades but in the past few years it has seen tremendous growth in its application and publicity. Social network analysis is an emerging area of research for computer scientists as it employs different

concepts from graph theory, probability, and statistics to solve problems in a wide range of disciplines. Since social network analysis can be performed on many real world networks from different domains. A number of social network analysis methods have been designed for various tasks.

1.5 Social Network Analysis Components

In recent times, the computer revolution has provided scholars with a huge amount of data and computational resources to process and analyze these data. For the analysis of a social network the main component that must be studied out is Community detection also known as clustering. Communities, also called clusters or modules, are groups of vertices which probably share common properties and/or play similar roles within the graph. In Figure 1.3 a schematic example of a graph with communities is shown where it has three communities, enclosed by the dashed circles.

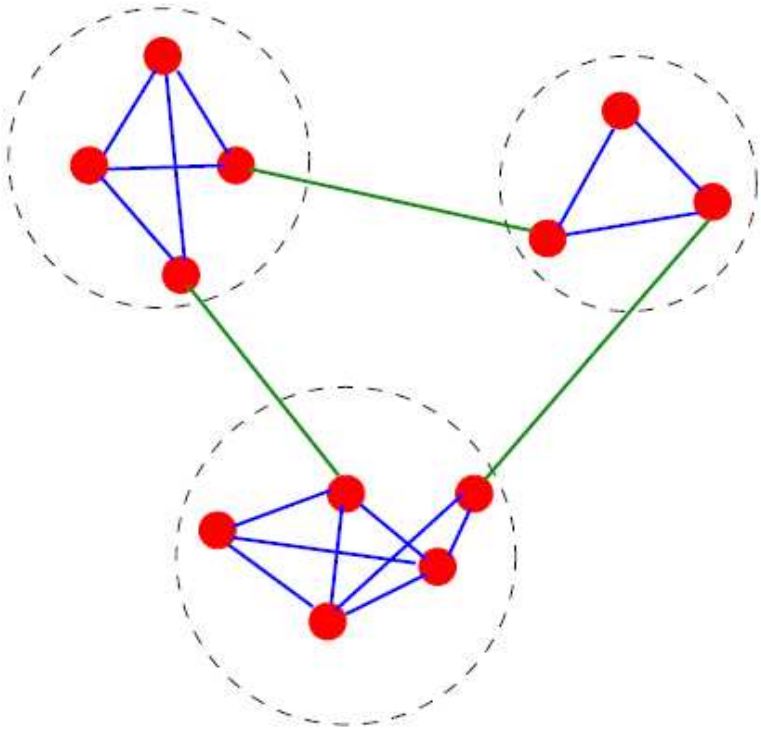

Figure 1.3: A Random Clustered Graph

Society offers a wide variety of possible group organizations: families, working and friendship circles, villages, towns, nations. The diffusion of Internet has also led to the creation of virtual groups that live on the Web, like online communities.

Normally data for huge graphs is stored in the form of adjacency matrix. One of the major problems associated with the use of adjacency matrix is that it has static array allocation and fixed entries for increased size of network creates problem during insertion. Secondly, dynamic array allocation requires more time to create a new array of increased size and then move the entries from previous array to new array and then deallocating the previous array [17, 18]. Finally, although the adjacency matrix for a graph is sparse, every null entry would take space. For dynamically increased network and efficient storage optimization [19, 20, 21, 22] this thesis proposes the use of Treaps to store a graph with its mapping information in an efficient manner [23, 24, 25].

1.5.1 Social Network Analysis tasks

i. Centrality analysis: aims to identify the “most important” actors in a social network. Centrality is a measure to calibrate the “importance” of an actor. This helps to understand the social influence and power in a network.
ii. Community detection: Actors in a social network form groups. This task identifies these communities through the study of network structures and topology.
iii. Position/Role analysis: identifies the role associated with different actors during network interaction. It serves as the bridge between two groups.
iv. Network modeling: attempts to simulate the real-world network via simple mechanisms such that the patterns presented in large-scale complex networks can be captured. Information diffusion studies how the information propagates in a network.
v. Information diffusion: also facilitates the understanding the cultural dynamics, and infection blocking.
vi. Network classification and outlier detection: Some actors are labeled with certain information. For instance, in a network with some terrorists identified, is it possible to infer other people who are likely to be terrorists by leveraging the social network information.

vii. Viral marketing and link prediction: The modeling of the information diffusion process, in conjunction with centrality analysis and communities, can help achieve more cost-effective viral marketing. That is, only a small set of users are selected for marketing. Hopefully, their adoption can influence other members in the network, so the benefit is maximized.

1.5.2 Graph Clustering

Graph clustering is the process of organizing objects into groups whose members are similar in some way. A cluster is therefore a collection of objects which are "similar" between them and are "dissimilar" to the objects belonging to other clusters [26, 27]. If there is a friendship network then through clustering it would be easy to find out the related group of people and also community detection becomes very easy. Similarly if there is a transportation network then using clustering it becomes very easy to analyze the traffic between various cities and using centrality property [28] of clustering, possible actions can be taken to improve the transportation network.

1.6 Structure of the Thesis

The rest of the thesis is organized in the following order:

Chapter 2 – Literature Review: This chapter reviews social network, Treap and Deap data structures for storage and mapping of clustering for analyzing social networks.

Chapter 3 - Problem Statement: It states the problem and provides the methodology used to solve it.

Chapter 4 - Graph Clustering using Treaps and Deaps: This chapter gives a detailed introduction about Treap data structure used to optimize graph storage and Algorithm used to do graph clustering and finally technique to optimally map the clustered information obtained using ClusteredDeap data structure.

Chapter 5 - Implementation Details: It includes the experiment performed using a real time example and the results evaluated.

Chapter 6 - Conclusion and Future Scope: It concludes the thesis and provides suggestions for future work. Thesis concludes with references and list of publications.

Chapter 2
Literature Review

The scientific study of social networks has been ongoing for decades but in the past few years it has seen tremendous growth in its application and publicity. Social network analysis is an emerging area of research for computer scientists as it employs different concepts from graph theory, probability, and statistics to solve problems in a wide range of disciplines. Similarly storage optimization has been expanding in all directions at an astonishing rate during the last few decades.

New algorithmic and theoretical techniques have been developed, the diffusion into other disciplines has proceeded at a rapid pace, and knowledge of all aspects of the field has grown even more profound. At the same time, one of the most striking trends in optimization is the constantly increasing emphasis on the interdisciplinary nature of the field. Optimization today is a basic research tool in all areas of engineering, medicine, and the sciences. The decision-making tools based on optimization procedures are successfully applied in a wide range of practical problems arising in virtually any sphere of human activity.

Depending on the nature of the problem, different techniques can be used to formulate and solve a typical optimization storage problem. Various data structure such as Treaps, Deaps deals with optimization problems, in which the objective and constraints can be formulated using only functions that are linear with respect to the decision variables. In nonlinear optimization, one deals with optimizing a nonlinear function over a feasible domain described by a set of, in general, nonlinear functions.

The pioneering works on the gradient projection method by J. B. Rosen [29] generated a great deal of research enthusiasm in the area of Treaps, resulting in a number of new techniques for solving large-scale dynamic graphs. This research resulted in several powerful nonlinear optimization software packages, including Lancelot [30].

In many optimization problems such as cluster mapping and storage, as well as other applications, the input data, such as demand or cost, are stochastic. In addition to the difficulties encountered in deterministic optimization problems, the stochastic problems

introduce the additional challenge of dealing with uncertainties. To handle such problems, one needs to utilize probabilistic methods alongside optimization techniques. This led to the development of a new area called Treap data structure, whose objective is to provide tools to help design and control stochastic systems with the goal of optimizing their performance.

Due to the large size of most practical optimization problems, especially of the stochastic ones, the so-called decomposition methods were introduced. Decomposition techniques [31] are used to subdivide a large-scale problem into sub problems of lower dimension, which are easier to solve than the original problem. The optimal solution of the large problem is then found using the optimal solution of the sub problems. These techniques are usually applicable if the problem at hand has some special structural properties. For example, the Dantzig-Wolfe decomposition method [32] applies to linear programs with block diagonal or blocks angular constraint matrices. Another popular technique used to solve large-scale linear programs of special structure is Benders decomposition [33]. One of the advantages of using Treap data structure is that they can be easily simulated and implemented in dynamic and large scale computing environments of social networks.

As a result of ongoing enhancement of the optimization methodology and improvement of available computational facilities, the scale of the problems solvable to optimality is continuously rising. However, many large-scale optimization problems encountered in practice cannot be solved using traditional optimization techniques. A variety of new computational approaches, called heuristics, have been proposed for finding good suboptimal solutions to difficult optimization problems.

A heuristic in optimization is any method that finds an “acceptable’’ feasible solution. Many classical heuristics are based on local search procedures, which iteratively move to a better solution (if such solution exists) in a neighborhood of the current solution. A procedure of this type usually terminates when the first local optimum is obtained. Randomization and restarting approaches used to overcome poor-quality local solutions are often ineffective. More general strategies known as meta-heuristics usually combine some heuristic approaches and direct them towards solutions of better quality than those found by local search heuristics. Heuristics and meta-heuristics play a key role in the

solution of large, difficult, applied optimization problems. Sometimes in searching for efficient heuristics people turn to nature, which seems to always find the best solutions.

In recent decades, new types of optimization algorithms have been developed and successfully tested, which essentially attempt to imitate certain natural processes. The natural phenomena observed in annealing processes, nervous systems, and natural evolution were adopted by optimizers and led to the design of simulated annealing [34], neural networks [35], and evolutionary computation methods in the area of optimization. The ant colony optimization method is based on the behavior of natural ant colonies. Other popular meta-heuristics include greedy randomized adaptive search procedures (GRASP). Some of the previous research [36] attempted to address the question of whether right-of-way should be acquired early.

