K–WAY PARTITIONING OF SIGNED BIPARTITE GRAPHS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

NURETTİN BURAK ÖMEROĞLU

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

SEPTEMBER 2012

Approval of the thesis:

K–WAY PARTITIONING OF SIGNED BIPARTITE GRAPHS

submitted by **NURETTİN BURAK ÖMEROĞLU** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan ÖZGEN _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Adnan YAZICI _____
Head of Department, **Computer Engineering**

Prof. Dr. İsmail Hakkı TOROSLU _____
Supervisor, **Computer Engineering Dept., METU**

Examining Committee Members:

Prof. Dr. Faruk POLAT _____
Computer Engineering Dept., METU

Prof. Dr. İsmail Hakkı TOROSLU _____
Computer Engineering Dept., METU

Prof. Dr. Göktürk ÜÇOLUK _____
Computer Engineering Dept., METU

Dr. Onur Tolga ŞEHİTOĞLU _____
Computer Engineering Dept., METU

Dr. Güven FİDAN _____
ARGEDOR Inc.

Date: **14.09.2012**

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name : NURETTİN BURAK ÖMEROĞLU

Signature :

ABSTRACT

K–WAY PARTITIONING OF SIGNED BIPARTITE GRAPHS

ÖMEROĞLU, Nurettin Burak

M.S., Department of Computer Engineering

Supervisor: Prof. Dr. İsmail Hakkı TOROSLU

September 2012, 88 pages

Clustering is the process in which data is differentiated, classified according to some criteria. As a result of partitioning process, data is grouped into clusters for specific purpose. In a social network, clustering of people is one of the most popular problems. Therefore, we mainly concentrated on finding an efficient algorithm for this problem. In our study, data is made up of two types of entities (e.g., people, groups vs. political issues, religious beliefs) and distinct from most previous works, signed weighted bipartite graphs are used to model relations among them. For the partitioning criterion, we use the strength of the opinions between the entities. Our main intention is to partition the data into k-clusters so that entities within clusters represent strong relationship. One such example from a political domain is the opinion of people on issues. Using the signed weights on the edges, these bipartite graphs can be partitioned into two or more clusters. In political domain, a cluster represents strong relationship among a group of people and a group of issues. After partitioning, each cluster in the result set contains like-minded people and advocated issues.

Our work introduces a general mechanism for k-way partitioning of signed bipartite graphs. One of the great advantages of our thesis is that it does not require any preliminary information about the structure of the input dataset. The idea has been

illustrated on real and randomly generated data and promising results have been shown.

Keywords: Move-Based Heuristic, Graph Partitioning, Signed Weighted Bipartite Graphs, Generic Algorithm, Linear Programming

ÖZ

İŞARETLİ AĞIRLIKLI İKİ PARÇALI GRAFLARIN K GRUBA AYRILMASI

ÖMEROĞLU, Nurettin Burak
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. İsmail Hakkı TOROSLU

Eylül 2012, 88 sayfa

Kümeleme verinin belli kriterler ışığında ayrıldığı, sınıflandırıldığı işlemdir. Gruplama işleminin sonucunda, veriler belli amaçlar doğrultusunda kümelere gruplandırılırlar. Sosyal ağlarda, insanların kümelenmesi en popüler problemlerden biridir. Bunları düşünerek, genel olarak bu probleme etkileyici bir yöntem geliştirebilmeye konsantre olduk. Bu çalışmamızda, veri iki farklı tipten oluşmaktadır (örneğin insanlar, gruplar ile politik meseleler, dini inanışlar) ve daha önceki birçok çalışmadan farklı olarak, verilerin modellenmesinde işaretli ağırlıklı iki parçalı graflar kullanıldı. Bölümleme kriteri olarak, varlıklar arasındaki düşüncelerin şiddetleri kullanılmaktadır. Bizim veriyi k bölüme ayırırkenki asıl amacımız, aynı gruba düşen nesnelerin grup içerisinde güçlü benzerlikler göstermesini sağlamaktır. Politika alanından bir örnek vermek gerekirse insanların konular hakkındaki görüşleri denebilir. Bağlantıları işaretli ve ağırlıklı kenarlar ile ifade ederek, ortaya çıkan iki parçalı grafları 2 veya daha fazla bölüme ayırabiliriz. Politika alanında, bölümleme sonrası ortaya çıkan gruplar, insanlar ve konular arasında güçlü ilişkiler olduğunu gösterirler. Kümeleme sonrasında, her bir sonuç kümesi benzer düşüncedeki insanları ve savundukları fikirleri içermektedirler.

Çalışmamız, k işaretli iki parçalı graflar için, genel mekanizmalar sunmaktadır. Tezimizin avantajlarından bir tanesi, gelen veri kümesi ne olursa olsun ön bilgi

sahibi olmak gerekmeksizin çalışabilmesidir. Fikirlerimiz gerçek ve makina tarafından üretilmiş veriler ile denenmiş ve tatminkar sonuçlar üretilerek, sonuçlar kısmında gösterilmiştir.

Anahtar Kelimeler: Hareket-Temelli Yöntem, Graf Bölümleme, İşaretli Ağırlıklı İki Parçalı Graflar, Genel Algoritmalar, Doğrusal Programlama

To my beloved wife

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude and appreciation to Prof. Dr. İsmail Hakkı TOROSLU, whom I feel very happy to have the opportunity to work with, for his valuable advices, critics, support and guidance throughout this study.

I am deeply grateful to my wife for her love, support and patience, to my son for being with us and to my family for their support, help and encouragements. Without them, this work could never have been completed.

My special thanks go to my boss Abdülkadir VARDAR for his help and suggestions during the period of writing the thesis.

I am very grateful to my colleagues for all their patience, understanding and tolerance and to all my friends who gave me support whenever I needed.

I would like to thank to the Scientific and Technological Research Council of Turkey (TÜBİTAK) for providing the financial means throughout this study.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

ABBREVIATIONS

KL – Kernighan-**L**in Algorithm

FM – Fiduccia-**M**attheyses Algorithm

LP – Linear Programming

IP – Integer Programming

ILP – Integer Linear Programming

BIP – Binary Integer Programming

MIP – Mixed Integer Programming

NLP – Nonlinear Programming

GAMS - General Algebraic Modeling System

GA – Genetic Algorithm

SA – Simulated Annealing

MBH1 – Move-Base Heuristic v.1

MBH2– Move-Base Heuristic v.2

MBH3 – Move-Base Heuristic v.3

MBH4 – Move-Base Heuristic v.4

Opt-MBH – Optimized Move-Based Heuristic

CHAPTER 1

INTRODUCTION

In this chapter, there are three sections. In the first section, we give some background information and then define the problem. In the second section, our motivation to do this work and contributions to the subject are presented. The last section gives us the road-map for the overall, explaining briefly what each chapter does.

1.1 Problem

A set of objects can be represented by a graph G in which edges are used to connect some pairs of vertices or nodes. In graph theory, there are different kinds of graphs, but in our problem we are concentrated on bipartite graphs. In [1], bipartite graph is defined as a simple graph G in which vertices $V(G)$ can be partitioned into two sets, U and V with the following properties:

1. If $u \in U$ then it may only be adjacent to vertices in V.
2. If $v \in V$ then it may only be adjacent to vertices in U.
3. $U \cap V = \emptyset$
4. $U \cup V = V(G)$

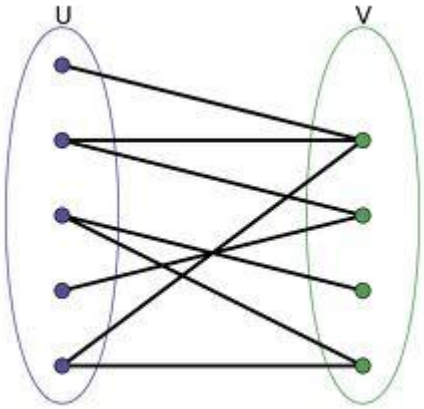


Figure 1: Example of a bipartite graph (Wikipedia)

It is clearly seen on Figure 1, bipartite graphs can be imagined as two lines of vertices parallel to each other, and edges can only be between those lines (i.e., two nodes cannot be connected if they are on the same line).

Social networks became one of the hottest topics of computer science in recent years. One very common form of a social network is actually a simple bipartite graph where one partition U represents actors (e.g., people, organizations) and the other partition V represents a set of issues (e.g., political issues, beliefs). One of the earliest definitions of this problem is given in [2]. An edge between a person and an issue represents the opinion of that person on that issue. This opinion expressed with a sign, as positive or negative, (no edge between a person-issue pair expresses “no opinion”), and, a numerical value representing the strength of the opinion of person.

Clustering, which is grouping of similar things, on such bipartite graphs is a non-trivial and interesting problem. This thesis extends previously introduced idea of [3] to be able to partition bipartite graphs into k clusters (k-way partitioning) based on the opinions expressed on the edges. Notice that the clustering should produce sub-bipartite graphs such that people in a sub-bipartite graph should have strong positive opinions on the issues of that sub-bipartite graph, and they should have strong negative opinion towards the issues in other sub-bipartite graphs.

Now, we will introduce the formal model for the problem. The inputs of *k-way partitioning of signed bipartite graph* problems are bipartite graph $G = (U \cup V, E)$, label function $\sigma : E \rightarrow R$ and partition count K. Label of an edge can be positive or negative real value. In most cases the range of the mapping is either a small subset of integers of real values. For this modeling, we assume that a positive edge from u to v where $u \in U$ and $v \in V$ means that u supports v, and a negative edge implies that u is against v. The goal of the partitioning problem is to divide the sets U and V into $(U_1, U_2, \ldots, U_k)$ and $(V_1, V_2, \ldots, V_k)$ simultaneously to form disjoint max K clusters $(U_1 \cup V_1, U_2 \cup V_2, \ldots, U_k \cup V_k)$, such that,

1. The sum of the weights of the positive edges within clusters is maximized (i.e., positive edges from U_i to V_i, $1 \leq i \leq k$),
2. The sum of the weights of the positive edges between clusters is minimized (i.e., positive edges from U_i to V_j, $1 \leq i, j \leq k \; and \; i \neq j$),
3. The sum of the weights of the negative edges within clusters is minimized (i.e., negative edges from U_i to V_i, $1 \leq i \leq k$),
4. The sum of the weights of the negative edges between clusters is maximized (i.e., negative edges from U_i to V_j, $1 \leq i, j \leq k \; and \; i \neq j$).

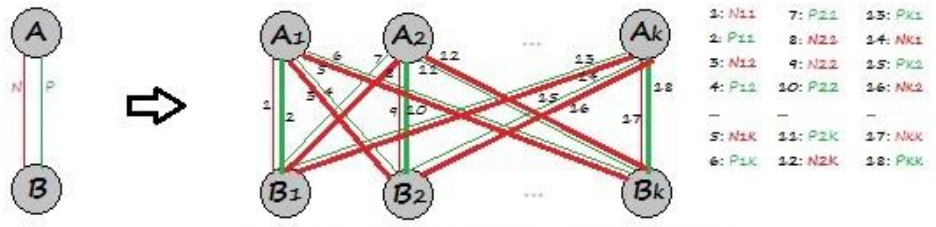


Figure 2: Partitioning of A into A_1, A_2, ... A_k and B into B_1, B_2, ... B_k

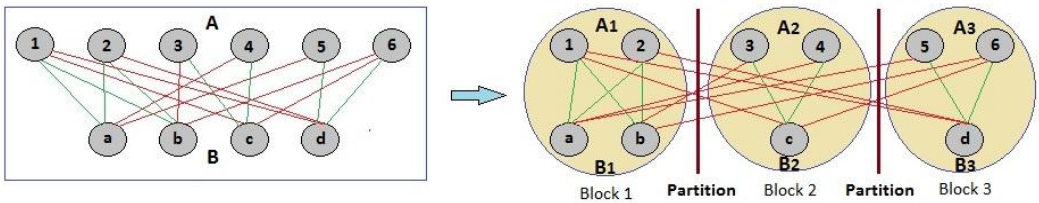


Figure 3: Nodes are distributed among blocks

In Figure 2, sets A and B are partitioned into K clusters from 1 to k, where clustering is done simultaneously. Simultaneous (i.e., vertical) partitioning is a little bit confusing; some may think that there are $2K$ clusters in the figure instead of K clusters. To clarify the key point of simultaneous partitioning, we can say that A_j and B_j jointly forms $Block_j$ like shown in Figure 3.

Each line in Figure 2 represents the sum of the weights of the edges from A_i to B_j, $1 \leq i, j \leq k$ where label N (i.e., red lines) denotes negative and label P (i.e., green lines) denotes positive signs. Note that some of the lines are thicker than the others. Thick lines represent higher values than the thin ones.

In Figure 3, six nodes from A and four nodes from B are distributed into three blocks. In this example, separators (i.e., thick vertical red lines) cut negatively weighted edges between blocks and divide nodes into 3 partitions. For illustration, all nodes are distributed ideally so that edges within blocks are all has positive and edges between blocks are all has negative sign. As one would predict, most of the time, partitioning may not be perfect, meaning that there can be negative edges within clusters and positive edges between clusters.

The clustering problem is modeled in Figure 2 and Figure 3. The goal can be realized by maximizing the sum of [(*positively weighted edges – negatively weighted edges within blocks)* – (*negatively weighted edges – positively weighted edges between blocks)*]. Meanwhile, if we maximize *the sum of positive edges within blocks*, it gives the minimal *sum of positive edges between blocks*, since the total sum is constant. The same thing appears similarly for the negatively weighted ones. Therefore, the goal statement reduces to maximize the sum of [(*positively weighted edges within blocks)* – (*negatively weighted edges between blocks)*]. More formal version of the objective function that must be maximized can be found in chapter 3.

1.2 Motivation and Contribution

In social networks, large blocks of data collected from various domains are analyzed by social analysts to dig out significant information about people and groups. We

thought that it would be great to have some efficient tools which can divide the data into meaningful subgroups. Thus, the objective of our thesis is that to create a generic tool which can partition the given dataset into subsets in reasonable time.

Since computational time for finding optimal solution to k-way partitioning of signed bipartite graph problem is unacceptably high, we present move-based (Ch.5), mathematical (Ch.3) and generic approaches (Ch.4) to solve the problem described in previous section. We provided the implementation details as well as the experimental results on various datasets for these algorithms.