The origin of graph theory dates back to Euler's solution of the puzzle of Konigsberg's bridges in 1736 [37]. Since then a lot has been learned about graphs and their mathematical properties [38]. In the 20th century they have also become extremely useful as the representation of a wide variety of systems in different areas. Biological, social, technological, and information networks can be studied as graphs, and graph analysis has become crucial to understand the features of these systems. For instance, social network analysis started in the 1930's and has become one of the most important topics in sociology [39, 40].

Social network analysis goes back to Jacob L. Moreno and his psychodrama studies in the 1930's [41]. He was among the first that operationalized the concept of social network and represented interpersonal relations in small groups using graphical methods in order to visualize channels of information. This visual device was called sociograph and the branch that describes and analyzes this kind of network configurations was denominated sociometry. By the end of the 1940's important advances were made in the research of the structural properties of networks.

As research in this area grew, network analysis was distinguished from traditional social science by the dyadic nature of the standard data set. These dyadic attributes (social relation) may be represented in matrix from by a square 1-mode matrix. But the data in traditional social science are represented as 2-mode matrices. However network analysis is not completely divorced from traditional social science and often has occasion to

collect and analyze 2-mode matrices. Some of the methods developed in network analysis are used in analyzing non-network data. In [42] authors discussed ways of applying and interpreting traditional network analytic techniques to 2-mode data and further presented an idea on developing new techniques.

In Patrick Doreian *et al.* in [43] examined graph clustering procedures reviewing both structural equivalence and regular equivalence. Realized that few empirical examples of exact partitioning exist, it was argued that the lack of fit between model and reality can be measured and used as a way of comparing the adequacy of different models. This idea was combined this with a generalization of the clustering method that permits many types of models to be constructed and compared. Sets of permitted ideal blocks were constructed and the model that shows minimum inconsistency is sought.

Batagelj Doreian and Ferligoj [44] developed a generalized approach to community detection and methods where a set of observed relations are applied to a pre specified cluster. This generalized community detection approach was implemented in program Pajek.

Patrick Doreian & Vladimir Batagelj [45] extend the direct approach for graph clustering one-mode data to two-mode data. The idea is that the rows and columns are partitioned simultaneously but in different ways. Many but not all of the generalized block types can be mobilized in block modeling two-mode network data. These methods were applied to some voting data from the 2000–2001 terms of the Supreme Court and to the classic Deep South data on women attending events. The insight that rows and columns can be partitioned in different ways can also be applied to one-mode data.

There are many circumstances in which binary relations are defined between pairs of objects in sociology. For example, there are social relations between people in business, there are trading relations between firms, and in design there are functional dependencies between components. In all of these situations, the clustering of objects into densely interconnected blocks discovers the actual structure of the system. Alan Jessop [46] presented a method which permits the construction of blocks to be formulated as a quadratic programme.

After going through various research proposals in the area of analyzing social network, it was realized that community detection or clustering can be used for analyzing complex

data as well as dynamic scalable graph as it is based on the idea that units in a network can be grouped according to the extent to which they are equivalent under some meaningful definition of equivalence. To initially store a dynamic scalable graph and for optimal storage of communities, Treaps and Deaps are some of the best models to use.

So the whole thesis work can be grouped into three basic steps:

1. Consider a graph or social network and optimally store it using Treap data structure
2. Generate a new graph clustering technique for social network analysis
3. Once the clusters are obtained, they would be stored and properly mapped using ClusteredDeap data structure

Chapter 3

Problem Statement

3.1 Problem Definition

With the growth of Internet, the size of social network is growing enormously i.e. for any real time problem that can be easily modeled to a graph problem must have an important parameter known as scalability. For increasing networks with the increase in storage size results in a new domain of study such as optimal storage techniques in which a problem can be easily formulated and mapped to compatible data structure efficiently. So formulation of dynamic graph storage as well as the study of social network provides a domain analysis of graph clustering in which the first problem is to look for a quantitative definition of community detection.

1. A data structure Treap, a combination of binary search tree and heap has been used to optimally store a large and scalable graph without using a static adjacency matrix.
2. After the storage is done effectively, graph clustering is gone using heuristic and approximation algorithms. This will overcome the problem of analyzing the social network using tradition approaches such as hierarchical clustering, graph partitioning, partitional clustering and spectral clustering.
3. Once the clustering is done using heuristic techniques, ClusteredDeap data structure has been used for storing the graph clusters and related mapping information. This would lead to efficient mapping and storage and overcome the problem encountered in linked list or any other such type of data structures.

3.2 Methodology

The step-by-step methodology to be followed in storing, designing and analyzing of a social network is given below:

- Design of Treap data structure for storing a random graph of social network which is scalable in nature.
- Designing of heuristic and approximation algorithms of graph clustering for SNA.

- Designing of efficient and compatible data structure such as Deap for storing clusters produced by the result of heuristic algorithms applied for analyzing social network.

Chapter 4

Graph Clustering using Treaps & Deaps

4.1 Introduction to Treaps

A *treap* is a binary search tree in which each node has both a key and a priority [47]. Nodes are ordered in an in-order fashion by their keys and are heap-ordered by their priorities [48]. The basic idea behind Treaps was to balance binary search trees [49] and to make optimal binary search tree so that in worst case as well, its amortized cost would never exceed O (log n). The Treap and the randomized binary search tree are two closely related forms of binary search tree data structures that maintain a dynamic set of ordered keys and allow binary searches among the keys. After any sequence of insertions and deletions of keys, the shape of the tree is a random variable with the same probability distribution as a random binary tree; in particular, with high probability its height is proportional to the logarithm of the number of keys or nodes, so that each search, insertion, or deletion operation takes logarithmic time to perform [50].

The basic traditional method to store any social network or graph is to use adjacency matrix. But for scalable network it would not give optimal results. For example as seen in Figure 1.2, if adjacency table (A) is considered then it would form n*n matrix (n being the total no. of nodes in the graph) where A_{ij}=1 if there is a direct edge between i and j else A_{ij}=0. But for large number of nodes maintaining adjacency matrix would take more space, so Treaps are used for optimization of space.

4.2 Graph Storage using Treaps

Let G be a graph comprises of two main components (V, E) where V contains a set of vertices of graph and E contains a set of edges of graph. To store the whole graph corresponds to its mapping information let's consider a graph having few nodes as shown in Figure 4.1.

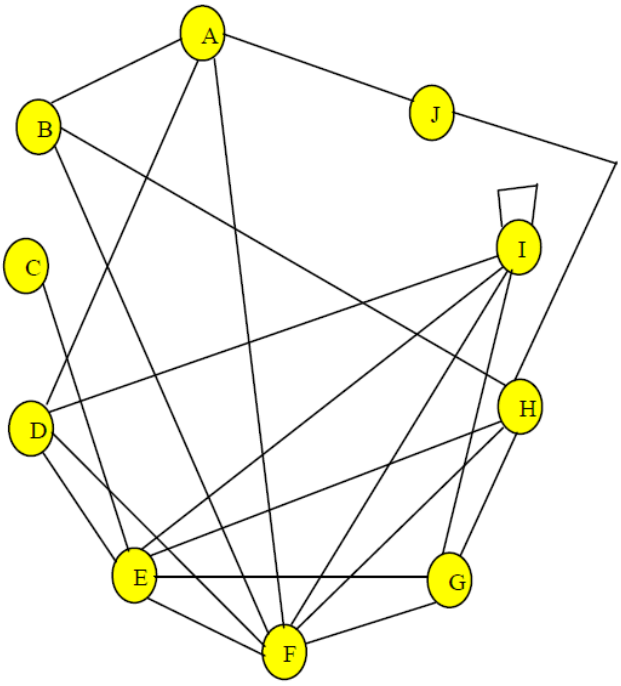

Figure 4.1: A random graph

The corresponding adjacency matrix for the graph as shown in Figure 4.1 is represented in Figure 4.2. If the given adjacency matrix is used to store and map the graph information the storage schema used is not scalable for dynamic graph approach. But as the applications are increasing day by day where graph has a strong impact to represent a problem domain, using static allocation pays no attention and cause problems during dynamic allocation.

To implement the concept, consider the corresponding adjacency matrix of the graph.

0	1	0	1	0	1	0	0	0	1
1	0	0	0	0	1	0	1	0	0
0	0	0	0	1	0	0	0	0	0
1	0	0	0	1	1	0	0	1	0
0	0	1	1	0	1	1	1	1	0
1	1	0	1	1	0	1	1	1	0
0	0	0	0	1	1	0	1	1	0
0	1	0	0	1	1	1	0	0	1
0	0	0	1	1	1	1	0	1	0
1	0	0	0	0	0	0	1	0	0

Figure 4.2: Adjacency matrix of a graph for Figure 4.1

Each row of matrix depicts the adjacency information of each node of graph respectively. Since Treap require a key and a priority, key is taken from the unique name of a node and priority would be the no. of links corresponding to each node i.e. no. of 1's to its row is its priority as depicted in Table 4.1.