To our knowledge, no efficient algorithm was presented for k-way partitioning of signed bipartite graph problem before our study, as stated in [3]. This study is the extension of the work in [3] in two folds: One of the extensions is k-way partitioning of bipartite graphs, and the other one is reduction of the execution time to the half. The second contribution is quite important, especially on big datasets.

1.3 Organization

This thesis is organized in seven main chapters, including this introduction chapter as the first chapter. In the second chapter, to let the reader understand the problem domain, survey is given. Background information and implementation details of major methods used in this work, mathematical, generic and move-based heuristic, are presented in the third, fourth and fifth chapters, respectively. The results obtained from real and randomly generated datasets are presented at the sixth chapter. Finally, the last chapter contains the conclusion and the future work.

CHAPTER 2

RELATED WORK

Due to the fact that there is vast amount of literature that highlights and studies clustering, classification (categorization), partitioning and grouping, we have addressed only to the ones that are most related to our concern. The general categorization of the unsigned partitioning algorithms and the survey can be found in [13]. As the title indicates, we have concentrated on (i) *signed* (i.e., the edge weights are both positive and negative) (ii) *bipartite* (i.e., people, organizations etc. at one side and thoughts, religious beliefs etc. at the other side) graph (iii) *multi-partitioning* (i.e., K>=2 clusters) in our survey. Nevertheless, for inspiration, we widened the circle of our interest and briefly examined some of the unsigned and arbitrary graph partitioning algorithms.

During the literature review, it was understood that graph partitioning algorithms that are relevant to our topic are categorized into different divisions based on (i) Type of graphs – Bipartite [7][9][14] / Arbitrary [4][5][12][13], (ii) Kind of clustering – Separately (one-mode) [4][5][12][13] / Simultaneously (two-mode) [7][9], (iii) Sign of edge weights – Signed [4][5][13] / Unsigned [7][9][12], (iv) Level of partitioning – Bi-partitioning (2-way) [3] / Multi-partitioning (k-way) [7][9], and (v) Distribution of nodes – Balanced [13] / Unbalanced [7][9]. In different studies, the type of our partitioning mechanism was also named as *simultaneous* [7] and *unbalanced* [13] partitioning.

In [7] and [9], two sets of entities (represented by two sets of nodes in the bipartite graph) were clustered. These works were related to document clustering, where one set of entities was set of words, and the other one was set of documents. In these works there was no information on the edges. In [4] and [5], signed arbitrary graphs

were considered, but they were not focused to bipartite graphs. In [2] and [8], similar problems were introduced; however, no effective algorithm has been introduced.

Kernighan-**L**in (KL - 1970) and **F**iduccia-**M**attheyses (FM - 1982) algorithms are two fundamental move-based heuristic algorithms used for graph partitioning from which several algorithms such as [3] have been inspired. While the first one works locally, the second one considers global connectivity. Due to the similarities with ours, we will step into details of these algorithms.

KL algorithm is an efficient heuristic method which finds effective optimal solutions for arbitrary graphs in reasonable time [11]. KL algorithm was designed for unsigned arbitrary graphs with weights on its edges. The objective of the algorithm was to divide the graph into subsets of no larger than a given maximum size in order to minimize the sum of the weights on all edges cut. The algorithm can be applied to so many fields. For instance, the problem of minimizing the number of connections in electrical circuit boards can be solved using KL algorithm. It is fast enough to solve large problems.

As stated in the paper, KL algorithm can also be adapted to paging problems in the paged memory organizations of computers. In this problem, a program can be thought as a set of connected entities which might be sub-routines, procedure blocks or instructions. References between entities constitute the connections from one entity to another. As a result, the problem can be described as a partitioning problem in which the entities must be partitioned into pages with the given page size to minimize the references across pages.

In the introduction section (2.1) of the paper [11], the simplest partitioning problem is defined and the 2-way uniform partitioning solution is applied to the problem as the following: Let $G(V, E)$ be an arbitrary graph, V be the set of vertices and E the set of edges. The KL algorithm tries to find a partition of V (2n) into two disjoint sets A (n) and B (n) of equal size such that the sum of the edges between vertices in A and B is minimized. I_a and E_a are defined in the paper as follows: I_a is the *internal cost of* $a \in A$, means the sum of the weights of the edges from vertex a to the other vertices in A. Similarly, E_a stands for *external cost of* a, that is, the sum of the

weights of the edges between vertex a and vertices in B. Let $D_a = E_a - I_a$ be the difference between external and internal values of a. If a and b are interchanged, then the gain (that is the reduction in cost) is given by $gain = old\ cost - new\ cost = T - T' = D_a - D_b - 2c_{ab}$ where c_{ab} is the cost of the possible edge between vertices a and b. The 2-way partitioning version of the KL algorithm attempts to find an optimal series of interchange operations between elements of A and B which maximizes gain and then executes the operations, producing a partition of a graph A and B. Till the cut sizes keeps decreasing, vertex pairs which give the largest decrease or the smallest increase in cut size are exchanged. These vertices are then locked and thus prohibited from participating in any further interchanges. This process continues until all the vertices are locked. When no improvement can be found in the algorithm, the partitions A and B are local minimum with respect to the algorithm. In the paper, it is claimed that "the resulting partitions has a fairly high probability of being a globally minimum partition." As the process does not guarantee the globally minimum, the process can be repeated to find as many local minimum as desired. This solution is the basic one and has some restrictions, but it can be extended easily for more general problems.

The first extension (section 2.6 of the paper) of the given KL algorithm is for unequal sized (unbalanced) subsets (assume $n_1 < n_2$). Let's assume we have partitions $|A| = n_1$ and $|B| = n_2$. Before applying Kernighan-Lin algorithm we can add dummy $n_2 - n_1$ vertices to set A such that dummy vertices have no connections to the original graph. At the end, we can remove all dummy vertices.

In the section 2.7, the second extension is described as vertices of unequal sizes. From the beginning of the algorithm, we assumed that the vertices are sized equally. Now, let's assume that the smallest vertex has unit size. Before applying KL, each vertex of sized s are replaced with s vertices which are fully connected with edges of infinite weights. In the paper authors advised that since it increases the size of the problem proportionally to vertex size s, it might be necessary to stop the generation of new vertices within acceptable bounds through tolerating the errors.

The last extension to the algorithm can be for k-way partitioning. To do that, instead of 2 sets, we partition the graph into k equal-sized sets. For each pair of subsets we

apply Kernighan-Lin algorithm. The time complexity can be reduced by recursive bi-partition.

The total running time of the algorithm is $O(Rn^3)$ for $|V| = 2n$, where "repeat loop" terminates after R passes. Since the time complexity of the KL algorithm is high, better implementations of KL algorithm can be found in the literature. Since the idea is the same, the variations of the algorithm will not be mentioned.

The other algorithm which we will go into the details is FM algorithm. The FM algorithm is more similar to ours, since just one vertex is moved to another block at each step.

FM presented in paper [10] is a KL-inspired mincut heuristic algorithm which iteratively partitions networks. Its worst case execution time is linearly dependent on the size of the netlist. In FM, new features to KL are implemented algorithm. These features can be listed as follows:

- It deals with various sizes of vertices.
- Single vertex at a time is moved across the blocks.
- Cut-size is extended to hyper-graphs. Hyper-graph is a graph in which an edge can connect any number of vertices.
- Unbalanced partitions and balanced factor are introduced.
- Efficient data structures are used to avoid unnecessary searching and improve running time.

In the paper, the mincut partitioning problem is defined for a network that contains a set of cells (modules) connected by a set of nets (signals). The aim of the algorithm is that to find a clustering of the set of modules into two blocks A and B such that the number of nets which have cells in both blocks is minimal.

Like KL, FM performs passes in which each cells move exactly once, returns best solution obtained during the pass, terminates when the pass fails to improve the final result. However, FM gives results much faster, $O|E|$ for undirected graphs and $O|p|$ implementation on hyper-graphs, where p is the number of pins in the netlist

defined in the paper. The key point is that to speed up the heuristic gain bucket structure presented in the Figure 4 is used, which allows constant time selection of the module with the highest gain and fast gain updates after each move.

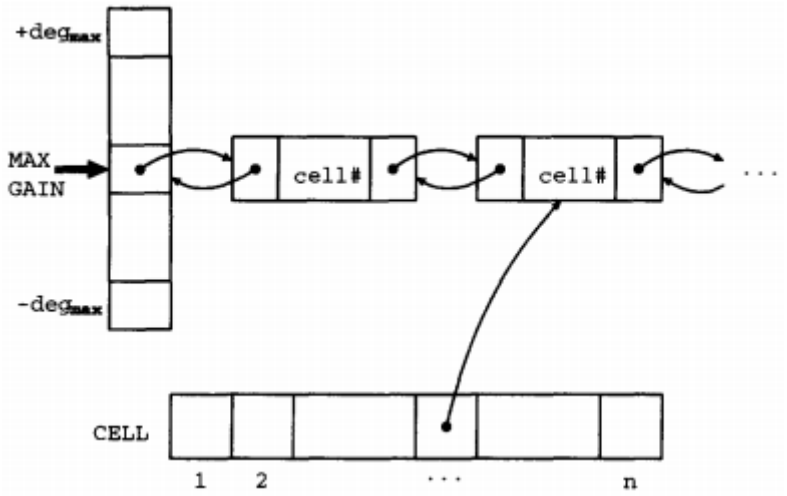


Figure 4: The Gain Bucket List Selection as shown in FM paper

Of course, the most related work is [3]. In [3], en efficient heuristic based solution has been introduced for 2-way partitioning of bipartite signed graphs. Our work extends it to k-way partitioning with time optimizations which is detailed in the next chapters.

CHAPTER 3

MATHEMATICAL METHODS

In this chapter, firstly, background information about linear and nonlinear programming is given. Secondly, linear and nonlinear programming formulations of our problem are shown. Thirdly, the simplex method used to solve linear programming is described. Finally, nonlinear programming solutions to our problem are explained.

3.1 Introduction

In this section, some of the programming methods which are linear programming (LP) and nonlinear programming (NLP) are described and the brief information about integer linear programming (ILP or IP), binary integer programming (BIP) and mixed integer programming (MIP) are given.

3.1.1 Linear Programming (LP)

Linear programming is the process of taking various linear inequalities relating to some situation, and finding the best value obtainable under those conditions. Looking back from the early 1980s, in [15], linear programming is defined by Dantzig as a “revolutionary development giving man the ability to state general objectives to find optimal policy decisions for a broad class of practical problems of great complexity.”

Dantzig’s discovery of linear programming was presented at the meeting of the Econometric Society at the University of Wisconsin in Madison in 1948. His abstract

from this meeting appeared in Econometrica [16] and some of his early linear programming papers appeared in [17][18].

In "real life", linear programming is part of a very important area of mathematics called "optimization techniques". This field of study (or at least the applied results of it) is used every day in the organization and allocation of resources. These "real life" systems can have dozens or hundreds of variables, or more.

Linear programming can be applied to various fields of study. It is used in business and economics, but can also be utilized for some engineering problems. Industries that use linear programming models include transportation, energy, telecommunications, and manufacturing. It has proved useful in modeling diverse types of problems in planning, routing, scheduling, assignment, and design.

The general process for solving linear-programming exercises is to graph the inequalities (called the "constraints") to form a walled-off area on the x, y-plane (called the "feasibility region"). Then figure out the coordinates of the corners of this feasibility region (that is, find the intersection points of the various pairs of lines), and test these corner points in the formula (called the "optimization equation") for which we are trying to find the highest or lowest value.

As evidenced by Dantzig's book [19], in 1947, he invented the Simplex Method, which solves linear programming by running along polytope edges of the visualization solid to find the best answer. L. Khachain (1979) found a new algorithm for linear programming [20], but it is $O(x^5)$ polynomial time algorithm and really slow. Narendra Karmarkar, in 1984, discovered a much more efficient polynomial-time algorithm for linear programming [21]. This method goes through the middle of the solid (making it a so-called interior point method), and then transforms and warps. Karmarkar's announcement led to these methods receiving a great deal of attention.

3.1.1.1 Standard Form of the Linear Programming

Every linear program can be expressed as in "standard" form where the *objective function* (3.1) is maximized, the *constraints* (3.2 - 3.m+1) are equalities and the *variables* (3.m+2) are all nonnegative. *Linear objective function* should be

represented as a linear function. An optimum solution that meets *constraints'* requirements is searched. Each constraint should be represented as a linear equation.

$$max\ z = \quad c_1x_1 + c_2x_2 + \cdots + c_nx_n \tag{3.1}$$

$$subject\ to$$

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \tag{3.2}$$

$$\vdots \qquad\qquad \vdots$$

$$a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = b_m \tag{3.m+1}$$

$$x_1 \geq 0, \ldots, x_n \geq 0 \tag{3.m+2}$$

This is done as follows:

- If the problem is min z, convert it to max –z.
- If a constraint is $a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n \leq b_i$, convert it to equality constraint by adding the nonnegative slack variable s_i so that the constraint becomes $a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n + s_i = b_i$.
- If a constraint is $a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n \geq b_i$, convert it to equality constraint by subtracting the nonnegative slack variable s_i so that the constraint becomes $a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n - s_i = b_i$.
- If the sign of the variable x_i is not known, replace the x_i with $x_i' - x_i''$ where $x_i' \geq 0$ and $x_i'' \geq 0$

3.1.1.2 Simplex Method

Basic algorithm generally used for linear programming is the simplex method [19][22]. It was proven to solve linear formulated problems of acceptable size in a reasonable time.

> The simplex method works by finding a feasible solution, and then moving from that point to any vertex of the feasible set that improves the cost function. Eventually a corner is reached from which any movement does not improve the cost function. This is the optimal solution. [22]

The problem is usually formulated in matrix form, and represented as:

$$\max \boldsymbol{c}^T \boldsymbol{x}$$

$$subject\ to\ \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \geq 0$$

where $\boldsymbol{x}$ represents the vector of variables (to be determined), $\boldsymbol{c}$ and $\boldsymbol{b}$ are vectors of (known) coefficients and $\boldsymbol{A}$ is a (known) matrix of coefficients [23]. In this formulation, a vector $\boldsymbol{x}$ is a feasible solution of the linear programming problem if it satisfies the given constraints. Problems defined in this formulation have three different types [19]:

1- **Infeasible:** None of the vectors in solution space can satisfy the given constraints.
2- **Unbounded:** Given constraints are not enough for bounding objective function parameters in the solution space. So, a better solution with an improved objective function value can exist.
3- **Optimal:** Problem formulated in linear programming has an optimum value for the objective function and there exists vector(s) that can create such optimum value with satisfying the given constraints.

3.1.1.3 Integer Linear Programming (ILP)

If the variables in linear programming may take only integer values instead of real values then it is called integer linear programming (ILP), also known as integer programming (IP). It is a special case of LP, but in contrast to LP finding the optimal solution is mostly NP-hard.

3.1.1.4 Binary Integer Programming (BIP)

It is a special case of ILP, in which variables can only take 0 or 1 (i.e., binary values, not arbitrary integers). These problems are also known as NP-hard.

3.1.1.5 Mixed Integer Programming (MIP)

If only some of the unknown variables are required to be integers, then the problem is called a mixed integer programming (MIP) problem. These are generally also NP-

hard. There are however some important subclasses of IP and MIP problems that are efficiently solvable.

We do not go into the details of how to solve these problems but some advanced algorithms that can solve integer unknown problems can be listed as follows:

- cutting-plane method
- branch and bound
- branch and cut
- branch and price

3.1.2 Nonlinear Programming (NLP)

A nonlinear programming problem is an optimization problem where the objective function or some of the constraints are nonlinear.

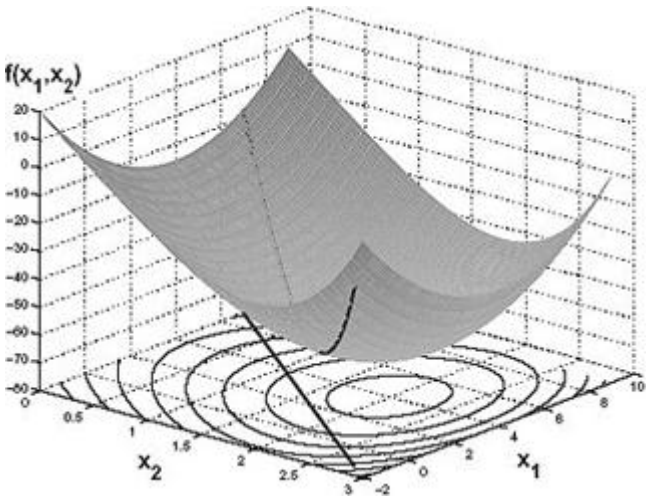


Figure 5: A convex function to be optimized. (Prof. Robert Freund)

Figure 5 is drawn in 3D environment to illustrate one of the optimization problems as a non-linear function. Since the figure has been added just to give an idea about NLP, we will not go into the details.

Nonlinear programming (NLP) involves minimizing or maximizing a nonlinear objective function subject to bound constraints, linear constraints, or nonlinear constraints, where the constraints can be inequalities or equalities. Example problems in engineering include analyzing design tradeoffs, selecting optimal designs, and incorporating optimization methods in algorithms and models.

3.1.2.1 General Form of the Nonlinear Programming

$$\min_{x} f(x)$$

$$subject\ to\ g(x) \geq 0$$

The objective function can be maximized or minimized and the constraints can be $\leq, \geq or =$.

Nonlinear programming is hard to solve. Most of the solvers can only find approximate solutions to NLPs, and local optimum values for objective functions. There are different types of algorithms used to solve NLPs, some of them are below:

1. **Interior-point:** especially useful for large-scale problems that have sparsely or structure
2. **Sequential quadratic programming (SQP):** solves general nonlinear problems and honors bounds at all iterations
3. **Active-set:** solves problems with any combination of constraints
4. **Trust-region reflective:** solves bound constrained problems or linear equalities only

3.1.2.2 Unconstrained Nonlinear Programming

Many NLPs do not have any constraints. They are called unconstrained NLPs. Unconstrained nonlinear programming is the mathematical problem of finding vector $\boldsymbol{x}$ that is a local minimum to the nonlinear scalar function $\boldsymbol{f}(\boldsymbol{x})$. Unconstrained means that there are no restrictions placed on the range of $\boldsymbol{x}$.

$$\min_{x} f(\boldsymbol{x})$$

Unconstrained NLPs can be solved by using the following algorithms:

1. **Quasi-Newton:** uses a mixed quadratic and cubic line search procedure and the Broyden-Fletcher-Goldfarb-Shanno (BFGS) formula for updating the approximation of the Hessian matrix
2. **Nelder-Mead:** uses a direct-search algorithm that uses only function values (does not require derivatives) and handles no smooth objective functions
3. **Trust-region:** used for unconstrained nonlinear problems and is especially useful for large-scale problems where sparsely or structure can be exploited

3.1.3 GAMS (General Algebraic Modeling System)

GAMS is a tool used for high-level modeling of mathematical formulations and optimization. There are various high-performance solvers available in it. Because of its own language for mathematical programming, it includes a compiler in itself to transform mathematical representation to representations required by specific solver engines. GAMS is tailored for complex, large scale modeling applications, and allows us to build large maintainable models that can be adapted quickly to new situations. Detailed information can be obtained from http://www.gams.com.

Table 1: GAMS Model Types and Descriptions

Model Type	Description
LP	Linear Programming
MIP	Mixed-Integer Programming
NLP	Non-Linear Programming
MCP	Mixed Complementarity Problems
MPEC	Mathematical Programs with Equilibrium Constraints
CNS	Constrained Nonlinear Systems
DNLP	Non-Linear Programming with Discontinuous Derivatives
MINLP	Mixed-Integer Non-Linear Programming
QCP	Quadratic Constrained Programs
MIQCP	Mixed Integer Quadratic Constrained Programs

In Figure 6, the solver/model type matrix for GAMS version 23.9 shows which solver is capable of which model type. We wish to emphasize that underlined bold solvers are the ones that are able to solve NLPs, which can be seen from the column named as “NLP”. Other columns of model type matrix are put for informational purposes only. Results of these solvers on various datasets are compared in chapter 6.

Solver/Model type availability - 23.9 July 4, 2012												
SOLVER	**LP**	**MIP**	**NLP**	**MCP**	**MPEC**	**CNS**	**DNLP**	**MINLP**	**QCP**	**MIQCP**	***Stoch.***	***Global***
ALPHAECP								✔		✔		
BARON 11.1	✔	✔	✔				✔	✔	✔	✔		✔
BDMLP	✔	✔										
COIN-OR	✔	✔	✔				✔	✔	✔	✔		✔
CONOPT 3	✔		✔			✔	✔		✔			
CPLEX 12.4	✔	✔							✔	✔		
DECIS	✔										✔	
DICOPT								✔		✔		
GLOMIQO 2.0									✔	✔		✔
GUROBI 5.0	✔	✔							✔	✔		
KNITRO 8.0	✔		✔				✔	✔	✔	✔		
LINDO 7.0	✔	✔	✔				✔	✔	✔	✔	✔	✔
LINDOGLOBAL 7.0	✔	✔	✔				✔	✔	✔	✔		✔
LGO	✔		✔				✔		✔			✔
MILES				✔								
MINOS	✔		✔				✔		✔			
MOSEK 6	✔	✔	✔				✔		✔	✔		
MPSGE												
MSNLP			✔				✔		✔			✔
NLPEC				✔	✔							
OQNLP			✔				✔	✔	✔	✔		✔
PATH				✔		✔						
SBB								✔		✔		
SCIP		✔	✔				✔	✔	✔	✔		
SNOPT	✔		✔				✔		✔			
SOPLEX	✔											
XA	✔	✔										
XPRESS 23.01	✔	✔							✔	✔		

Figure 6: The Solver / Model type Matrix (www.gams.com)

3.2 Mathematical Representation of the Problem

In this section, we set up linear and nonlinear programming formulations of the problem. Problem definition can be found in the introduction part (chapter 1). Formulations are supported with the example throughout the section.

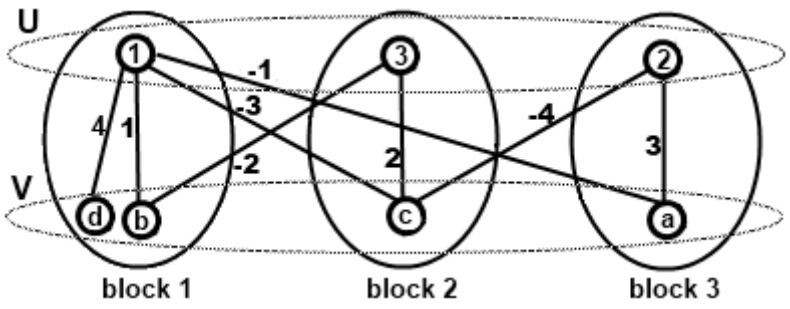


Figure 7: Illustrative Example

Let $U = U_1 \cup U_2 \cup .. U_k$ and $V = V_1 \cup V_2 \cup .. V_k$ be partitioning of the nodes of bipartite graph $G = (U \cup V, E)$.

In Figure 7, U, V and E are illustrated as $U = \{1,2,3\}$, $V = \{a, b, c, d\}$ and $E = \{\{1, d, 4\}, \{1, b, 1\}, \{1, c, -3\}, \{1, a, -1\}, \{3, b, -2\}, \{3, c, 2\}, \{2, c, -4\}, \{2, a, 3\}\}$. In the example, vertices of U and V are divided into three blocks $(K = 3)$, where $U_1 = \{1\}, U_2 = \{3\}, U_3 = \{2\}, V_1 = \{b, d\}, V_2 = \{c\}$ and $V_3 = \{a\}$. For the sake of clarity, in the figure "a", "b", "c" and "d" are used as the members of V, however, in the remaining part; we refer to these elements with numbers as $\boldsymbol{a = 1, b = 2, c = 3}$ $\boldsymbol{and}$ $\boldsymbol{d = 4}$.

Let

$$\boldsymbol{b_j} = (b_{j1},\ b_{j2}, \dots,\ b_{j|U|})^{\boldsymbol{T}}, \quad \boldsymbol{p_j} = (p_{j1},\ p_{j2}, \dots,\ p_{j|V|})^{\boldsymbol{T}}$$

be indicator vectors for U_j and V_j respectively, $1 \leq j \leq k$. ($\boldsymbol{T}$ → transpose) Thus,

$$b_{ju} = \begin{cases} 1, & if\ node\ u \in U_j \\ 0, & otherwise \end{cases}, \quad p_{jv} = \begin{cases} 1, & if\ node\ v \in V_j \\ 0, & otherwise \end{cases}$$

According to the figure, indicator vectors are as following:

$$\boldsymbol{b_1} = \begin{bmatrix}1\\0\\0\end{bmatrix} \boldsymbol{b_2} = \begin{bmatrix}0\\0\\1\end{bmatrix} \boldsymbol{b_3} = \begin{bmatrix}0\\1\\0\end{bmatrix} \; and \; \boldsymbol{p_1} = \begin{bmatrix}0\\1\\0\\1\end{bmatrix} \boldsymbol{p_2} = \begin{bmatrix}0\\0\\1\\0\end{bmatrix} \boldsymbol{p_3} = \begin{bmatrix}1\\0\\0\\0\end{bmatrix}$$

Clearly,

$$\sum_{j=1}^{k}\sum_{u=1}^{|U|} b_{ju} = |U| \text{ and } \sum_{u=1}^{|U|} b_{ju} = \left|U_j\right|,\ 1 \le j \le k$$

,

$$\sum_{j=1}^{k}\sum_{v=1}^{|V|} p_{jv} = |V| \text{ and } \sum_{v=1}^{|V|} p_{jv} = \left|V_j\right|,\ 1 \le j \le k$$

and

$$\sum_{j=1}^{k} b_{ju} = 1,\ 1 \le u \le |U|, \text{ similarly, } \sum_{j=1}^{k} p_{jv} = 1,\ 1 \le v \le |V|.$$

Let $\boldsymbol{A} = (a_{uv})$ represents the adjacency matrix for the bipartite graph $G = (U \cup V, E)$. The sum of all edges in the clusters is given by [6]

$$\sum_{j=1}^{k}\sum_{u\in U_j}\sum_{v\in V_j} a_{uv} = \sum_{j=1}^{k}\sum_{u=1}^{|U|}\sum_{v=1}^{|V|} a_{uv} b_{ju} p_{jv} = \sum_{j=1}^{k} \boldsymbol{b_j}^{\boldsymbol{T}} \boldsymbol{A}\, \boldsymbol{p_j}$$

The mathematical programming formulation can be written as follows:

$$\max L = \sum_{j=1}^{k}\sum_{u=1}^{|U|}\sum_{v=1}^{|V|} a_{uv} b_{ju} p_{jv} - \sum_{\substack{j,n=1\\ j\neq n}}^{k}\sum_{u=1}^{|U|}\sum_{v=1}^{|V|} a_{uv} b_{ju} p_{nv} \qquad (3.1)$$

Subject to

$$\sum_{j=1}^{k} b_{ju} = 1, \qquad 1 \le u \le |U| \qquad (3.2)$$

$$\sum_{j=1}^{k} p_{jv} = 1, \qquad 1 \le v \le |V| \qquad (3.3)$$

The objective function (3.1) gives the maximized L value as the objective value, by the help of constraints (3.2) and (3.3). b_{ju}'s and p_{jv}'s expressed in (3.1), (3.2) and (3.3) are the variables of these equations. It appears that the equation (3.1) is non-linear, since there are non-linear multipliers (i.e., $b_{ju}p_{jv}$) in it.