Table 4.1: Adjacency table

A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**
4	3	1	4	6	7	4	5	5	2

4.2.1 Structure of a node

Each node of Treap has a specific internal representation. Each node will store the information about its key and priority, pointers to left and right child and a pointer to an array containing the name of its adjacent node such as in Figure 4.3:

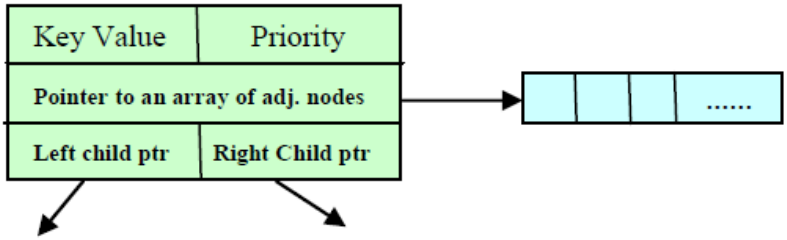

Figure 4.3: Node representation of a Treap

4.2.2 Creation of Treap for Graph storage

Various operations in a Treap can be performed such as insert new node, delete a node, traverse a node. For creation of Treap, 'n' nodes would recall insert Algorithm:

4.2.2.1 Algorithm for insertion of a node

To insert a node in a Treap:

Input:

- Key value of node as "key"
- Priority of node as "pr"
- Root node of Treap as "root"

Output:

- Insert the new node to its appropriate position by treap rotations.

```
1.  Set new ← Getnode() // Create an empty new node
2.  Set new→key=key
3.  Set new→pr=pr
4.  if root = NULL then
5.          Set root=new
6.          return
7.  Set ptr=root
8.          Repeat  while ptr ≠ NULL do
9.                  Set prev=ptr
10.                 if new→key > ptr→key then
11.                    ptr=ptr→right
12.                 side=right
13.                 else
14.                  ptr=ptr→left
15.                  side=left
16. Set prev→side=new
17. Set ptr=new→parent
18. Repeat while new→pr < ptr→pr do
19.         temp=ptr
20.         ptr=new
21.         if temp→key < ptr→key then
22.           prev=ptr→left
23.           ptr→left=temp
24.           if prev→key < temp→key then
25.                 temp→left=prev
26.           else
27.                 temp→right=prev
28.         else
29.           prev=ptr→right
30.           ptr→right=temp
31.           if prev→key < temp→key then
32.                 temp→left=prev
33.           else
34.                 temp→right=prev
35. go to step 16 until new→key < new→parent→key
36. return
```

Algorithm 1: Insert a node in a Treap

Analysis:

As Treap is a combination of Binary search tree and Heap so in every step it would cost log n time to put a node to its corresponding position after rotation. So when

combinations of ‘n’ nodes are inserted in a sequence then the total cost would be: **O (n*log n).**

4.3 Clustering using Probabilistic method and Heuristic approach

After the storage is done successfully, the next task is to generate clusters. Consider a small graph, Figure 4.4, depicting a friendship network consist of some nodes and edges where each node represent a people and the edges between two nodes represent that two people are friend of each other. So this graph would be bidirectional and unweighted and follow homogeneous relationship [51].

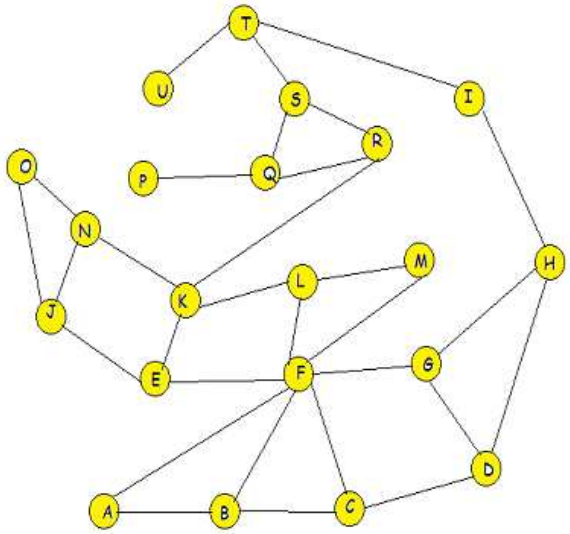

Figure 4.4: A random graph depicting a Social network of Friends

For the sake of simplicity node numbers have been considered as alphabets which could otherwise be any unique email id or IP address or any unique user name depends upon the application being used.

4.3.1 Grouping of nodes in a network

Clustering (grouping) of nodes is done using the Algorithm give below:-

Input: Node name whose set is to be created as $node_i$ and no. of its adjacent links as $nlinks_i$ and a temporary array to store the result as arr_i
Output: This Algorithm will create set for each and every node of graph and store the result into their corresponding array as arr_i

```
1.    foreach i←1 to n do
2.     add all adjacent nodes of node_i to arr_i
3.       foreach k←1 to nlinks_i do
4.          add all unique adjacent   nodes of  node_arr_i[k] to arr_i
5.          remove node_i from arr_i
6.       end
7.    end
```

Algorithm 2: Create Set Algorithm

4.3.2 Calculating NearByFactor (NBF) of a node

Once the set for each node is created, every node is assigned one value called NearByFactor (NBF). Nodes having same NBF need not come in same cluster i.e. same NBF value nodes may form different cluster but different NBF value node must form different cluster.

Algorithm to assign NBF value to each node is given below:-

Input: Node name whose set is to be created as $node_i$ and their set which was just created in Algorithm 1 as $node_i$.set and total no of nodes in the graph as N.
Output: This Algorithm will assign NBF to each and every node of graph and store the result into their corresponding variable as $node_i$.NBF

```
1.    Set t=1
2.    foreach i←1 to N do
3.       node_i.NBF=t
4.    end
5.    foreach i←1 to N do
6.       foreach j←i+1 to N do
7.          if node_i.NBF=node_j.NBF AND node_i.set ∩ node_j.set = Ø
8.          then    node_j.NBF=node_j.NBF+1
9.       end
10.   end
```

Algorithm 3: Create NBF Algorithm

4.3.3 Finding importance of a node in the network

A node is stronger if its network is more dispersed, so importance of a node in the graph is calculated by a factor called multiplication factor (MF).

$$MF = \frac{No.of\ direct\ links}{Max.\ links\ that\ it\ can\ have} \quad (1)$$

4.3.4 Determining active participation of a node in the network

Participation of a node in a social network is determined by a value known as NodeExistenceFactor (NEF). It actually calculates the usage and active participation of a node in a social network makes i.e. a node with more NEF will have more impact on a strong clustering group than others.

Grouping of node into different clusters is done by calculating the Total Value (TV) for each node:

$$TV_{node_i} = \left(\sum_{j=1}^{n} MF\ of\ nodes\ in\ set\ of\ node_i\right) * NEF_{node_i} \quad (2)$$

In Eq. (2), n signifies the total no. of nodes present in set of a particular node.

4.3.5 Factors affecting the participation of any node in a network

Various factors would decide the value of NEF depending upon the application used. For example if there are friendship networks i.e. a social network graph, then NEF would depend upon:

1. Week days visibility
2. Daily hours visibility
3. No of friend request sent (average)
4. No of friend request received (average)
5. Pending friend request
6. Applications or Games used or created

So on the basis of application and requirement the above 6 factors can be modified, increased or decreased that influence NEF the most and value of each factor would be presented in Table 4.2.

Table 4.2: Factor influencing Scale

Not present	Less	Moderate	High
0	1	2	3

4.3.6 Calculation of threshold (Θ)

For calculating threshold (Θ) take an average given as:

$$\Theta_{NBF} = \frac{Sum\ of\ TV\ of\ any\ 2\ or\ 3\ nodes\ of\ same\ NBF}{2\ or\ 3} \quad (3)$$

Calculation of Θ: For the calculation randomly pick any two or three nodes of same NBF and take their average. After calculation of Θ_{NBF} nodes can be easily clustered as:

$Node_i$ of same NBF is part of: $\begin{Bmatrix} Strong\ Cluster\ if\ TV_i \geq \Theta_{NBF} \\ Weak\ Cluster\ elsewhere \end{Bmatrix}$ (4)

4.4 Introduction to Deaps

After the clusters are successfully created using heuristic techniques, the next step is to store and map those clusters into compatible data structure so that no loss of information would occur. So for purpose ClusteredDeap data structures are introduced which are inherited by Deaps data structure.

A Deap is a double-ended heap that supports the double-ended priority operations of insert, delete-min, and delete-max. Similar to min-max heap but Deap is faster on these operations by a constant factor, and the algorithms are simpler. A Deap is a complete binary tree that is either empty or satisfies the following properties:

(1) The root contains no element.

(2) The left subtree is a min heap.

(3) The right subtree is a max heap.

(4) If the right subtree is not empty, then let i be any node in the left subtree. Let j be the corresponding node in the right subtree. If such a j does not exist, then let j be the node in the right subtree that corresponds to the parent of i. The key in node i is less than or equal to that of j.

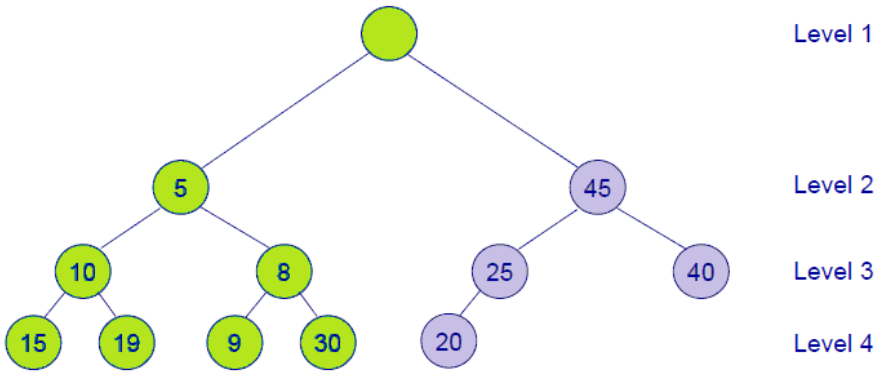

Figure 4.5: A Deap Example

But for storing the whole information into Deaps the basic data structure that is often used to implement is an array. Internal representation that works for Deap is an array data structure, where each node contains only one element. If heterogeneous information is suppose to be stored for each node then linked lists are used as array wouldn't have the capability to do the same. Each and every node is internally stored in the form of Structure. It has the capability to store heterogeneous items together. So a modified version of Deap can be used to map the whole graph with the clustered information known as ClusteredDeap. For n element ClusteredDeap, the min element is the root of min heap and the max element is the root of the max heap and if n = 1, then the min and max elements are the same and are in the root of the min heap. Since Deap is a complete binary tree, it may be stored as an implicit data structure in a one-dimension array similar to min, max, min-max heaps. In the case of ClusteredDeap, the position 1 of the array is not used as in Deap so for an n-element ClusteredDeap, it occupies n+1 elements of an array.