As seen in the above formulation (3.1), L value can be obtained by subtracting *the sum of edges across clusters* (let's say "**O"-out**) (i.e., right part) from *the sum of edges within clusters* (let's say "**I"-in**) (i.e., left part). Clearly, we can find **O** by subtracting **I** from *the total sum of edges* **(**let's say **T)**, since **T = I + O**. Thus, the above formulation (3.1) (**max L= I - O**) can be rewritten as follows (**max L= 2I - T**):

$$\max L = 2\sum_{j=1}^{k}\sum_{u=1}^{|U|}\sum_{v=1}^{|V|} a_{uv}b_{ju}p_{jv} - \sum_{u=1}^{|U|}\sum_{v=1}^{|V|} a_{uv} \qquad (3.4)$$

As the right part of the formulation (**T**) is constant, to maximize L we need to maximize the left part of the formulation (let's say L_0);

$$\max L_0 = \sum_{j=1}^{k}\sum_{u=1}^{|U|}\sum_{v=1}^{|V|} a_{uv}b_{ju}p_{jv} \qquad (3.5)$$

$$\max L_0 = \sum_{j=1}^{k} \boldsymbol{b_j}^{\boldsymbol{T}} \boldsymbol{A}\, \boldsymbol{p_j} \qquad (3.6)$$

$$\max L_0 = \sum_{u=1}^{|U|}\sum_{v=1}^{|V|} a_{uv}\sum_{j=1}^{k} b_{ju}p_{jv} \qquad (3.7)$$

To be more understandable, the constraint (3.6) can be shown in matrix form in the following way

$$\max L_0 = [b_{11} \quad b_{12} \quad \dots \quad b_{1|U|}] \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1|V|} \\ \vdots & & \ddots & \vdots \\ a_{|U|1} & a_{|U|2} & \cdots & a_{|U||V|} \end{bmatrix} \begin{bmatrix} p_{11} \\ p_{12} \\ \vdots \\ p_{1|V|} \end{bmatrix}$$

$$+\begin{bmatrix} b_{21} & b_{22} & \dots & b_{2|U|} \end{bmatrix}\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1|V|} \\ \vdots & & \ddots & \vdots \\ a_{|U|1} & a_{|U|2} & \cdots & a_{|U||V|} \end{bmatrix}\begin{bmatrix} p_{21} \\ p_{22} \\ \vdots \\ p_{2|V|} \end{bmatrix}$$

$$\vdots$$

$$+\begin{bmatrix} b_{k1} & b_{k2} & \dots & b_{k|U|} \end{bmatrix}\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1|V|} \\ \vdots & & \ddots & \vdots \\ a_{|U|1} & a_{|U|2} & \cdots & a_{|U||V|} \end{bmatrix}\begin{bmatrix} p_{k1} \\ p_{k2} \\ \vdots \\ p_{k|V|} \end{bmatrix}$$

This gives;

$$\max L_0 = b_{11}(a_{11}p_{11} + a_{12}p_{12} + \cdots + a_{1|V|}p_{1|V|}) + \ldots + b_{1|U|}(a_{|U|1}p_{11} + \cdots + a_{|U||V|}p_{1|V|})$$

$$+ b_{21}(a_{11}p_{21} + a_{12}p_{22} + \cdots + a_{1|V|}p_{2|V|}) + \cdots + b_{2|U|}(a_{|U|1}p_{21} + \cdots + a_{|U||V|}p_{2|V|})$$

$$\vdots$$

$$+ b_{k1}(a_{11}p_{k1} + a_{12}p_{k2} + \cdots + a_{1|V|}p_{k|V|}) + \cdots + b_{k|U|}(a_{|U|1}p_{k1} + \cdots + a_{|U||V|}p_{k|V|})$$

Clearly, (3.7) can also be written as;

$$maxL_0 = a_{11}(b_{11}p_{11} + b_{21}p_{21} + \cdots + b_{k1}p_{k1}) + \cdots + a_{1|V|}(b_{11}p_{1|V|} + \cdots + b_{k1}p_{k|V|})$$

$$+ a_{21}(b_{12}p_{11} + b_{22}p_{21} + \cdots + b_{k2}p_{k1}) + \cdots + a_{2|V|}(b_{12}p_{1|V|} + \cdots + b_{k2}p_{k|V|})$$

$$\vdots$$

$$+ a_{|U|1}(b_{1|U|}p_{11} + b_{2|U|}p_{21} + \cdots + b_{k|U|}p_{k1}) + \cdots + a_{|U||V|}(b_{1|U|}p_{1|V|} + \cdots + b_{k|U|}p_{k|V|})$$

Using values of Figure 7 for equation (3.7), we can find L_0 as

$$L_0 = (-1)(0+0+0) + (1)(1+0+0) + (-3)(0+0+0) + (4)(1+0+0)$$

$$+ (3)(0+0+1) + (0)(0+0+0) + (-4)(0+0+0) + (0)(0+0+0)$$

$$+ (0)(0+0+0) + (-2)(0+0+0) + (2)(0+1+0) + (0)(0+0+0)$$

$L_0 = 10$. Thus $L = 2L_0 - T = 2x10 - 0 = 20$

3.3 LP Formulation

As far as we know, it is not possible to convert this problem to LP as it is and solve with a LP solver, since there are nonlinear variables (i.e., $b_{ju}p_{jv}$) in (3.7). Because of this reason, in this thesis, in order to generate LP model, the mathematical formulation of the problem is simplified by replacing nonlinear multipliers with linear multipliers.

Let's say $x_{juv} = b_{ju}p_{jv}$ where $1 \le j \le k,\ 1 \le u \le |U|,\ 1 \le v \le |V|$.

To clarify the replacement of variables, the values of Figure 7 are used. As a result, $\boldsymbol{x_j}$'s, which are $|U|x|V|$ matrices, are found as;

$$\boldsymbol{x_1} = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \boldsymbol{x_2} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \boldsymbol{x_3} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

If we replace multipliers $b_{ju}p_{jv}$ with x_{juv} in the (3.7), we will obtain the following:

$$\begin{aligned} \max L_0 = {} & a_{11}(x_{111} + x_{211} + \cdots + x_{k11}) + \cdots + a_{1|V|}\left(x_{11|V|} + \cdots + x_{k1|V|}\right) \\ & + a_{21}(x_{121} + x_{221} + \cdots + x_{k21}) + \cdots + a_{2|V|}\left(x_{12|V|} + \cdots + x_{k2|V|}\right) \\ & \qquad\qquad \vdots \\ & + a_{|U|1}\left(x_{1|U|1} + x_{2|U|1} + \cdots + x_{k|U|1}\right) + \cdots + a_{|U||V|}\left(x_{1|U||V|} + \cdots + x_{k|U||V|}\right) \end{aligned} \tag{3.8}$$

It is easy to identify the values of x_{juv};

$$x_{juv} = \begin{cases} 1, & if\ \ b_{ju} = 1, p_{jv} = 1 \\ 0, & otherwise \end{cases}, \quad 1 \le j \le k$$

By using constraints (3.2) and (3.3), we obtain new constraints:

$$\sum_{j=1}^{k} x_{juv} = \begin{cases} 1, & if\ one\ of\ the\ x_{juv} = 1, \\ 0, & otherwise \end{cases}, \qquad 1 \le u \le |U|\ and\ 1 \le v \le |V| \tag{3.9}$$

(3.9) is true, because for j from 1 to k max one of the x_{juv} can be 1 (not necessarily), since only one of the b_{ju} and p_{jv} can be 1 (i.e., constraints (3.2) and (3.3) mean that each nodes of U and V can be present in one block). If they are in the same block, value of x_{juv} is 1, otherwise 0.

(3.8) can be rewritten as;

$$\max L_0 = \sum_{u=1}^{|U|} \sum_{v=1}^{|V|} a_{uv} \sum_{j=1}^{k} x_{juv} \qquad (3.10)$$

As seen in (3.9) $\sum_{j=1}^{k} x_{juv}$ can take values 0 or 1. So, to maximize L_0 in (3.10), it is clear that if $a_{uv} \leq 0$, we do not add it to L_0 by setting 0 to $\sum_{j=1}^{k} x_{juv}$. Thus, L_0 becomes *the sum of all positively weighted edges* (let's say **P**).

If we return to (3.4), in our hand, we have (**max L = 2L$_0$ – T**). Since **T** is the sum of all positively weighted edges (**P**) and all negatively weighted edges (let's say **N**), we can restate (3.4) as (**max L = 2P – (P+N) = P – N**), meaning that the sum of absolute values of all edges. (**P - N**) can be obtained only if we put all the positively weighted edges into blocks and all negatively weighted ones between blocks. Of course, this is not always possible.

Consequently, in the NLP version of the formulation we have |U|+|V| variables and constraints of b's (3.2) and p's (3.3), but after simplification we get |U|*|V| variables and constraints of x's (3.9). It can be clearly seen that, in (3.7), coefficient a's are dependent on each other by the help of variables b's and p's, but, unfortunately, we lost the dependency in (3.8) by putting extra variables x's. Constraints (3.2) and (3.3) guarantee that each node u or v can only be in one block, but not the same for (3.9), since x_{juv} may be 0 where b_{ju} or p_{jv} is 1 (i.e., nodes may stay in more than one block). As a result of all, the final value is not the optimal one. Even so, the result obtained from LP solution (**P-N**) can be used for an upper bound in comparisons.

3.4 NLP Solvers (GAMS)

GAMS has a special syntax to be obeyed while writing project in it, therefore a tool is developed to transform the one on our hand to the format for GAMS to understand. Some part of the mathematical representation to the problem (3.7), number of partitions K and the adjacency matrix of nodes (a_{uv}) are used to obtain the desired model for GAMS.

GAMS has an enormous number of features and options which allow it to support the most sophisticated mathematical programming and econometric applications. Fortunately, we need to know to use the language in solving our mathematical program is much less. Therefore, we will not go into the details of GAMS syntax; however, we will illustrate some basics in the following parts.

```
1:  free variable outer;
2:  positive variables
3:  b11, b12, b21, b22, p11, p12, p21, p22;

4:  equations
5:  obj, totalB1, totalB2, totalP1, totalP2;

6:  obj..
7:  +(1)*(b11*p11+b21*p21)
    +(-1)*(b11*p12+b21*p22)
    +(1)*(b12*p11+b22*p21) =e= outer;
8:  totalB1..
9:  b11+b21 =e= 1;
10:  totalB2..
11:  b12+b22 =e= 1;
```

12: totalP1..

13: p11+p21 =e= 1;

14: totalP2..

15: p12+p22 =e= 1;

16: **model** myModel /all/;

17: **solve** myModel **using** nlp maximizing outer;

Figure 8: Sample output of the tool

Table 2: GAMS Model Statistics for 10x10 Dataset

MODEL STATISTICS - 1	K=2	K=3	K=4	K=5	K=6	K=7	K=8	K=9
Blocks of Equations	21	21	21	21	21	21	21	21
Single Equations	21	21	21	21	21	21	21	21
Blocks of Variables	41	61	81	101	121	141	161	181
Single Variables	41	61	81	101	121	141	161	181
Non-zero Elements	81	121	161	201	241	281	321	361
Non-linear N-Z	40	60	80	100	120	140	160	180
Derivative Pool	10	10	10	10	10	10	10	10
Constant Pool	17	17	17	17	17	17	17	17
Code Length	391	586	781	976	1171	1366	1561	1756

Before proceeding, analyzing the model statistics might be helpful. Table above shows the GAMS model statistics for 10x10 dataset. It can be seen in the table that values of “Blocks of Equations”, “Single Equations”, “Derivative pool” and “Constant Pool” do not depend on K’s, whereas the other results systematically increase while K increases. Equation count can be computed as $U + V + 1 = 21$, where U, V and 1 corresponds to equations of totalBs, equations of totalPs (see Figure 8) and objective equation, respectively. The other important point in the table is that the number of variables. Number of variables can be computed as $(K\ x\ U) + (K\ x\ V) + 1$. As shown in Figure 8, the number of b variables is KxU and the number of p variables is $K\ x\ V$. 1 stands for the objective function variable (outer).

3.4.1 Defining Decision Variables

Figure 8 shows the sample model of GAMS for $K = 2$. In GAMS model, there must always be free variable(s) defined to represent the objective function value(s). "**free variable** outer;" (line 1) plays this role. In addition to that, the first main part of GAMS input file is a declaration of the decision variables by type (line 2-3). We define them at the top of all, since such declarations must precede any use of the variables. Decision variables of b's and p's correspond to variables of constraints (3.2) b_{ju} and (3.3) p_{jv}, respectively.

Allowed types and corresponding GAMS keywords are as follows:

Table 3: Types and GAMS Keywords

Type	**GAMS Keyword**
unrestricted (continuous) variable(s)	**free variable(s)**
nonnegative (continuous) variable(s)	**positive variable(s)**
non-positive (continuous) variable(s)	**negative variable(s)**
0-1 variable(s)	**binary variable(s)**
nonnegative integer variable(s)	**integer variable(s)**

For the keyword, we have chosen **positive variables** instead of **binary or integer variables**, because nlp solvers only support continuous variables.

3.4.2 Defining Equations (Objectives and Constraints)

The objective function and main constraints of GAMS mathematical programs are entered as "**equation(s)**" (Figure 8 line 4). Two steps are required. First, one or more equation(s) statements declare names for the equations of the model (line 5).

The second part of defining an equation is to add detail on each declared equation name in a separate statement beginning "equationname..**"** (i.e., like "obj..**"** for the objective function name) and continuing with left-hand side and right-hand side expressions separated by one of the following operators: "=e=", "=l=", "=g=", which

are explained in Table 4 (line 6-15). Our model generator tool use only "=e=" operator in its equations.

Table 4: Equation Types and GAMS Operators

Equation Type	Operator
equals	=e=
less than or equal to	=l=
greater than or equal to	=g=

One such equation always sets the objective function equal to the "free" objective value variable. For this purpose, in line 6-7, "obj.." equation set the "outer" variable to the objective function.

The syntax of detail statements of equations follows the pattern of languages with + for addition, - for subtraction, * for multiplication, / for division. Parentheses may be added to group quantities or aid readability. (Line 7,9,11,13,15)

For objective function (obj..), one part of the mathematical programming formulation of the problem (3.7) has been used in the figure. As shown in the Figure 8, constraints b_{ju} (3.2) and p_{jv} (3.3) are named in the form of totalBu and totalPv, respectively, and their equations are taken position right after the objective function equation.

3.4.3 Model Statements

GAMS can define many models within a single file by collecting different combinations of equations under different names. That is why the user is required to give a name to his/her model even if there is only one.

For simple cases, this is accomplished with the statement "**model** modelname /all/;". As noted, for our case, we have also only one model named "myModel" and defined in line 16.

3.4.4 Solve Statements

GAMS does not solve any problems itself. Instead it translates the model into the input required by one of several "solvers", listed in Figure 6. A solve statement in the format

"**solve** *model_name* **using** *solver_type m(ax/in)imizing objective_value_variable*;"

invokes a solver where *model_name* is the declared name of the model, *objective_value_variable* is the free variable representing the objective function value, *maximizing* can be *minimizing* as well, and *solver_type* is one of the following:

Table 5: GAMS Types and Descriptions

Type	Description
LP	exact solution of a linear program
MIP	exact solution of an integer linear program
RMIP	solution of the LP relaxation of an integer linear program
NLP	local optimization of a nonlinear program over smooth functions
DNLP	local optimization of a nonlinear program with non-smooth functions
MIDNLP	local optimization of an integer nonlinear program with nonlinearities all in the continuous variables
RMIDNLP	local optimization of the continuous relaxation of an integer nonlinear program with nonlinearities all in the continuous variables

In line 17, values of model_name (myModel), solver_type (nlp), objective_value_variable (outer) and (maximizing) are specified. There is a default solver for each of these model types. For default NLP solver LINDO is chosen under Options on the File menu (Figure 9).

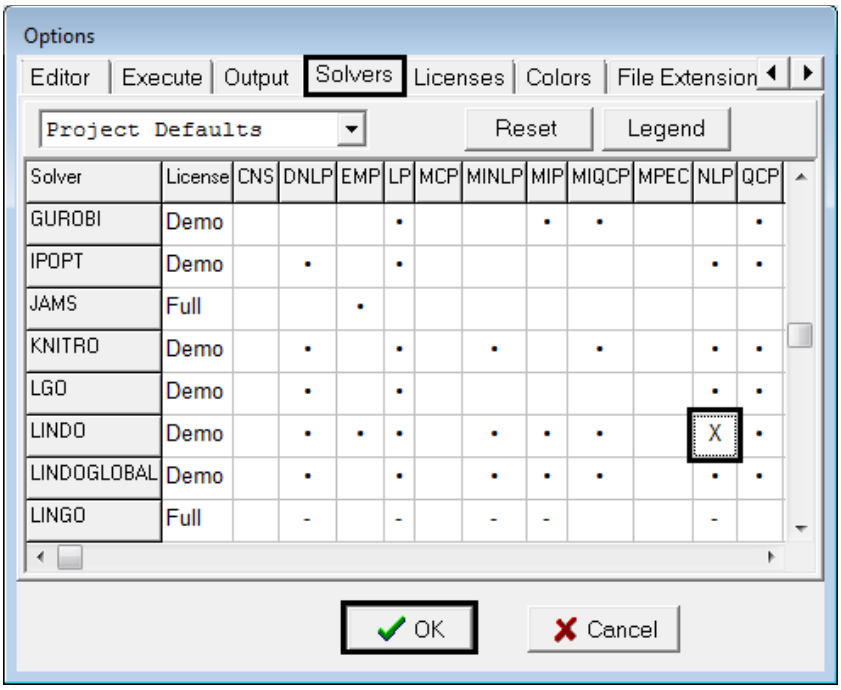


Figure 9: Default NLP Solver Selection

3.4.5 Output

Once all errors are corrected, the “SOLVE SUMMARY” part of the .lst file details the results of the optimization (Figure 10). In the figure, to be more understandable, we have grouped some parts and made important texts bold.

S O L V E S U M M A R Y

MODEL	**myModel**	OBJECTIVE	**outer**
TYPE	**NLP**	DIRECTION	**MAXIMIZE**
SOLVER	**LINDO**	FROM LINE	22

**** SOLVER STATUS	1 Normal Completion	
**** MODEL STATUS	1 Optimal	
****** OBJECTIVE VALUE**	**2.0000**	
RESOURCE USAGE, LIMIT	0.312	1000.000

ITERATION COUNT, LIMIT	39	2000000000
EVALUATION ERRORS	NA	0

LINDO Jul 4, 2012 23.9.1 WIN 33934.33953 VS8 x86/MS Windows
LINDO Driver
Lindo Systems Inc, www.lindo.com
Lindo API version 7.0.1.421 built on Feb 9 2012 18:11:14

Solution = 1.99999999999999
Best possible = 2
Absolute gap = 7.105427357601E-15
Relative gap = 0

Table 6: Values of Equations

	LOWER	**LEVEL**	**UPPER**	**MARGINAL**
---- EQU obj	.	.	.	-1.000
---- EQU totalB1	1.000	**1.000**	1.000	1.000
---- EQU totalB2	1.000	**1.000**	1.000	1.000
---- EQU totalP1	1.000	**1.000**	1.000	2.000
---- EQU totalP2	1.000	1.000	1.000	EPS

Table 7: Values of Decision Variables

	LOWER	**LEVEL**	**UPPER**	**MARGINAL**
---- VAR outer	-INF	**2.000**	+INF	.
---- VAR b11	.	**1.000**	+INF	.
---- VAR b12	.	**1.000**	+INF	.
---- VAR b21	.	**.**	+INF	-2.000
---- VAR b22	.	**.**	+INF	-1.000
---- VAR p11	.	**1.000**	+INF	.

---- VAR p12	.	**.**	+INF	-1.000
---- VAR p21	.	**.**	+INF	-2.000
---- VAR p22	.	**1.000**	+INF	.

Figure 10: Solve Summary

Table 6 reviews results for model equations. “LEVEL” values are given for each objective and constraint. According to the “LEVEL” values, we can say that all the constraints are satisfied.

Table 7 lists the results for all decision variables. These reports show the final “LEVEL” for each variable along with any upper and lower bounds and a “MARGINAL” value. In the table, the final “LEVEL” for *outer* value is equal to 2. Therefore, L_0 value is set as 2. Addition to the L_0, we must compute the sum of all edges in equation (3.4) to obtain L value.

The “LEVEL” values of decision variables are all meaningful. *b11, b12, p11, p22* are set to 1, meaning that 1st and 2nd element of U and 1st element of V are put into the 1st block, and only 2nd element of V is put inside the 2nd block. When we check the constraints, we see that all the final values of decision variables satisfy them.

CHAPTER 4

GENERIC ALGORITHMS

This chapter consists of four sections. It starts with the introductory sections of genetic algorithm (GA) and simulated annealing (SA) methods, and continues with their implementations and solution details for the *k-way partitioning of signed bipartite graph problem.*

Because of the difficulty in finding the optimal solution, two of our solution approaches are GA and SA for this specific problem. In general, these approaches are capable of finding optimal or sub-optimal solutions for optimization problems. While developing generic algorithms, selecting proper initial values for variables and providing enough randomness in operations are some of the critical issues to take into consideration. For instance, if on some selection points, deterministic selections are to be made instead of random selections, it is more possible that the solution will get stuck at a local maximum or minimum and will not reach to the desired optimal or suboptimal solutions.

4.1 Introduction to Genetic Algorithms

A genetic algorithm [24] is a heuristic algorithm, inspired by evolutionary processes of ecological systems, that finds optimal (or near-optimal) solutions to complex optimization problems. Genetic algorithms are particular class of evolutionary algorithms.

In genetic algorithms, possible solutions to the problem are coded in chromosomes. A “chromosome” (or “individual”) can be designed as a string, binary digit or other symbols that corresponds to a solution of the problem at hand. To give a trivial

example, suppose our goal is to find a value of x between 0 and 255 which makes the result for function $f(x) = x^2$ maximum. In this example, our potential solutions are integer values from 0 to 255, meaning that our chromosome can be represented as 8-digit binary strings (i.e., values of chromosomes are from 00000000 to 11111111). GAs use sets of these chromosomes, called "populations" that evolve through "generations", by the operators of "crossover" and "mutation", in order to discover optimal solution to the problem.

The fitness function of GA analyzes “genes” in the chromosomes, makes some qualitative assessment and provides a meaningful and comparable fitness value for that solution. Basically, thanks to the fitness function, candidate solutions pass to the next generation of solutions by discarding solutions with a “poor” fitness and accepting any with a “good” fitness value.

4.1.1 Outline of the Basic Genetic Algorithm

- Construct a large initial population of chromosomes by generating randomly attempted solutions to a problem
- Do the following till you accomplish (i.e., satisfactory fitness level has been reached) or run out of time
    - Evaluate each fitness of the solutions
    - Keep a subset of these solutions (take best possible solutions)
    - Use these solutions to generate a new population, the children chromosomes may be formed by operating the *crossover* operator to the two selected parents (*selection)*, and diversified by applying *mutation* techniques.

4.1.2 Representations

Representation is a way of encoding the possible solutions in evolutionary computation methods. Genetic representation can encode different aspects of the solution as the digits in the represented strings.

4.1.2.1 Fixed-Length Representation

Genetic algorithms mostly use fixed-length representations. It is easier to crossover chromosomes when they are fixed size.

4.1.2.1.1 Binary Representation

In binary representation, array of bits is used as the standard. In this representation, we encode solutions as binary strings: sequences of 1's and 0's.

4.1.2.1.2 Real-Value Representation

Another representation approach is real-value representation. It encodes solutions as arrays of integer or decimal numbers, where the real-values at each position of chromosome represent the value of some behavior of the solution.

4.1.2.1.3 Letter Representation

A third approach is to represent individuals in a GA as strings of letters, where each letter again stands for a specific aspect of the solution.

4.1.2.2 Variable-Length Representation

Variable length representations are inconvenient in most of the cases, because of the cost of the complexity of the crossover implementation.

4.1.3 Selections

There are many different techniques which a genetic algorithm can use to select the individuals to be copied over into the next generation. Some of them are as follows:

4.1.3.1 Roulette-Wheel Selection

Conceptually, roulette-wheel selection, a method favorite to choose parents, is represented as a game of roulette. Each individual in the game gets a slice of the wheel, but more fit ones get larger slices than less fit ones, meaning that the probability of a chromosome to be selected as a parent is proportional to its fitness.

4.1.3.2 Elitist Selection

The fittest members of each generation are guaranteed to be selected.

4.1.3.3 Rank Selection

Each individual in the population is assigned a numerical rank based on fitness, and selection is done based on this ranking.

4.1.4 Reproduction Techniques

There are two basic reproduction strategies, which are crossover and mutation. There is a chance that the chromosomes of the two parents are copied unmodified (i.e., without applying crossover or mutation) as offspring.

4.1.4.1 Crossover

Crossover is a reproduction technique to generate two offspring from two selected parents. The chromosomes of the two parents are recombined according to some techniques to form offspring. In the following, some of the mostly used techniques are described:

4.1.4.1.1 Single Point Crossover [25]

As shown in Figure 11, randomly one point in the chromosomes is chosen. The binary strings of the two parents are cut at this specific point and the two substrings are exchanged.

After crossover operation, offspring 1 is head of chromosome of parent 1 with tail of chromosome of parent 2, and similarly, offspring 2 is head of 2 with tail of 1.

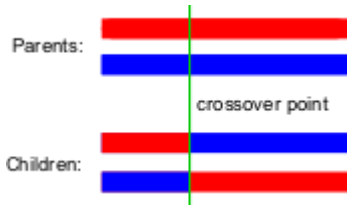


Figure 11: Single Point Crossover (Wikipedia)

4.1.4.1.2 Two-point Crossover (Multi Point Crossover)

Randomly two points (or more) in the chromosomes as drawn with green lines in Figure 12 are chosen. This type of crossover provides that genes at the head and genes at the tail of a chromosome are always split when recombined.

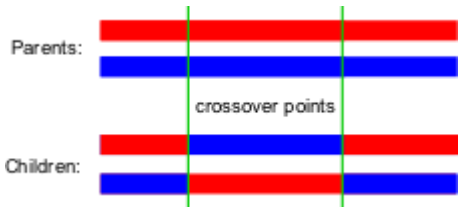


Figure 12: Two-point Crossover (Wikipedia)

4.1.4.1.3 Uniform Crossover

Each bit in the chromosomes is compared between two parents and one of two is chosen based on a certain proportion, like 0,5.

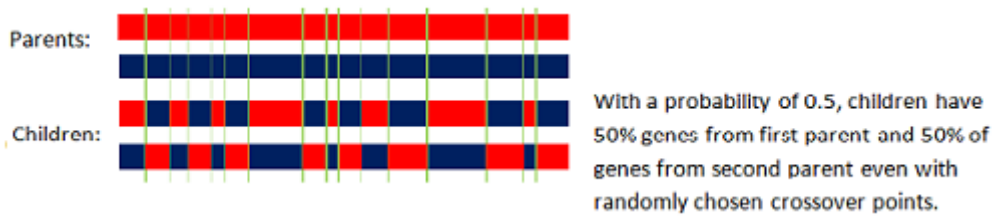


Figure 13: Uniform Crossover (Wikipedia)

4.1.4.2 Mutation

Mutation is a reproduction mechanism, which generates new offspring from single parent. Mutation operator is performed upon the children chromosomes to create the final chromosomes of the new generations. Each binary digit of the chromosome is subject to inversion (from 1 to 0 and vice versa) under a given probability (most of the time small). Mutation also gives opportunity new genes to be introduced in the population.

4.2 Introduction to Simulated Annealing

Simulated annealing (SA) is a generic probabilistic meta-algorithm used to find an approximate solution to global optimization problems, which was introduced by Kirkpatrick (1983) [26]. It is inspired by annealing in metallurgy which is a technique of controlled cooling of material to reduce defects. The material is subjected to high temperature and then cooled gradually. The gradual cooling process makes the material stronger and there exist few weak points in it. It is similar to the achieving global maximum by discarding the local ones. If the cooling process is rapid, then it does not produce strong object. Some parts may be broken easily whereas some areas may achieve to the local optimum strength.

4.2.1 Outline of the Basic Simulated Annealing

- Initialize temperature T, epsilon ε, alpha α
- Generate a random initial solution as current solution S_c
- Do the following till $T < \varepsilon$ or run out of time
    - While *stopping criteria not met* do
        - Find the neighbor of the current solution S_n
        - Compute $\Delta = f(S_n) - f(S_c)$ (i.e., f: fitness function)
        - Randomly generate a real number r from 0 to 1
        - If $\Delta < 0$ or $(e^{-\Delta/T} > r)$ then $S_c = S_n$
    - Reduce T by multiplying with α

SA starts with some solution that is totally random, and changes it to another solution that is similar to the previous one. Newly generated solutions are generally chosen randomly, though more sophisticated methods can be applied. If this solution is a better solution, it will replace the current solution. If it is a worse one, it may be chosen to replace the current solution with a probability that depends on the temperature (i.e., cooling process, T decreases with time) and the distance Δ (i.e., difference between new (worse) solution and the old one) parameters. As the algorithm progresses, the temperature parameter decreases by multiplying alpha α, giving worse solutions a lesser chance of replacing the current solution.