4.5 Mapping and Storage using ClusteredDeap

One needs to store the clustering information effectively so that it can be accessed easily and other operations like insert, delete, update, search, etc. can be done. Storage of information includes successful mapping of clustered information of each node with respect to other nodes and also storing the adjacency information with respect to each node. If adjacency matrix is chosen then access time is O(1) but for storing adjacency

information of nodes it would take more space i.e. $O(N^2)$ where N is the total no. of nodes in the graph. Secondly, if any operation like insert, delete occurs then adjacency matrix fails as it is static in nature. This thesis proposes creation of ClusteredDeap data structure for storing and mapping graph.

4.5.1 Structure of a node

Each node of ClusteredDeap has a specific internal representation. Each node will store the information about its node name, number of links that a node has to its adjacent nodes, processed links details and pointer to an array which keep track of its adjacent nodes details as shown in Figure 4.6.

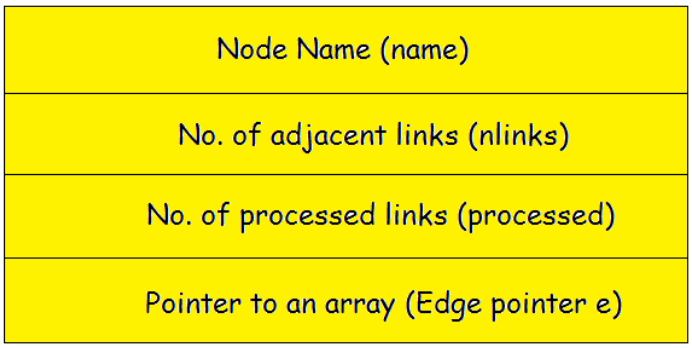

Figure 4.6: Internal memory representation of a node

4.5.2 ClusteredDeap Creation

For creating ClusteredDeap consider a Clustered graph as depicted in Figure 4.7; the information about the adjacent node information corresponding to Figure 4.7 is given in Table 4.3.

Table 4.3: Nodes with adjacent information

Node	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
No. of links	3	3	4	4	3	4	3	3	2	2	3	4	2	1	4	4	3	2	3	2	4	4	4	1	3	3

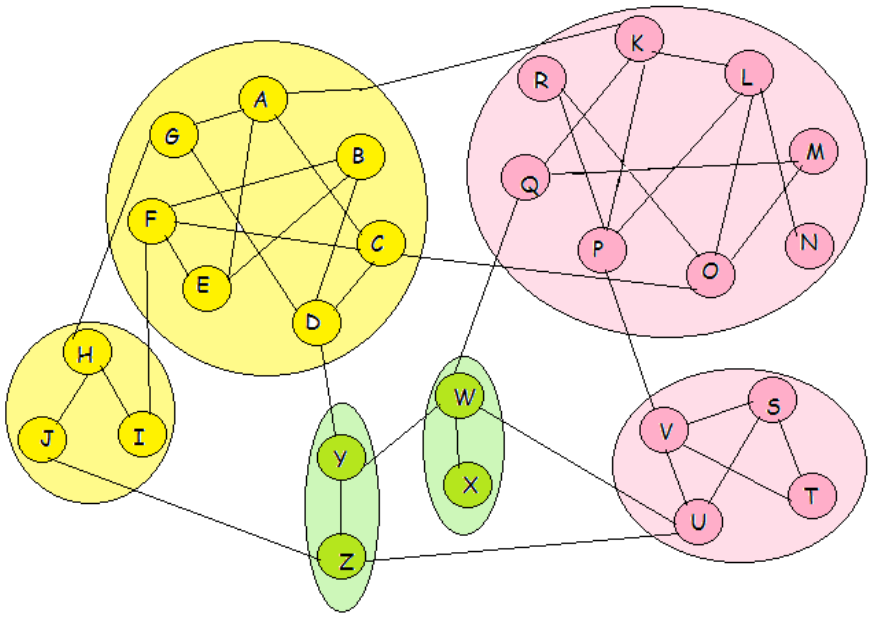

Figure 4.7: A random clustered graph

As there are 6 clusters and 3 relating groups. Here relating groups are those groups which are having same NBF value but lie in different clusters or community as computed by Algorithm 3, so total 3 ClusteredDeap will be formed.

4.6 Accessibility operations

Various operations in a ClusteredDeap can be performed such as insert new node, traverse a node and find an optimal path from one node to another node, *etc*.

4.6.1 Algorithm for insertion of a node

To insert a node in a ClusteredDeap:

Input: Total no. of relating groups in the cluster as 'tnbf' and New node's parameter that is going to insert such as name, no. of links etc.
Output: Insert the new node to its corresponding position with all adjacent nodes satisfied.

```
1.      foreach i←1 to 2*tnbf do
2.          if A[i].nbf = inbf then
3.              if ipart = 2 then
4.                  Set i=i+1
5.              end if
6.            t1=A[i].nlinks
7.            n1=Getnode(t1+1);  // Function used create node(s)
```

```
8.          foreach j←1 to t1 do
9.                n1[j]=A[j]       // Copy all entries of array A to new Array n1
10.         end for
11.         n1[t1+1].name=name
12.         n1[t1+1].nlinks=nlinks
13.         n1[t1+1].e=NULL
14.         A[i].nlinks=A[i].nlinks+1
15.         A[i].nptr=n1
16.         t=A[i].nptr[t1].nlinks
17.         foreach k←1 to t do
18.               ed=Getadjacentnode(1); //Function used to create a adjacent node
19.               ed.name=aname
20.               ed.next=NULL
21.               Call Procedure Increase(ed.name);
22.               Call Procedure Add(ed.name, A[i].nptr[t1].name);
23.               ptr=A[i].nptr[t1].e
24.               if ptr=NULL then
25.                     A[i].nptr[t1].e = ed
26.               end if
27.         end for
28.         return;
29.       end if
30.   end for
```

Algorithm 4: Insert a node

4.6.1.1 Procedure Increase

```
Input: Adjacent node name as 'aname'.
Output: This procedure increases the no. of links of adjacent node by 1.

1.   foreach p←1 to 2*tnbf do
2.       foreach q←1 to A[p].nlinks do
3.           temp=A[p].nptr[q].name
4.           if aname=temp then
5.                A[p].nptr[q].nlinks++
6.                return;
7.           end if
8.       end for
9.   end for
```

Procedure 1: Increase links

4.6.1.2 Procedure Add:

Input: Newly created node name as's' as done in Algorithm 4.
Output: This procedure adds the adjacent information at the side of nodes which are adjacent to new node.

```
1.  foreach p←1 to 2*tnbf do
2.      foreach q←1 to A[p].nlinks do
3.          temp=A[p].nptr[q].name
4.          if aname=temp then
5.              edge1=Getadjacentnode(1); //edge1 as temporary Edge pointer
6.              edge1.name=s
7.              edge1.next=NULL
8.              edge=A[p].nptr[q].e   //edge as temporary Edge pointer
9.              if edge=NULL then
10.                 A[p].nptr[q].e=edge1
11.                 return;
12.             end
13.             while edge ≠ NULL do
14.                 pedge=edge  // pedge as temporary Edge pointer
15.                 edge=Next[edge]
16.             end
17.             Next[pedge]=edge1
18.         end
19.     end
20. end
```

Procedure 2: Add adjacent information

Analysis:

Best Case: If no. of clusters in the graph are less such as (1 or 2) then step 1 of Algorithm 1 will run for 1 time so the overall running time would be less than n^2 i.e. **O (n^2)**, where n is the total no. of nodes in a cluster where new node is going to be inserted.

Worst Case: If no. of clusters in the graph are equal to total no. of nodes present in a cluster in which new node is going to be inserted i.e. in step 1 of Algorithm 1 if tnbf=n then overall complexity would be less than n^3 i.e. **O (n^3)**

4.6.2 Algorithm for Traversal of ClusteredDeap

Output: This Algorithm will traverse the whole ClusteredDeap stored.

```
1.      foreach i←1 to 2*tnbf do
2.          foreach j←1 to A[i].nlinks do
```

```
3.                print "A[i].nptr[j].name"
4.                Set ptr←A[i].nptr[j].e   // ptr is temporary Edge pointer
5.                while ptr ≠ NULL do
6.                        Print "name[ptr]"
7.                  ptr = next[ptr]
8.              end
9.          end
10.     end
```

Algorithm 5: ClusteredDeap Traversal

Analysis: Let 'k' be the no. of clusters in the graph, 'n' is the no. of nodes in each cluster (approx.) and each node is connected to 'l' no. of links as an adjacent node. So overall running time of the Algorithm 2 would be **O (knl)**

Best Case: If no. of links 'l' is minimal or equal to no. of clusters i.e. l=k then running time is **$O(nk^2)$**.

Worst Case: If no. of links 'l' is equal to no. of nodes in a cluster i.e. l=n i.e. for a **completely connected graph** then running time is **$O(kn^2)$**.

4.6.3 Algorithm for finding Path from one node to another

Input: Source node as 'from'; Destination node as 'to';
Output: This Algorithm will print the optimal path from one node to another node

```
1.      foreach i←1 to top+1 do // top is stack pointer having initial value -1
2.        Set mark[i]=0
3.      end
4.      Set top=0
5.      foreach p←1 to tnbf do
6.        foreach q←1 to A[p].nlinks do
7.          temp=A[p].nptr[q].name
8.            if from = temp then
9.              top++
10.             stack[top]=temp
11.           L2:  result=Call Procedure check_neighbours(stack[top],to);
12.             if result = 1 then
13.               mark[top]=1
14.               Call Procedure show_path(to);
15.             end
16.             else if result = 2 then
17.               go to Label L2;
18.             end
19.           return;
```

```
20.              end
21.          end
22.      end
```

Algorithm 6: Find Path between any two nodes in a *ClusteredDeap*

4.6.3.1 Procedure Check_neighbours:

Input: 'dest' as the node name pointed by top of stack**;** 'to' as the node name to be searched
Output: This procedure will firstly search the entire nodes that are neighbours of the node which is on the stack pointed by 'top' pointer

```
1.       foreach p←1 to tnbf do
2.          foreach q←1 to A[p].nlinks do
3.             temp=A[p].nptr[q].name
4.                if dest = temp then
5.                   ptr=A[p].nptr[q].e
6.                   while ptr ≠ NULL do
7.                      if ptr→name = to then
8.                         return 1;
9.                      end
10.                    ptr = ptr→name
11.                  end
12.                  if ptr = NULL then
13.                     ptr = A[p].nptr[q].e
14.                     mark[top]=1
15.                     while ptr ≠ NULL do
16.                        foreach i←1 to top+1 do
17.                           if stack[i] = ptr→name then
18.                              go to Label L1;
19.                           end
20.                        end
21.                        top++
22.                        stack[top] = ptr→name
23.                        L1: ptr = ptr→name
24.                     end
25.                  return 2;
26.                  end
27.               end
28.            end
29.      end
```

Procedure 3: Search source node's neighbours for destination

4.6.3.2 Procedure show_path:

Input: 'to' as the destination node
Output: This procedure will print the path between source node as 'from' to destination node as 'to'

```
1.    foreach i←1 to top+1 do
2.       if mark[i] = 1 then
3.          print "stack[i]"
4.       end
5.    end
6.    print "to"
```

Procedure 4: Path from source to destination

Analysis: For the computation of Algorithm 3and its corresponding procedure the overall complexity or running time would be **O (k^2*n^2*l);** where 'k' is the no. of clusters; 'n' is the no. of nodes in a cluster and 'l' is the no. of links connected to adjacent nodes.

Best Case: If k=l then running time would be, **O (k^3*n^2)**

Worst Case: For a completely connected graph i.e. l=n the running time would be, **O (k^2*n^3)**

Chapter 5

Implementation and Results

5.1 Implementation of Treaps

First step in the entire implementation is to store a random graph into Treap data structure and for this Figure 4.1 and its corresponding Table 4.1 have been considered. When Algorithm 1 is complied on the Table 4.1, the step by step process of insertion is discussed below.

Step 1:

From Table 4.1, the first key value would be “A” and its priority is “4” so initially there is no node in the Treap so new node inserted in itself act as a root of the Treap as shown in Figure 5.1:

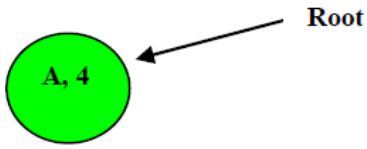

Figure 5.1: First node of Treap

Step 2:

When another node from Table 4.1, is inserted then the Treap would be:

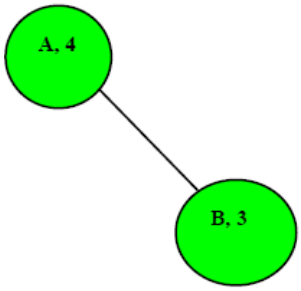

Figure 5.2: Insertion after new node

Nodes are inserted according to BST based on their key value. Here A is less and B is more so more value node comes at right child. Now according to Treap rule, rotation would takes place because priority of B is less than A as shown in Figure 5.3 and min heap is being considered.

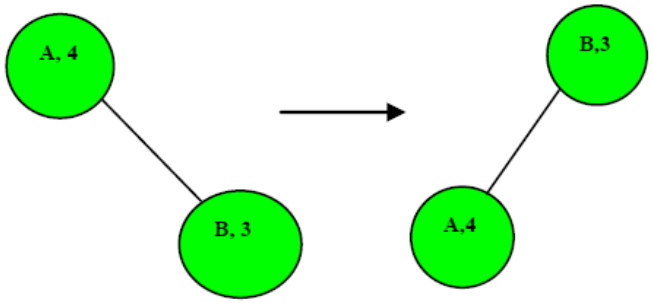

Figure 5.3: Treap after Rotation

Now B becomes the root of the Treap and A comes at left to B because key value is less so according of BST rule, it becomes left child.

Step 3:

After inserting another node from Table 4.1, structure of Treap would be shown as in Figure 5.4.

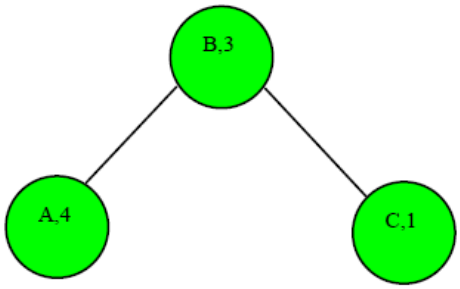

Figure 5.4: Treap after insertion of new node

Again the same rotation would take place because priority of C is less than B so after rotation treap formed would be shown as in Figure 5.5.

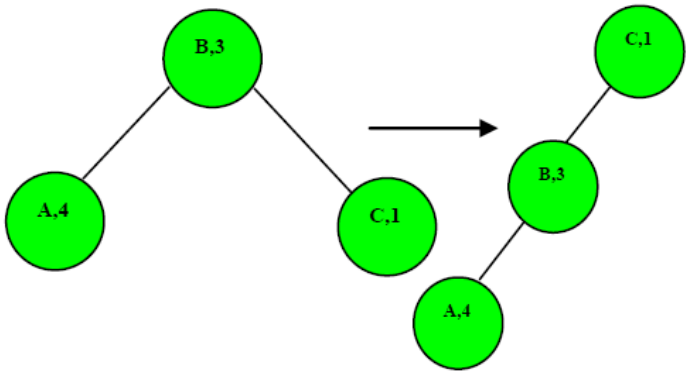

Figure 5.5: Treap after Rotation

Step n:

Final structure of the Treap after insertion of series of nodes from Table 4.1, one by one would be shown in Figure 5.6:

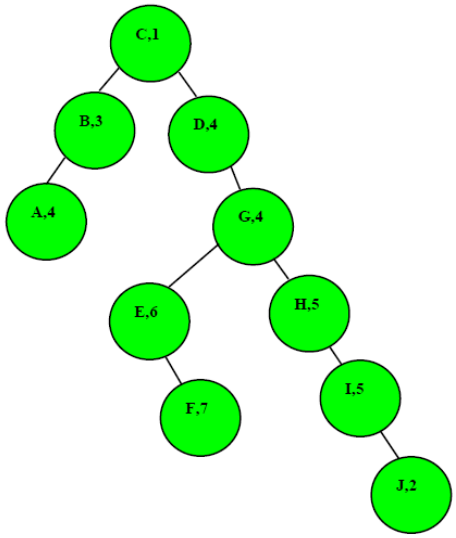

Figure 5.6: Treap after insertion of last node

Since the newly inserted node makes the Treap unbalanced as the priority of J is less than to its parent node as well as to its ancestor nodes, so after performing a series of transformations the final result which will be achieved is shown in Figure 5.7.

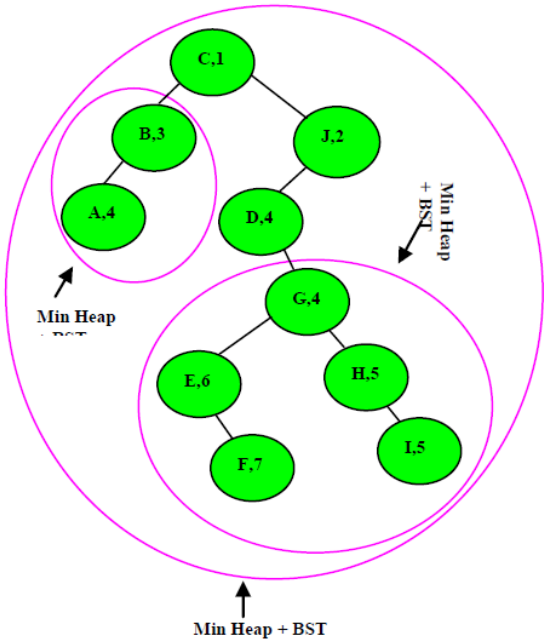

Figure 5.7: Final Output

As seen in Figure 5.7, every sub-tree and sub-sub-tree in itself satisfy the property of Treaps. Thus the graph which would be actually stored in adjacency matrix can now be stored in Treaps where it is easy to modify and scalable to large networks which adjacency matrix fails for storing graphs of dynamic nature.

5.2 Implementation of Graph Clustering

This section provides the implementation details of the technique proposed in Section 4.3.