SA uses random numbers in its execution, therefore in every run of the algorithm; we can come up with a different solution. It produces a sequence of solutions; each one is derived from the other by slightly altering it, or falls back to the original state by discarding the changes.

The difficulty in search algorithms is that while they rapidly find a local maximum, it cannot get to the global maximum. SA allows worse solutions at the beginning so that it avoids converging to a local maximum rather than the global maximum. SA achieves to the global maximum most of the time through the introduction of two tricks [26]: the first trick is metropolis algorithm [29] and the second trick is to lower the temperature. If *T* is large, many worse solutions are accepted and a large part of the solution space is discovered. Lowering the temperature limits the size of the allowed worse solutions.

4.3 Genetic Algorithm (GA) Implementation

4.3.1 Representation Mechanism

For representation arrays of integer-values have been used. We have had two arrays; which holds the block numbers of nodes U and V. Arrays are fixed size and the values correspond to integer values from 1 to K.

4.3.2 GA Details

In this part, we present the details of the algorithm.

We have set 500 for population size and the number of iterations was 50. From old to new population we transferred the best 5% of the solutions (elitist selection). The 90% of the solutions were selected with roulette-wheel selection mechanism. Certain proportion used for uniform crossover was 6/10. The remaining 5% of solutions were randomly generated in new generations. Mutation rate was 0.01%.

Algorithm 4.1: Genetic Algorithm

Input : The signed weighted bipartite graph $G = (U \cup V, E)$ and K

Output: Maximal L value and corresponding partitions of nodes.

1: Initiate population size $popSize$, and number of iterations I

2: Generate current population (P_c) with random solutions

3: **while** $true$

4: For all solutions in P_c compute fitness **(Algorithm 4.2)**

5: Sort solutions in the P_c

6: **if** $iterationCount > I$ **then exit while**

7: Generate next population P_n **(Algorithm 4.3)**

8: $P_c \leftarrow P_n$

9: **end while**

10: **print** the solution at the top of P_c

Algorithm 4.2: Fitness Calculation

Input : One of the solutions of P_c, adjacency matrix A (a_{uv})

Output: Fitness value of the given solution

1: $fitnessValue \leftarrow 0$

2: **for each** edges in graph G (let's say from u to v)

3: **if** u and v are in the same block **then**

4: $fitnessValue \mathrel{+}= a_{uv}$

5: **else**

6: $fitnessValue \mathrel{-}= a_{uv}$

7: **end if**

8: **end for**

9: **return** $fitnessValue$

Algorithm 4.3: Generate Next Population

Input : Current population P_c , elitist selection rate E , roulette-wheel selection rate RW, proportion rate of uniform crossover UC, mutation rate M

Output: Fitness value of the given solution

1: best $(popSize * E)$ solutions taken as is from the old to the new population

2: $(popSize * RW)$ solutions are selected by using *roulette-wheel selection* and *uniform crossover* is applied onto them to generate children

3: $(popsize$ - $(popSize * E) - (popSize * RW))$ solutions are generated randomly

4: apply *mutation* by randomly changing the $(U + V) * M$ genes of all chromosomes except those chosen by *elitist selection* (i.e., mutation changes the block of node randomly)

The variables that affect the outcome of the algorithm are $popSize$, E, RW, UC and M (**Algorithm 4.3**). We adjust these variables and see how the outcome is affected. It is also important to set the termination criteria. We change the number of iterations the algorithm does to see how the final result and the algorithm's speed are affected. Considering all of these constraints, we tried to set the best values for variables.

4.4 Simulated Annealing (SA) Implementation

In this section, we will explain the details of the SA algorithm. Our SA algorithm implementation is standard; yet, we present the implementation of the algorithm and give some brief information about the algorithm.

Algorithm 4.4: SA Algorithm

Input : A signed weighted bipartite graph $G = (U \cup V, E)$ and K

Output: Maximal L value and corresponding partitions of nodes.

1: Initiate temperature (T) and epsilon (ε)

2: Generate a random solution (S_c), final solution $(S_f) \leftarrow S_c$

3: Compute the fitness of S_c $(f(S_c))$ **(Algorithm 4.2)**

4: **while** $T > \varepsilon$

5: randomly select a node $(nod1)$ from U or V

6: change block of $nod1$ to the best possible place **(Algorithm 4.5)**

7: Iteratively compute the $f(S_n)$ from $f(S_c)$

8: $\Delta \leftarrow f(S_n) - f(S_c)$

9: randomly generate a real number r from 0 to 1

10: **if** $\Delta < 0$ or $(e^{-\Delta/T} > r)$ **then** $S_c \leftarrow S_n$ **end if**

11: **if** $f(S_c) > f(S_f)$ **then** $S_f \leftarrow S_c$ **end if**

12: $T \mathrel{*}= \alpha$

13: **end while**

14: **print** S_f

Algorithm 4.5: Find Best Place for the Selected Node

Input : A signed weighted bipartite graph $G = (U \cup V, E)$, K, S_c and $nod1$

Output: Best block for $nod1$

1: initiate max value $(maxVal) \leftarrow -INF$

2: **for** all possible blocks numbered j: $1\ to\ K$, except the current block of $nod1$

3: generate a solution S_{dummy} by putting the $nod1$ into $block_j$

4: recalculate $f(S_{dummy})$ (iteratively from S_c)

5: **if** $maxVal < f(S_{dummy})$ **then**

6: $maxVal \leftarrow f(S_{dummy})$

7: output Value $(outVal) \leftarrow j$

8: **end if**

9: **end for**

10: **return** $outVal$

Note that in **Algorithm 4.4** line 6 and **Algorithm 4.5** line 4, computations have been done iteratively. For each movement of the node, we can compute the new fitness value just by traversing all of connected edges of the selected node and adding the difference caused by the movement (Figure 14).

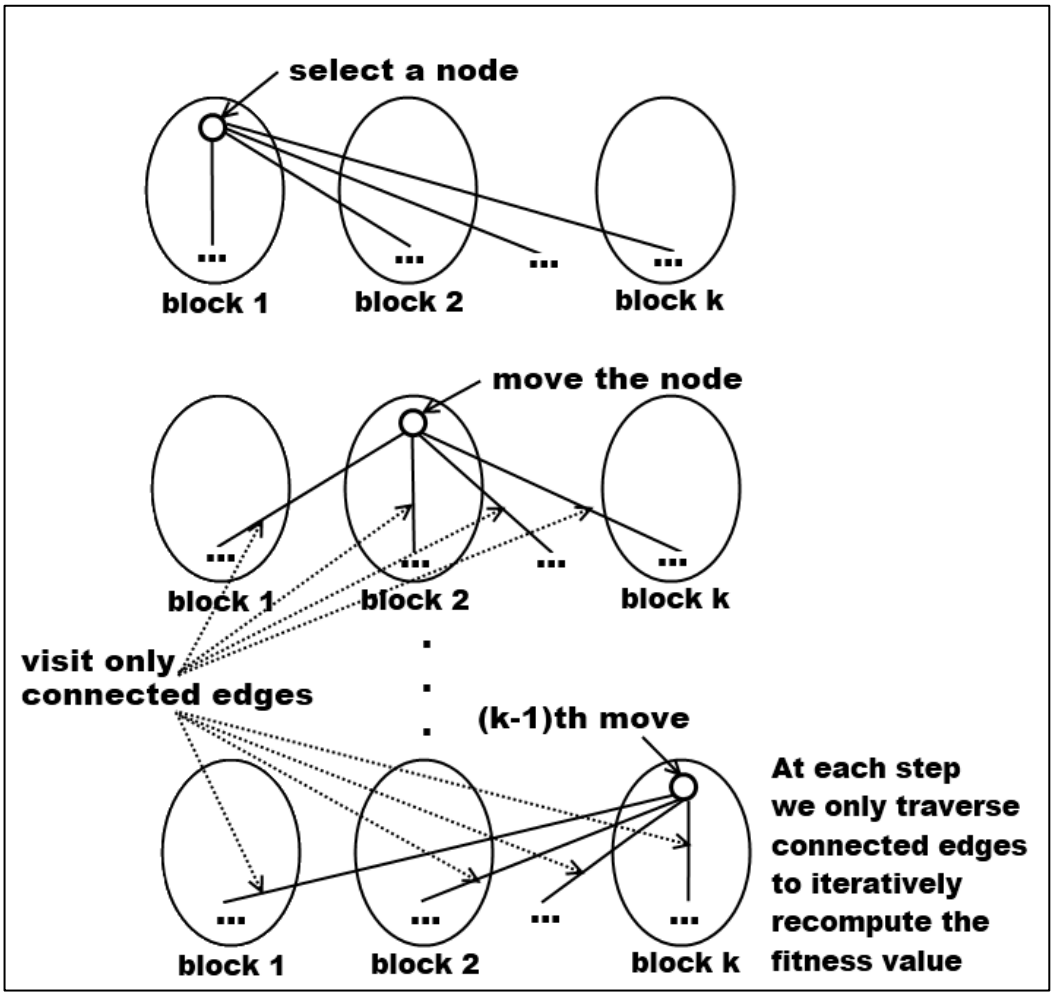


Figure 14: Iteratively Fitness Value Computation

The variables that affect the outcome of the algorithm are the initial temperature, the rate at which the temperature decreases (alpha) and the stopping condition of the algorithm (epsilon). We adjust these values to see how the algorithm responds.

CHAPTER 5

MOVE-BASED HEURISTIC ALGORITHMS

In order to solve k-way partitioning of signed or weighted bipartite graph problem, mathematical methods and several generic heuristic techniques have been explored in previous chapters. However, in our study, we have mainly concentrated on move-based heuristics. In our experiments move-based heuristics worked very effectively, in obtaining good partitioning (satisfying most of the maximization and minimization constraints defined in the introduction section) in a very fast way.

In this chapter we will give the details of 5 different move-based heuristic algorithms for our problem. In the first section, we will talk about the common characteristics of the algorithms. The original algorithm given in [3] with some extensions will be explained in the second section. Instead of 2-way partitioning, k-way partitioning is used and K is taken as an input to the algorithm. First one is also named as MBH1 (move-based heuristic version.1). We will continue with the 2nd, 3rd and 4th versions of algorithms which have some differences from the 1st one. In each of them, we have tried to find better version of 1st algorithm by changing selection mechanisms, which will be detailed later. We have named them as MBH2, MBH3, and MBH4, respectively (names correspond to their versions). 5th one has the same structure with 1st but has some extra heuristics in it. In the last version, we have applied special-cut heuristic which halved the running time of the MBH1. Since it is the optimal one, as a name, opt-MBH (optimized-MBH) has been chosen.

5.1 Common Characteristics of the MBHs

Our incremental method works for multi-partitioning as well as bi-partitioning. For the extension of our algorithm from 2-way to the k-way partitioning, some differences from the approach given in [3] can be provided as follows:

1. At the beginning, in 2-way partitioning, the authors of [3] randomly distributed the nodes U and V in between 2 blocks, but in our case we randomly distributed the nodes among K blocks.
2. In bi-partitioning, in paper [3], they have only one option in their hand, which was to move the selected node to the other block, thus, the job was easy. However, from our side, it can be seen that there are $(K-1)$ different possible blocks to move the selected node. In order to make a good move, we had to make some calculations among K-1 possibilities and moved the node to “the best block”. We can describe “the best block” as a block the selected node was put into that with the purpose of maximizing the result value L at that moment.

In generally, move-based heuristics are applied in our work as follows:

We place vertices randomly into blocks at the beginning. Then, through iteration, the node with the highest gain value is selected and moved to another block. It is checked if the move of each vertex to another block increases the result value L or not. If the value is increased, found value is set to L value. After each movement the related node is locked. Until all the nodes are locked, the iteration continues. Locking a node means that marking that node, and it is not moved again till all nodes are marked. After all are locked, the rise in the result value L is checked. If so, we restart the iteration by configuring the initial state with the best state found in the previous iteration. Otherwise, we end the iterations and print the best solution to the output. Details and variations of the algorithms will be presented in the next sections. **Algorithm 5.2** gives this algorithm.

When we compare the results we use L value for maximizing the objective function. As a result, the higher the L value, the better the clustering.

Our move-based algorithms have worked several times try to find the best clustering for the given partitioning value K. For this purpose, we repeat process R times. In this way, local search restarts from randomly selected R points. This helps the algorithm to come close to the global maximum.

In order to measure good clustering of bipartite graphs, as in [3], we have defined a gain computation function **(Algorithm 5.1)** that changes gains of all nodes as the vertices placed into blocks as described in Section 1.

Algorithm 5.1: Gain Computation of All Nodes

Input : A signed weighted bipartite graph $G = (U \cup V, E)$, K and current solution S_c

Output: Gains of all nodes

1: clear all gains
2: **for** all edges from u to v, where $u \in U, v \in V$
3: **if** u and v are in the same block **then** $mult1 \leftarrow -1$
4: **else** $mult1 \leftarrow 1$ **end if**
5: **for** $j\ from\ 1\ to\ K$
6: **if** the place of the node u is equal to j **then** $mult2 \leftarrow 1$
7: **else** $mult2 \leftarrow -1$ **end if**
8: $gain[u][j] += (a_{uv} * (mult1 + mult2))$
9: **if** the place of the node v is equal to j **then** $mult2 \leftarrow 1$
10: **else** $mult2 \leftarrow -1$ **end if**
11: $gain[v][j] += (a_{uv} * (mult1 + mult2))$
12: **end for**
13: **end for**

The aim of **Algorithm 5.2** is to calculate the gains of all vertices. We traverse all the edges in the graph (Line 2) and for the vertices of each edge, the gain is calculated and added to the sum (Line 14). This is done for all *j* possible places (Line 5) that these nodes can be moved to.