5.2.1 Set of nodes generated

After Algorithm 2 is complied on the graph as in Figure 4.4, set for each node created is depicted in Figure 5.8, where columns depict the node name and corresponding rows depict the set that a particular node maintains.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
	A	A		A	A	A					A	A								
B		B	B	B	B	B					B	B								
C	C		C	C	C	C	C				C	C								
	D	D			D	D	D	D												
E	E	E			E	E			E	E	E	E	E	E			E			
F	F	F	F	F		F	F		F	F	F	F								
G	G	G	G	G	G		G	G			G	G								
		H	H		H	H		H											H	
			I			I	I											I	I	I
				J	J					J	J		J	J			J			
				K	K				K		K	K	K	K		K	K	K		
L	L	L		L	L	L			L	L		L	L				L			
M	M	M		M	M	M				M	M									
				N					N	N	N			N			N			
				O					O	O			O							
																P	P	P		
										Q					Q		Q	Q	Q	
				R					R	R	R		R		R	R		R	R	
								S		S					S	S	S		S	S
							T	T								T	T	T		T
								U										U	U	

Figure 5.8: Set of each node

5.2.2 Calculating NearByFactor(NBF)

After Algorithm 3 is run on the graph in Figure 4.4, NBF value generated for each node is shown in Figure 5.9, where Columns depict the node name and corresponding rows depict the set that a particular node maintains as same as in Figure 5.8. Last row of each column depict the NBF value of the node respectively.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
	A	A		A	A	A					A	A								
B		B	B	B	B	B					B	B								
C	C		C	C	C	C	C				C	C								
	D	D			D	D	D	D												
E	E	E			E	E			E	E	E	E	E	E			E			
F	F	F	F	F		F	F		F	F	F	F								
G	G	G	G	G	G		G	G			G	G								
		H	H		H	H		H											H	
			I			I	I											I	I	I
				J	J					J	J		J	J			J			
				K	K				K		K	K	K	K		K	K	K		
L	L	L		L	L	L			L	L		L	L				L			
M	M	M		M	M	M				M	M									
				N					N	N	N			N			N			
				O					O	O			O							
																P	P	P		
										Q					Q		Q	Q	Q	
				R					R	R	R		R		R	R		R	R	
								S		S					S	S	S		S	S
							T	T								T	T	T		T
								U										U	U	
NBF																				
1	1	1	1	1	1	1	1	1	2	1	1	1	2	2	3	2	2	2	3	3

Figure 5.9: NBF for each Node

5.2.3 Calculating Multiplication Factor(MF)

MF for each and every node is calculated and the final result output is depicted in Table 5.1.

For idle case taking into consideration that each factor has moderate influence on each node then for each and every node NEF would be (2+2+2+2+2+2=12).

Table 5.1: MF value for each node

Node	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**
Links	2	3	3	3	3	7	3	3	2	4	5	3	2	3	2	1	3	3	3	3	1
MF	.1	.15	.15	.15	.15	.35	.15	.15	.1	.2	.25	.15	.1	.15	.1	.05	.15	.15	.15	.15	.05

5.2.4 Calculating Total Value(TV)

Now to calculate TV for a node is given as:

For node A:

TV_A = (Sum all the MF values present in a set of $Node_A$) * NEF_A

= (3/20 + 3/20 + 3/20 + 7/20 + 3/20 + 3/20 + 2/20) * 12

= (24/20) * 12

= 1.2 * 12

= 14.4

Similarly TV for all other nodes would be given in Table 5.2.

Table 5.2: TV for each node

Node	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
Sum of MF	1.2	1.3	1.4	1.0	2	1.7	1.5	1.0	.8	1.3	1.6	1.9	1.4	1	.7	.4	.7	1.4	.9	.7	.4
TV	14.4	15.6	17.4	12.6	24	20.4	18.6	12.6	9.6	15.6	19.8	22.8	17.4	12	9	5.4	9	16.8	10.8	9	4.8

5.2.5 Calculating threshold value (Θ)

To cluster nodes threshold value theta (Θ) is determined to make groups of related information. As in Figure 5.9, there are total of 3 NBF so there is a requirement of 3 threshold values for each NBF say (Θ_1, Θ_2, Θ_3). So Θ_1 is only used to make cluster for nodes of NBF=1 and similarly Θ_2 is only used to make cluster for nodes of NBF=2 and so on for all other NBF. So Θ_{NBF} is calculated and the clustering is done for the nodes.

For example Θ_1 is the sum of any 3 node's TV i.e. TV of $node_A$, $node_C$ and $node_D$ in Figure 4.4 so,

$\Theta_1 = (14.4+17.4+12.6)/3$

$= 14.8$

Similarly Θ_2, Θ_3 can be calculated in Table 5.3.

Table 5.3: Calculated Θ values

Θ_1	Θ_2	Θ_3
14.8	13.8	7.2

5.2.6 Final clusters achieved after applying the proposed technique

So on the basis of the values of Θ shown in Table 5.3; Figure 4.4 can be clustered as shown in Figure 5.10.

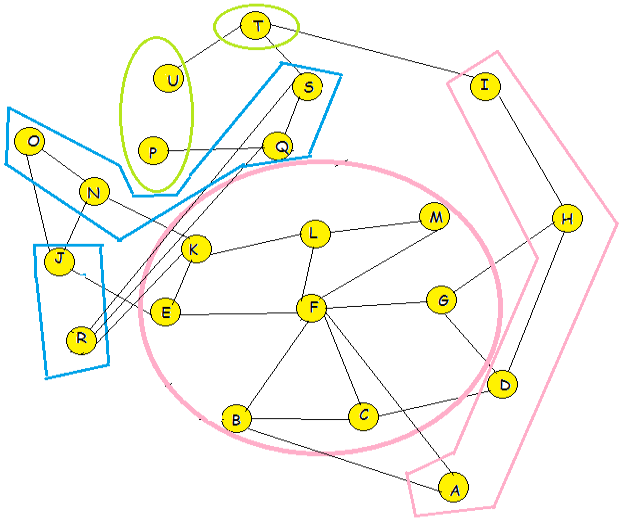

Figure 5.10: Clustered Social Network

5.3 Implementation of ClusteredDeap

Consider a clustered graph as depicted in Figure 4.7, as there are 6 clusters having 3 relating group so when Algorithm 4 is compiled on the respective clustered graph then a dynamic array of 6 indexes are created which would map the clusters as cluster head or cluster root such as:

Step 1:

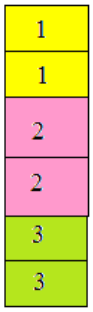

Figure 5.11: Dynamic Array

Now from the Figure 5.11, it is clear that Cluster 1 as (A[i].nbf=1) and Cluster part is 1 as (ipart=1), there are 7 nodes as (A[i].nlinks=7). So corresponding to index 1 of existing array, another dynamic structured array of linked list is created for storing 7 nodes such as shown in Figure 5.12;

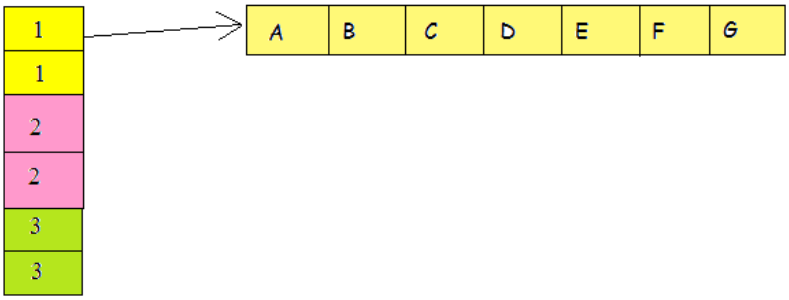

Figure 5.12: Dynamic Array with nodes present

In Figure 5.12 each index of structured linked list (node) has an internal representation that is same as that is shown in Figure 4.6. So from the Algorithm 4, step 7 will create an array of structured list of 7 nodes and 'n1' is a pointer (*n1) to that array and this corresponding pointer is mapped to array index A[1] as shown in step 15. Now the corresponding entry of each node is inserted using A[i].nptr[j].name, A[i].nptr[j].nlinks, A[i].nptr[j].e as shown in Figure 4.6. Now the information about adjacent node(s) is inserted using the steps 17 to 29 of Algorithm 4 as shown in Figure 5.13;

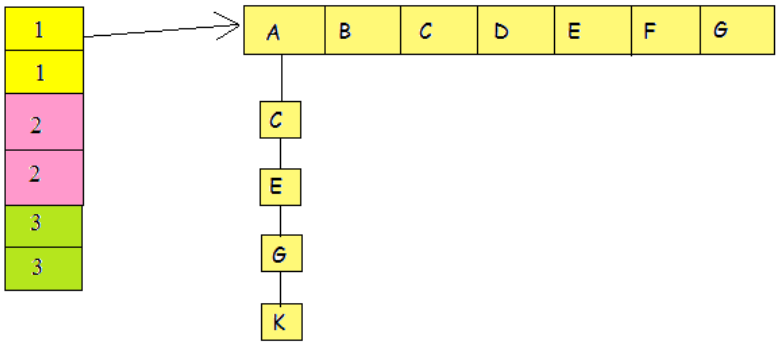

Figure 5.13: Nodes with Neighbors

From step 21 of Algorithm 4, a procedure is called to increase the no. of links by 1 to each of node that is added as an adjacent node. For example, in Figure 5.13, (C, E, G, K) are added as node A's adjacent so procedure increase will automatically increase the no. of links by 1 to the nodes (C, E, G, K) and similarly from step 22 of Algorithm 4, a procedure Add is called up which would add the node (such as 'A') as an adjacent node to (C, E, G, K) as shown in Figure 5.14.

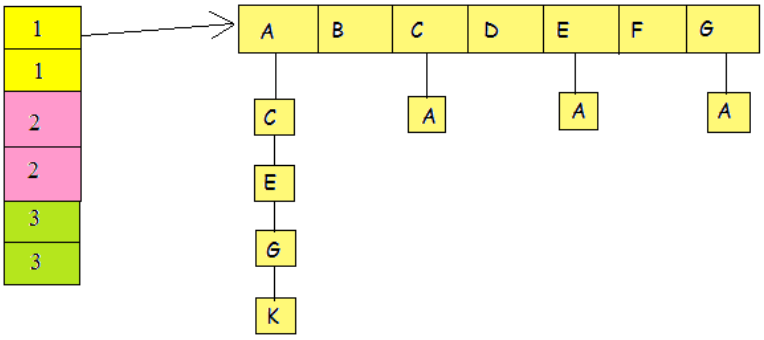

Figure 5.14: Result after Procedure Add

Step 2:

As seen in Figure 4.7, when the clustered graph would be properly mapped to a ClusteredDeap then using Algorithm 5, it would process the nodes present in the ClusteredDeap as given in Figure 5.15

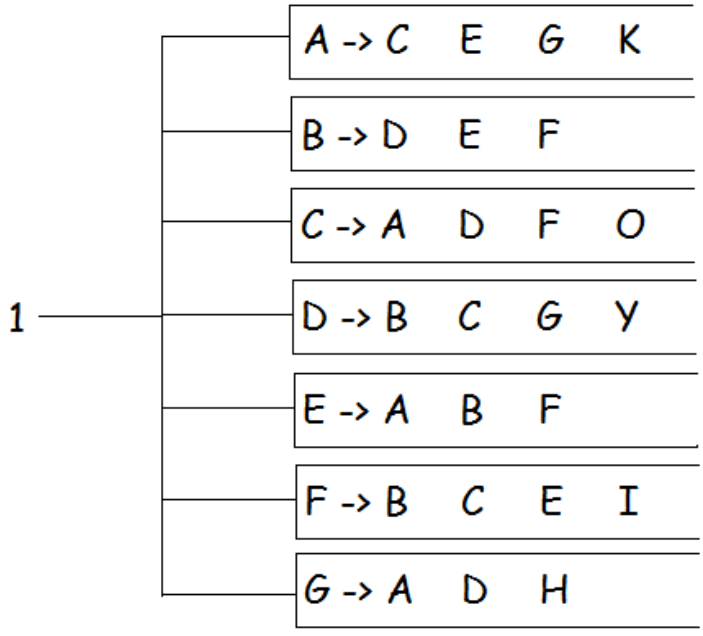

Figure 5.15: Graph mapped to ClusteredDeap

Figure 5.15 depicts the ClusteredDeap traversal for just one cluster. But when whole Algorithm would be complied then it would traverse all the clusters present in a graph as shown in Figure 4.7.

Step 3:

Consider the graph as shown in Figure 4.7; and if there is a requirement to find the path from node ‘A’ to node ‘M’ then it would be processed as shown in Figure 5.16;

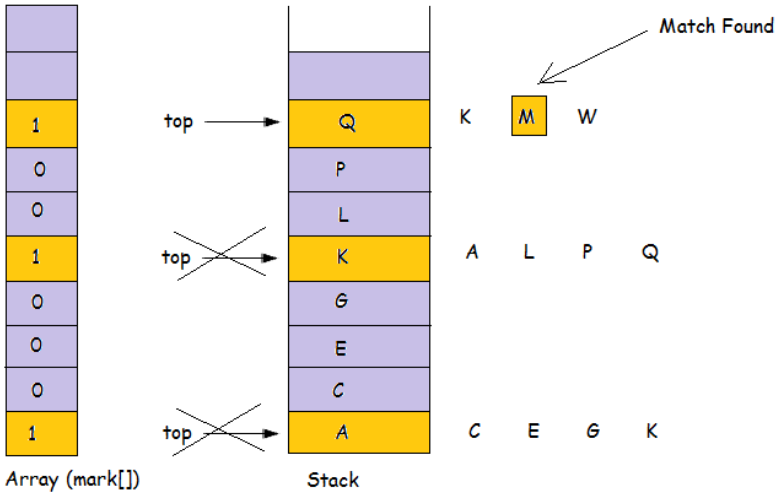

Figure 5.16: Path Processing in a Stack

So in Figure 5.16; there is one stack which would keep track of nodes that are being searched for finding the destination node and there is one array named ‘mark []’ as used in step 2 of Algorithm 6 whose all entries are kept 0 initially, which would keep track of only those nodes which are part of intermediate nodes for finding path between source to destination. As in step 10, source node is put on to stack and its status is marked as 1 means this node is already processed so that in future if this node comes again as an adjacent node of any node then it wouldn’t be added to stack. Now the adjacent nodes of marked node are being searched for finding the destination. If the destination node is not found then all the adjacent nodes are added to stack as done in step 21-22 of procedure 3 (Algorithm 6) and then the same process is repeated for finding the destination node. Thus when the destination node is found then from the step 12-14 of Algorithm 6, the path would be printed as shown in Figure 5.17.

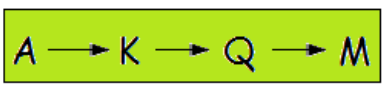

Figure 5.17: Actual Path

In Figure 5.17; only those nodes gets printed whose value in array mark is 1 as done in Figure 5.16 using step 2-6 of procedure 4 (Algorithm 6);

5.3.1 Final Output of ClusteredDeap

As there are 6 clusters and 3 relating groups so total of 3 *ClusteredDeap* will be formed such as:

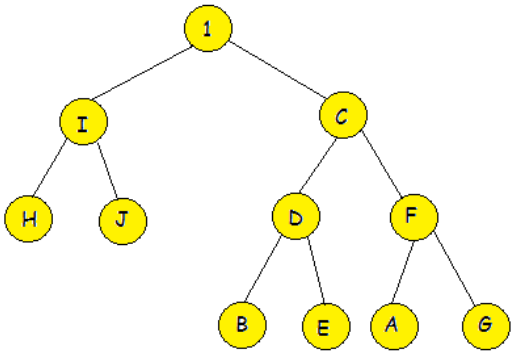

Figure 5.18: ClusteredDeap with relating group 1

Figure 5.18 depicts a ClusteredDeap with relating group 1 consists of 2 clusters together. Left subtree is min-Heap and right subtree is a max-Heap. Nodes are inserted into min-Heap or max-Heap on the basis of their no. of links parameter and root node of ClusteredDeap is named as 1 so that it would become easy to distinguish with other ClusteredDeap having different relating groups. Similarly other ClusteredDeap are depicted in Figure 5.19 and Figure 5.20.

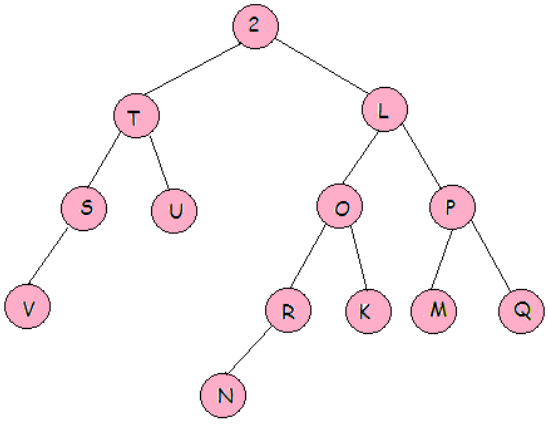

Figure 5.19: ClusteredDeap with relating group 2

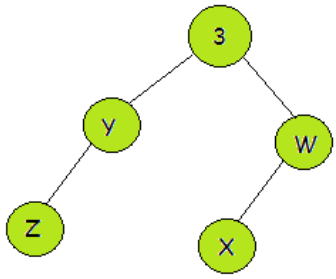

Figure 5.20: ClusteredDeap with relating group 3

5.3.2 Memory Layout of ClusteredDeap

As depicted in Figure 5.18, Figure 5.19 and Figure 5.20 there are a total of 3 ClusteredDeap formed. But internally all these 3 ClusteredDeap are a part of one package which is stored in a heterogeneous data structure. The internal memory layout of the package containing all these ClusteredDeap is depicted in Figure 5.21.

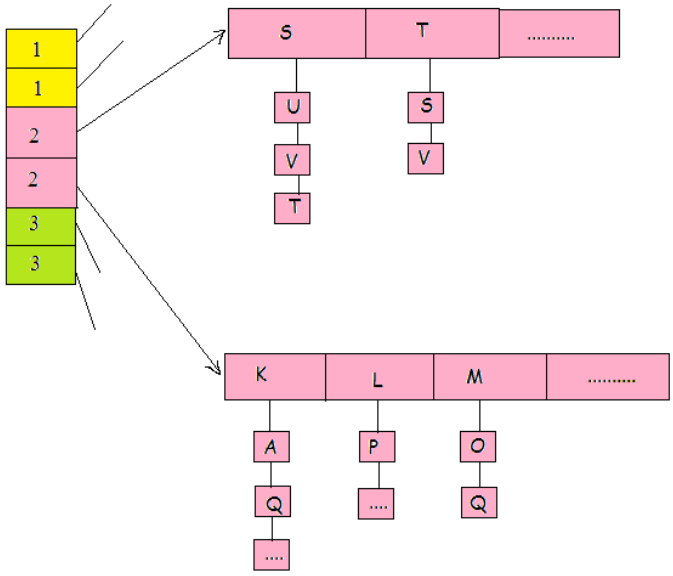

Figure 5.21: Full Memory Layout having ClusteredDeap of relating group 2

Chapter 6

Conclusion and Future Scope

6.1 Conclusion

In general, the techniques used to analyze the social network achieve good results theoretically and by using ClusteredDeap data structure the space used for storing and mapping clustered graph works efficiently as a compatible data structure. The technique discussed in this thesis is easy to implement and scalable to a large network. By using Treaps the space used for storing a graph is tremendously decreased because of the use of linked list and pointers. The heuristic and approximation approach used here gives promising results. Above all, these techniques can also work for other social networks when there are few parameters such as transport network, railway network, chemical compounds, etc.

The current work done has a strong impact on the storage behavior of social network or graph. The approach used to store a Graph into Treap data structure gives promising results and the storage schema in itself has a strong implementation impact for optimization of space and query time.