In gain calculation, there are 4 possibilities. Two values, namely *mult1* (Line 4) and *mult2* (Line 6) are used for that purpose as follows:

- If both vertices are in the same block, and the edge between them is positively weighted, then moving either one will produce negative effect on gains.
- Similarly, if the vertices are in different blocks and the edge between them is negatively weighted, then putting them into the same block will also reduce the gain.
- If the vertices are in the same block, but the edge between them is negative weighted, then, moving one of them to a different block will increase the gain.
- Finally, if two vertices in different blocks, but the edge between them are positively weighted, then moving them into the same block will increase the gain.

Figure 15 depicts the gain computation on a simple example. As stated on the figure, it is clear that the movement of the selected node to the 3rd block makes the result value L bigger. As we always move the nodes to the best possible place according to the gains, in this case, it is wise to move the node to the 3rd block.

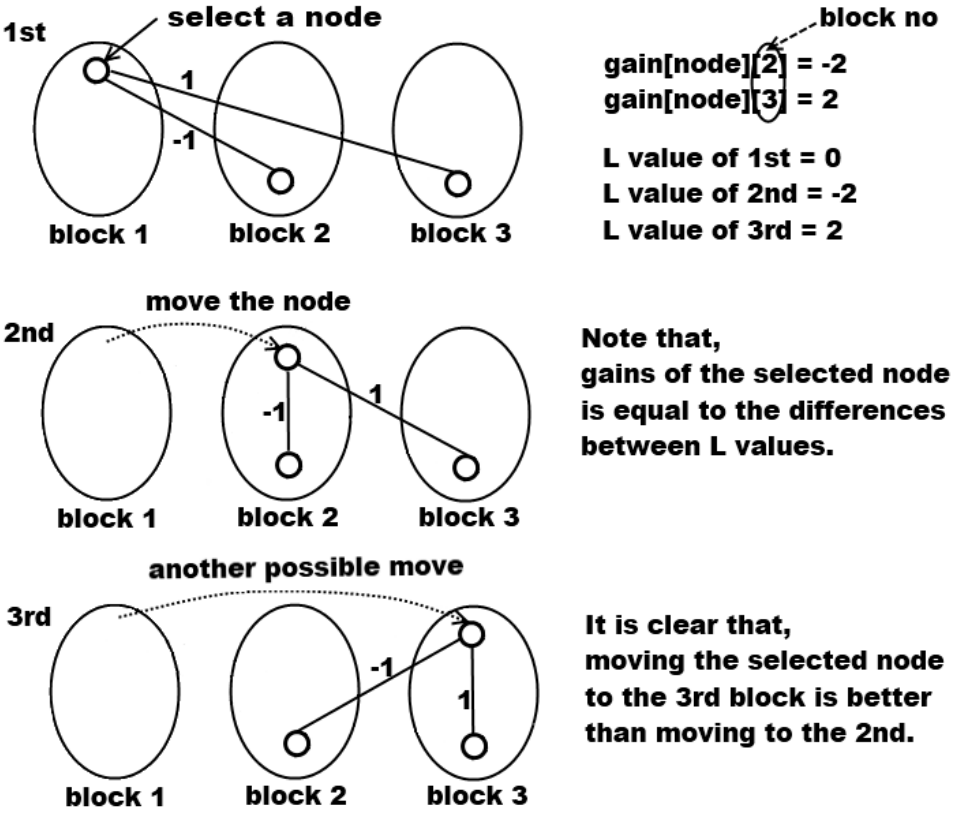


Figure 15: Gain Computation

5.2 MBH1 Implementation

Algorithm 5.2 gives MBH1 algorithm.

Algorithm 5.2: Move-Based Heuristic version.1 (MBH1)

Input : A signed weighted bipartite graph $G = (U \cup V, E)$, K and the number of random start R.

Output: Maximal L value and corresponding partitions of nodes.

1: $L \leftarrow -\infty$
2: **while** $R - -$
3: Initially, place each node into block 1 to K randomly
4: $L \leftarrow$ Current RESULT, $L' \leftarrow L$
5: **do**
6: $L \leftarrow L'$
7: Compute gains of all nodes **(Algorithm 5.1)**
8: **do**
9: *nod1* ← select the unlocked node with max gain
10: *blck1* ← select the best block for *nod1*
11: place the *nod1* into *blck1*
12: update gains of *nod1*'s neighbors
13: L'' ← New RESULT
14: lock *nod1*
15: **until** all nodes are locked
16: **while** $L' < L''$
17: $L' \leftarrow L''$
18: **end while**
19: **print** L

In **Algorithm 5.2**, L represents the result of the objective function detailed in chapter 3, and K is the cluster number. The algorithm takes K value as an input. As we previously explained in the mathematical computation (chapter 3) and in the fitness calculation of generic algorithms **(Algorithm 4.2)**, we skipped L value calculation in this part.

Move-based algorithms strictly incline but it is always possible to strike at local maximum. In order to avoid this problem we repeat the process several times. In **Algorithm 5.2** the R value in line 2 is used for that purpose.

While we were moving nodes from one block to another, we updated the gain values of the related nodes (**Algorithm 5.2** Line 14).

5.3 MBH2, MBH3, MBH4 Implementations

MBH1 and MBH2, MBH3, MBH4 are all the same except the node selection mechanisms (see **Algorithm 5.2** line 9).

In MBH1 algorithm, we select the node with the highest gain value among nodes in U and V.

We changed the order of selection in MBH versions 2-4. Our intention was to expand the search space and not to strike at local maximums, while creating the variations of selection mechanism given in MBH1, but as detailed in experimental results, the results are far beyond the expectations. In MBH2, firstly, nodes of U and then nodes of V are selected in decreasing order of gain values. In MBH3, exactly the opposite way, we picked nodes from set V and then from set U. Finally, in MBH4, we mixed MBH2 and MBH3. We chose nodes in the order of one from U and one from V. See the figure below which clarifies it and shows the differences.

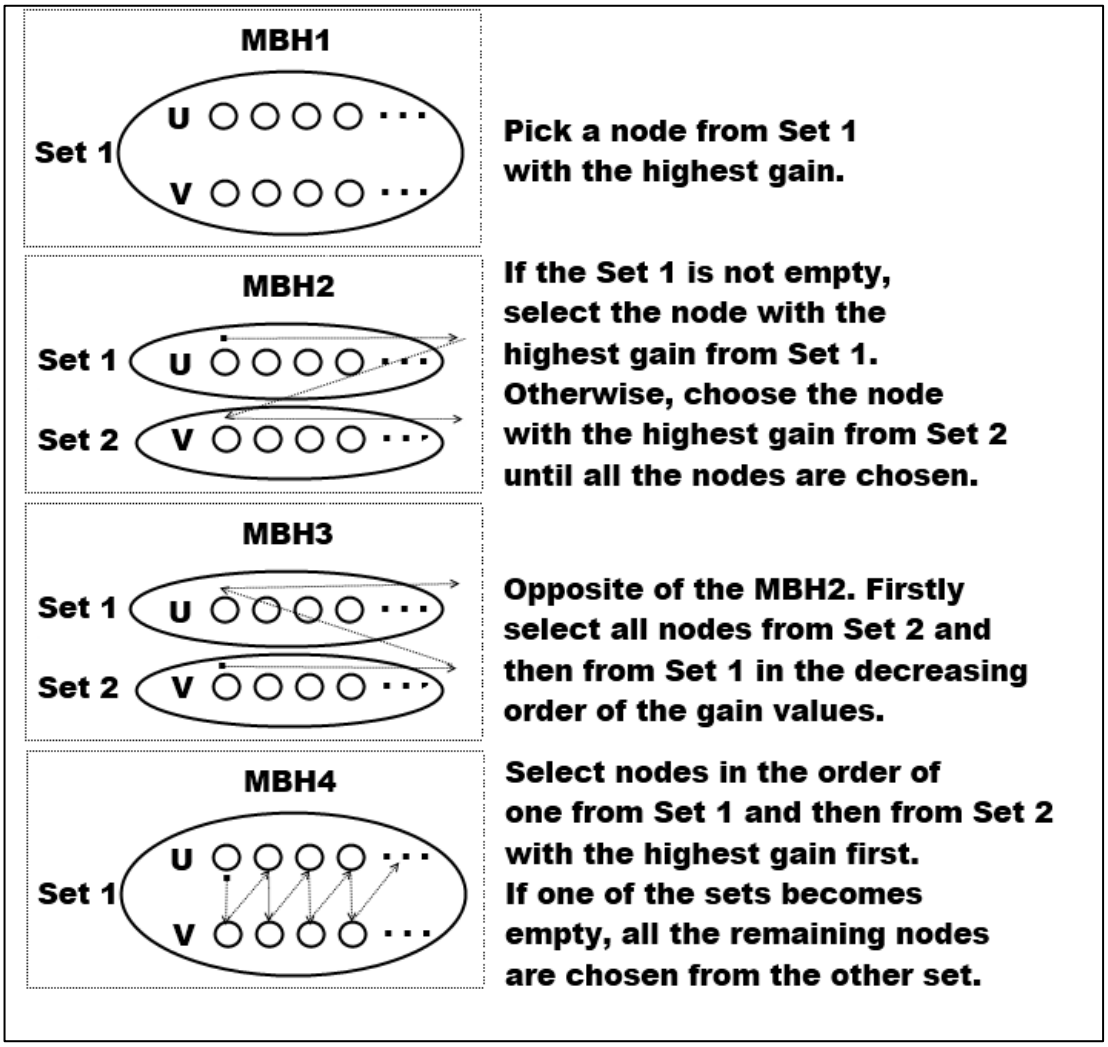


Figure 16: Traversing Differences of MBHs

5.4 Opt-MBH Implementation

Opt-MBH, as the name declares, is the optimized version of MBH1. While we were analyzing the outputs of MBH1, we saw that there have been some unnecessary moves in the process of MBH1 algorithm. We clearly observed these redundant moves by displaying a sample run of the algorithm on the chart. Figure 17 shows one of the sample run of the MBH1. In order to express some justifications of Opt-MBH, we will give all the related and critical points and details of the graphics in this chapter instead of in chapter 6.

These figures has been obtained from questionnaire dataset experiment for K=9. The questionnaire dataset had been selected, since it is the biggest dataset to clearly show the differences. Randomly start count was 3 in this experiment to differentiate the lines from each other.

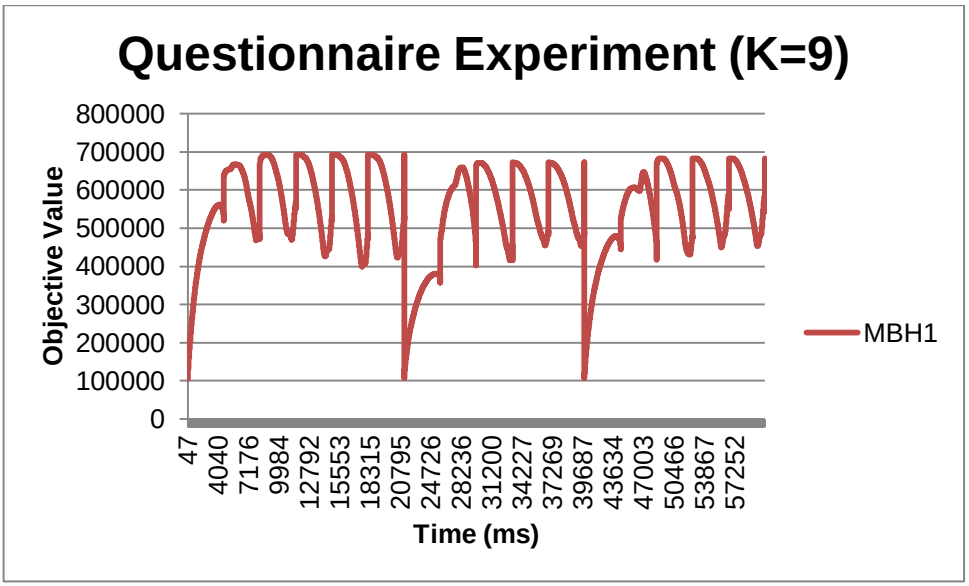


Figure 17: Questionnaire Experiment (K=9), MBH1 Stats

In Figure 17, we see that objective values are increasing and decreasing in a systematic way. Decreasing parts are not necessary for our problem, since we have been trying to find the global maximum. Furthermore, as the chart presents, calculation of the descending values is really time consuming. As a result of all, we have intended to remove the declining parts from the chart to improve execution time. By detecting the values when they started to fall below the local maximums, we managed to cut the unnecessary parts as shown in Figure 18. Note that the elapsed time has fallen below the half of it (approximately 57000 to 20000 ms).

To explain values of the charts in detail, it can be stated that the values are lower than 0 corresponds to initial solutions that are randomly generated at the beginning of the iterations. Random initiation count R=3 is clearly understand from the graphics.

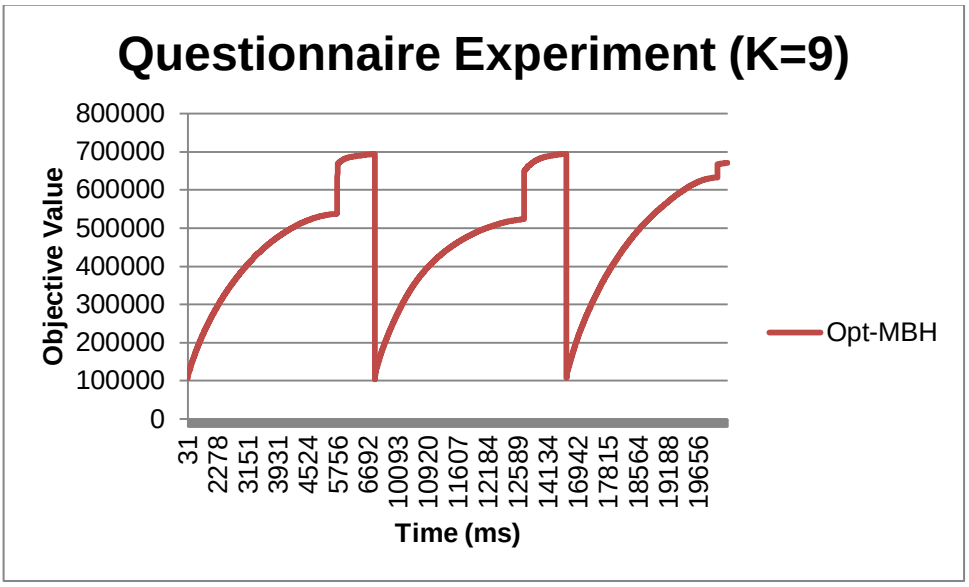


Figure 18: Questionnaire Experiment (K=9), Opt-MBH Stats

The main reason was hidden in the **Algorithm 5.2** line 15. The original version of move-based algorithm (MBH1) locks all of the nodes (traverses all of the nodes) in each iteration. As previously explained, we detected that at some point (let's say after achieving to the local maximum), traversing the remaining part is unnecessary, even it is time-killer. So that we add some extra heuristic, this compares the current objective value L'' with the one that is already on our hand L'. If the trend of the current value tended to fall, we cut at that point without traversing the rest of nodes. **Algorithm 5.3** gives this heuristic. It is placed into the inner loop between lines 8-15 of **Algorithm 5.2**.