Similarly the approach used to find out the community or clusters in a social network for analysis also have various advantages over traditional approaches. As it undertook various parameters for consideration such as Near by Factor, Node existence Factor, Multiplication Factor, Threshold values etc. Here the detailed study of analysis on social networks made the study of various other real time networks very easy.

6.2 Future Scope

The work done here can be extended to large number of parameters and for more scalability up to million and billions of nodes. The work can be enhanced by using various mathematical models which work with real time entity. Secondly the storage schema can be upgraded with any new storage technique where it would require lesser storage space as well as lesser access time leading to further optimization of graph storage and cluster storage.

References

[1] D.J. Watts. 1999; “Small Worlds: The Dynamics of Networks between Order and Randomness”; Princeton University Press, Princeton, NJ.

[2] J.B. Shearer, R. J. McEliece. 1977; “There is no Mac Williams identity for convolutional codes”; IEEE Trans. Inform. Theory, IT-23, pp. 775-776.

[3] O. Frank. 1981; “A Survey of Statistical Methods for Graph Analysis. Sociological Methodology”, pp. 110-155.

[4] Bernard Elspas and James Turner. 1970: “Graphs with circulant adjacency matrices”, pp. 297

[5] Moreno, L. Jacob. 1934; “Who Shall Survive? Foundations of Sociometry Group Psychotherapy and Sociodrama”; Nervous and Mental Disease Publishing Washington DC.

[6] M. Newman, A. L. Barabasi and D. J. Watts. 2006; “The Structure and Dynamics of Networks”;

[7] J. Leskovec, E. Horvitz. 2008; “Planetary-scale views on a large instant messaging network”; In WWW ’08: Proceeding of the 17th international conference on World Wide Web, New York, NY, USA, ACM, pp. 915–924

[8] S. Wasserman, K. Faust. 1994; “Social Network Analysis: Methods and Applications”; Cambridge University Press

[9] K. Tsuda, H. Saigo. 2010; “Graph Classification, in Managing and Mining Graph Data”; Springer

[10] L. Tang, H. Liu, J. Zhang, Z. Nazeri. 2008; “Community evolution in dynamic multi-mode networks”; In KDD ’08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining, New York, NY, USA, ACM, pp. 677–685

[11] J. Denrell. 2009; “Indirect social influence Science”, pp. 47–48.

[12] A. A. Nanavati, S. Gurumurthy, G. Das, D. Chakraborty, K. Dasgupta, S. Mukherjea, A. Joshi. 2006; “On the structural properties of massive telecom call graphs: findings and implications”; In CIKM ’06: Proceedings of the 15th ACM

international conference on Information and knowledge management, New York, NY, USA, ACM, pp. 435–444

[13] M. Akiko, R. Rudy. 2010; “Classifying positions in online debates from reply activities and opinion expressions”; In Proceedings of the 23rd International Conference on Computational Linguistics, Beijing, China, pp. 869–875

[14] Scott. 2000; “Social Network Analysis: A Handbook”; Sage

[15] Yeung. 2003; “Clustering gene-expression data with repeated measurements”; Genome Biology, vol. 4, no. 5, pp. G34.1–17

[16] K. Tsuda, H. Saigo. 2010; “Graph Classification, in Managing and Mining Graph Data”; Springer

[17] A. W. Lim, S.-W Liao, and M. S. Lam. 2001; “Blocking and array contraction across arbitrarily nested loops using affine partitioning”; In Proceedings of the eighth ACM SIGPLAN symposium on Principles and practices of parallel programming, ACM Press, pp. 103-112

[18] W. Thies, F. Vivien, J. Sheldon, and S. Amarasinghe. 2001; “A unified framework for schedule and storage optimization”; In Proceedings of the ACM SIGPLAN 2001 Conference on Programming Language Design and Implementation, ACM Press, pp. 232-242

[19] A. Cohen, V. Lefebvre. 1988; “Optimization of storage mappings for graphs” Technical Report 1998/46, PRiSM, U. of Versailles.

[20] G. Pike. 2002; “Reordering and Storage Optimizations for graphs”; PhD thesis, University of California, Berkeley.

[21] A. Cohen. 1999; “Parallelization via constrained storage mapping optimization”.

[22] A. Cohen and V. Lefebvre. 1988; “Optimization of storage mappings for parallel programs”; Technical Report 1998/46, PRiSM, U. of Versailles.

[23] H. Orsila, E. Salminen, M. H¨annik¨ainen, T.D. H¨am¨al¨ainen. 2007: “Optimal Subset Mapping And Convergence Evaluation of Mapping Algorithms for Distributing Task Graphs on Multiprocessor”; SoC, Symposium on SoC.

[24] M.S. Handcock, D.R. Hunter, C.T. Butts, S.M. Goodreau, M. Morris. 2008. Statnet; “Software Tools for the Representation, Visualization, Analysis and Simulation of

Network Data”; Journal of Statistical Software 24 (1), URL, http://www.jstatsoft.org/v24/i01.

[25] P. Feautrier. 2001; “The use of farkas lemma in memory optimization”.

[26] Eisen. 1998; “Cluster analysis and display of genome-wide expression patterns”; Proceedings of the National Academy of Sciences of the United States of America, vol. 95, no. 25, pp. 14863–14868

[27] J. Leskovec, E. Horvitz. 2008; “Planetary-scale views on a large instant messaging network”; In WWW ’08: Proceeding of the 17th international conference on World Wide Web, New York, NY, USA, ACM, pp. 915–924

[28] M. Rattigan, M.Maier, D. Jensen. 2007; “Graph Clustering with Network Structure”; ICML

[29] P. K. Chan, M. D. F. Schlag, J.Y. Zien. 1993; “Spectral k-way ratio-cut partitioning and clustering”, in: Proceedings of the 30th International Conference on Design Automation, ACM Press, New York, USA, pp. 749-754.

[30] T. Hastie, R. Tibshirani and J.H. Friedman. 2001; “The Elements of Statistical Learning”, Springer, Berlin, Germany.

[31] L. S. Lasdon. 1970; “Optimization Theory for Large Systems”; MacMillan, New York.

[32] D. B. Dantzig, and P. Wolfe. 1960; “The Decomposition Principle for Linear Programs.” Operations Research, pp. 101-111.

[33] J. F. Benders. 1962; “Partitioning Procedures for Solving Mixed Variables Programming Problems.” Numerische Mathematik, pp. 238–252.

[34] S. Kirkpatrick, C. D. Gellat Jr., and M. P. Vecchi. 1983; “Optimization by Simulated Annealing”; Science, pp. 671–680.

[35] J. J. Hopfield. 1982; “Neural Networks and Physical Systems with Emergent Collective Computational Abilities.” Proceedings of the National Academy of Sciences, pp. 2554–2558.

[36] T. Siethoff, B. Kockelman, and K. Kockelman. 2002; “Property Values and Highway Expansions: Timing, Size, Location, and Use Effects.” Transportation Research Record, No. 1812, pp. 191-200.

[37] L. Euler. “Solutio problematis and geometriam situs pertinentis”; ACM 8 (1736), pp. 128-140.

[38] B. Bollobas. 1998; “Modern Graph Theory, Springer Verlag, New York, USA.

[39] S. Wasserman, K. Faust. 1994; “Social Network Analysis”; Cambridge University Press, Cambridge, UK.

[40] J. Scott. 2000; “Social Network Analysis: A Handbook”; SAGE Publications, London, UK.

[41] Network Analysis of an Exchange Program (PhD thesis) by Jaime Antonio Rivero Ostoic.

[42] S. P. Borgatti, M. G. Everett. 1997; “Network analysis of 2-mode data”; Social Networks, 19(3): pp. 243-269.

[43] J. Peter Carrington. “Models and Methods in Social Network Analysis”; Series(book): Structural Analysis in the Social Sciences 26.

[44] Mrvar. 2004; “Generalized Blockmodeling with Pajek Vladimir Batagelji”;

[45] P. Doreian, V. Batagelj, and A. Ferligoj. 2004; “Generalized blockmodeling of two-mode network data”; Social Networks 26, pp. 29-53.

[46] Blockmodels with maximum concentration European Journal of Operational Research 148, pp. 56–64.

[47] D. D. Sleator and R. E. Tarjan. 1985; “Self-adjusting binary trees”; Journal of the Association for Computing Machinery

[48] Blelloch, Guy E, Reid-Miller, Margaret. 1998; "Fast set operations using Treaps", Proc. 10th ACM Symp. Parallel Algorithms and Architectures; SPAA, New York, NY, USA: ACM, pp. 16–26.

[49] R. W. Irving and L. Love. 2003; “The suffix binary search tree and suffix AVL tree”; J. Discrete Algorithms, 1(5-6): pp. 387-408.

[50] G. D. Forney and M. D. Trott. 2004; “The Dynamics of Group Codes: Dual Abelian Group Codes and Systems”; IEEE Trans. Inform. Theory, IT-50: pp. 2935-2965.

[51] A. Galstyan, V. Musoyan, and Cohen. 2009; “Maximizing influence propagation in networks with community structure”; Physical Review E, vol. 79, no. 5.

List of Publications

Published

[1] Dharya Arora, Shalini Batra. “Using Treaps for Optimization of Graph Storage”; International Journal of Computer Applications, Vol. 41, No. 14, pp. 41-44, 2012 (Impact Factor: 0.7).

Communicated

[1] Dharya Arora, Shalini Batra. “Optimal Graph Clusters Storage Technique”; communicated to Elsevier: Information Processing Letters.

[2] Dharya Arora, Shalini Batra. “A Novel Technique for Social Network Analysis using Graph Clustering”; communicated to The International Arab Journal of Information Technology.