Algorithm 5.3: Special Cut in Opt-MBH

At each iteration, we set the $declineCount \leftarrow 0$

.

.

1: **if** $L''(current\ result) < L'(local\ maximum)$ **then**

2: $declineCount += 1$

3: **else if** $L'' == L'$ **then**

4: ; // Do Nothing

5: **else**

6: **if** $declineCount > 0$ **then** $declineCount -= 1$

7: **else** $declineCount = 0$

8: **end if**

9: **if** $declineCount > 10$ **then**

10: **exit loop**

11: **end if**

.

.

CHAPTER 6

EXPERIMENTAL RESULTS

The results of our experiments presented in this chapter are very interesting. The first 4 sections give details of the development environment and auxiliary tools which helped us throughout the work. Fifth section mainly describes the results of the mathematical methods. Sixth section puts all the results together and compares them. In 7^{th} and 8^{th} sections, some sample analyses are provided. Finally, MBH1 and Opt-MBH comparison which produced successful results using the same random data is shown in the last section.

Before starting, please note that, the quality of the clusters that have been found is not the subject of our study. As mentioned in the first chapter, nodes are distributed in an unbalanced way. After clustering, examination of the result is the concern of the social analysts, why the data is distributed so. Aim of our algorithms is just to maximize the objective function value L (3.1).

6.1 Development Environment

The series of methods expressed in Chapter 3, 4 and 5 are all implemented in C++, using the Visual Studio 2005 development environment. All these algorithms run on a commodity computer having Windows 7 x86 OS, Intel Core 2 Duo 2.00 GHz CPU, and 3 GB RAM. Thus, the running times of algorithms given in the following sections should be assessed under these specifications.

6.2 Output Controller Tool (Double Checker)

The outputs of the algorithms described in previous chapters are verified by a controller program not to make any mistakes in the experimental results. Controller program is also written in C++ with Visual Studio 2005. Inputs of the controller program are the consumed input and the produced output of each algorithm. The tool reads the adjacency matrix from the input file and the cluster positions of each node from the output file to recalculate the L value. It compares the resulting L value with the old L value in the output file, and gives “PASS” or “FAIL” as an output. “FAIL” means that there is a mistake in the algorithm to be corrected. This tool was really helpful and necessary, especially; it must be used after making changes in the codes. Thanks to the tool, we have easily detected whether or not there is an error in our implementations or in results.

6.3 Dataset Generator Tool

In this thesis, in addition to the real datasets, we have tested our algorithms with the randomly generated data. We developed an input generator tool to create randomized datasets with the help of VS2005 environment and C++ language. In our work, randomized datasets are generated to be able to experiment special cases. The tool generates datasets with dimensions 10x10, 20x20, 40x40 and 80x80. And for each of them, to see the differences and obtain varied results, sparse vs. dense, binary (i.e., -1, 0, 1) vs. arbitrary (-10, …, 0, …, 10) versions are produced. For a dataset to be dense, it is conditioned such that approximately %70 of the edges have nonzero values, other than the dense datasets (i.e., sparse ones) can have up to %20 nonzero values. As a result of generation, 4 x 2 x 2 = 16 different input sets are created. Input generator does not get anything from outside; just works with predefined parameters and gives the intended datasets as an output.

6.4 Dataset Analyzer Tool

We have written a small but useful tool with language C++ and IDE VS2005 to analyze datasets. In Table 8, we can see the output of the tool for the datasets which have been mentioned previously. At first glance, with the help of the output of the tool, we might obtain a general idea and overall vision about datasets. Using the tool, 18 datasets are analyzed as can be seen on the 1st column. **U**, **V** and **U*V** values are presented on the 2nd, 3rd and 4th columns. By traversing all of the edges, the numbers of positively weighted and negative weighted edges, the sums of all positively weighted and all negatively weighted edges are computed (column 5-10). At the end of columns, densities of the datasets can be found.

In the table, first 16 rows correspond to randomly generated datasets. 17th and 18th rows show the results of the real world data sets. In the following sections, we will use 1st, 2nd … 18th to refer to the datasets expressed in the table below.

Table 8: Characteristics of Datasets

	U	V	U*V	#ofP	#ofN	#ofT	SumOfP	SumOfN	SumOfT	Denst
1	10	10	100	31	34	65	31	-34	65	65%
2	10	10	100	30	27	57	158	-137	295	57%
3	10	10	100	9	7	16	9	-7	16	16%
4	10	10	100	9	10	19	60	-62	122	19%
5	20	20	400	121	109	230	121	-109	230	58%
6	20	20	400	122	122	244	698	-643	1341	61%
7	20	20	400	38	37	75	38	-37	75	19%
8	20	20	400	39	33	72	193	-170	363	18%
9	40	40	1600	484	448	932	484	-448	932	58%
10	40	40	1600	454	484	938	2414	-2748	5162	59%
11	40	40	1600	135	148	283	135	-148	283	18%
12	40	40	1600	137	153	290	733	-837	1570	18%
13	80	80	6400	1964	1862	3826	1964	-1862	3826	60%
14	80	80	6400	1894	1912	3806	10459	-10789	21248	59%
15	80	80	6400	580	579	1159	580	-579	1159	18%
16	80	80	6400	595	561	1156	3211	-3155	6366	18%
17	7572	48	363456	138894	132811	271705	494380	-632641	1127021	75%
18	108	696	75168	40609	25807	66416	40609	-25807	66416	88%

17th dataset contains questionnaire with 48 questions, which are applied to 7572 people. The questions were ranked between -5 and +5. The data size and the density is the biggest compared to all others.

18th dataset corresponds to US Congress (SENATE) dataset which is published publicly in www.govrack.us. From this site, we have used the *roll call votes* for the 111th US Congress Senate that covers the years 2009-2010.

The 111th Senate data contains information about 108 senators and their votes on 696 bills. We have constructed a signed bipartite graph as in [3] based on the votes of the senators on the bills.

6.5 Comparisons of NLP Solvers

Some of the NLP solvers have not worked because of license error. We have put results of only those that we have been able to test. We have also added the result of the Opt-MBH to compare results. For each execution, we have given at most 5 minutes to the solvers, because some of them took quite long time. After 5 minutes, execution was interrupted and objective value at that time has been taken. In order to get an idea about all of NLP solvers in GAMS, all of them are tried with the smallest, 1st, dataset.

Table 9: Results of NLP Solvers for 1st Dataset

1st	OptMBH	COUENNE	SCIP	BARON	CONOPT	LINDO	LINDO2	MINOS	SNOPT
K=2	**35**	35	35	35	33	33	33	33	29
K=3	**43**	43	43	43	37	37	37	35	28
K=4	**43**	43	43	43	43	37	37	35	27
K=5	**43**	43	43	43	37	31	31	35	25,5
K=6	**43**	43*	39*	37*	37	37	37	35	25
K=7	**43**	43*	39*	37*	37	37	37	35	37
K=8	**43**	43*	41*	39*	39	37,3	37,3	35	24,5
K=9	**43**	43*	43*	39*	39	37,2	37,2	35	24,25

Solvers IPOPT, KNITRO and PATHNLP found irrational results (like 0, 1, and 1.5), so those results are not put on the table. We sorted the solvers according to their success. Opt-MBH proved itself as an upper bound to the mathematical solutions. Solvers COUENNE, SCIP and BARON found better results but their execution time were considerably high. Results with asterisk (*) correspond to those takes more than 5 minutes and are interrupted. Solvers CONOPT, LINDO, LINDOGLOBAL (LINDO2), MINOS and SNOPT gave solutions in a very short period of time; however, results were not so reliable.

We tried COUENNE, SCIP and BARON for bigger datasets, but we could not get any result in reasonable time. For 17th they even did not start execution, because of "Time-limit exceeded for parsing phase" error. Since we needed solutions in reasonable time, we left the rest for future work.

Each NLP solver uses various methods to obtain the desired value. One of the NLP solvers, BARON, prints the used methods in its output as in Figure 19. Figure shows the result of BARON which uses the model of 1st dataset and K=5. As it is understood from the output, LPs are used in preprocessing phase. It is also clear that before starting iterations some feasible solutions are obtained in that phase. We can see the list of iterations, elapsed time and boundaries. LP sub-solver consumed all the time with cut generations. It is presented that 227338 multi-linear cut generations totally takes 28.05 sec. It is obvious that in the time distribution, most of the time is used on probing and then on relaxation. At the end of the output, we see the best solution and the error tolerance. It may be confusing that the best solution is not the same as in Table 9 (i.e. $20 \neq 43$). The table is not faulty; simply the result of the solver corresponds to L_0 value (3.5). We put the result into the equation (3.4) and get the L value manually.

```
================================================================================
 Doing local search
 Preprocessing found feasible solution with value 0.160000000000D+02
 Solving bounding LP
 Starting preprocessing LPs
 Done with preprocessing LPs
 Starting multi-start local search
 Preprocessing found feasible solution with value 0.200000000000D+02
 Done with local search
================================================================================
 We have space for 35375 nodes in the tree (in 96 MB memory)
================================================================================
  Iteration    Open nodes       Total time     Lower bound       Upper bound
          1             1        000:00:01    0.200000D+02      0.155000D+03
          1             1        000:00:01    0.200000D+02      0.260000D+02
        44+            22        000:00:31    0.200000D+02      0.250303D+02
        75+            32        000:01:01    0.200000D+02      0.247469D+02
       119+            44        000:01:31    0.200000D+02      0.244116D+02
       158+            49        000:02:02    0.200000D+02      0.242336D+02
        187            53        000:02:32    0.200000D+02      0.240795D+02
       251+            58        000:03:02    0.200000D+02      0.237143D+02
        342            55        000:03:32    0.200000D+02      0.234091D+02
       509+            35        000:04:02    0.200000D+02      0.227054D+02
        575             0        000:04:09    0.200000D+02      0.222222D+02

 Cleaning up solution and calculating dual

                        *** Normal Completion ***

  LP  subsolver time:        000:03:32,    in seconds:        212.49
  NLP subsolver time:        000:00:00,    in seconds:          0.25
  Cutting time:              000:00:29,    in seconds:         29.39
  All other time:            000:00:06,    in seconds:          6.40

  Total time elapsed:        000:04:09,    in seconds:        248.53
       on parsing:           000:00:00,    in seconds:          0.14
       on preprocessing:     000:00:00,    in seconds:          0.16
       on navigating:        000:00:00,    in seconds:          0.25
       on relaxed:           000:00:46,    in seconds:         45.97
       on local:             000:00:00,    in seconds:          0.23
       on tightening:        000:00:02,    in seconds:          2.26
       on marginals:         000:00:00,    in seconds:          0.02
       on probing:           000:03:19,    in seconds:        199.49

  Total no. of BaR iterations:     575
  Best solution found at node:      -1
  Max. no. of nodes in memory:      60

  Cut generation statistics (number of cuts / CPU sec)
    Bilinear                        0         0.00
    Multilinears               227338        28.05
    Convexity                       0         1.34

 All done
================================================================================

Solution      = 20  best solution found during preprocessing
Best possible = 22.2222222222
Absolute gap  = 2.2222222222  optca = 1E-9
Relative gap  = 0.10000  optcr = 0.1
```

Figure 19: Sample Output of NLP Solver (BARON)

6.6 Comparisons among Algorithms

In our second experiment, we have run each algorithm (GA, SA, MBH1-4, and Opt-MBH) 10 times on 18 datasets. In these executions, the variables used in algorithms and their values are as follows:

GA:

Iteration Count: 50

Population Size: 500

Elitist Selection: 5%

Roulette-Wheel Selection: 90%

Random Generation: 5%

Uniform Crossover Rate: 0,6

Mutation Rate for Each Chromosome: 0,0001

SA:

Alpha: 0,99999

Temperature: 400,0

Epsilon: 0,001

MBHs:

R (Randomly Restart Number): 25

Total time of executions was longer than 5 days. Average results have been shown on 8(K=2..9) x 3(Results, Time To Find Best Result, Total Time) = 24 different tables. Rows of the tables correspond to datasets from 1^{st} to 18^{th}. Since just to show the results would be incomplete, how long it took to get the results and the total time of executions have been presented as well.

LP results have been displayed in the 1st columns of the tables. The result obtained from LP solution (**P-N**) is used for an upper bound in comparisons. Generally, as expected, all results are lower than LP values. Apart from that, when the value of K was 2 to 9 and dataset was 3rd or 4th, it is clear to see that the results of other algorithms have reached to LP's.

6.6.1 Comparison between Generic Algorithms

As the complexity of the GA is related with the input size, "total execution time" and "time to find the best solution" increase when the input size increases. From AxA sized datasets to 2*Ax2*A sized datasets running times doubled. Although input sizes are the same, GA run slower when the graph is dense and the values are arbitrary. On the other hand, for SA, input size and the running time are not entirely related. In general, SA acts independently of K, input size, density or the edge weights. When the input size is so large, like in the 17th dataset, we could detect the differences between time values.

For small datasets and for 17th datasets after $K >= 3$, it is obvious that GA is better than SA. However, when the input size is large enough, SA performs better. Additionally, on 18th dataset, SA is better than or the same as GA. Values of the below figures collected from tables to depict the mentioned issues in this section.

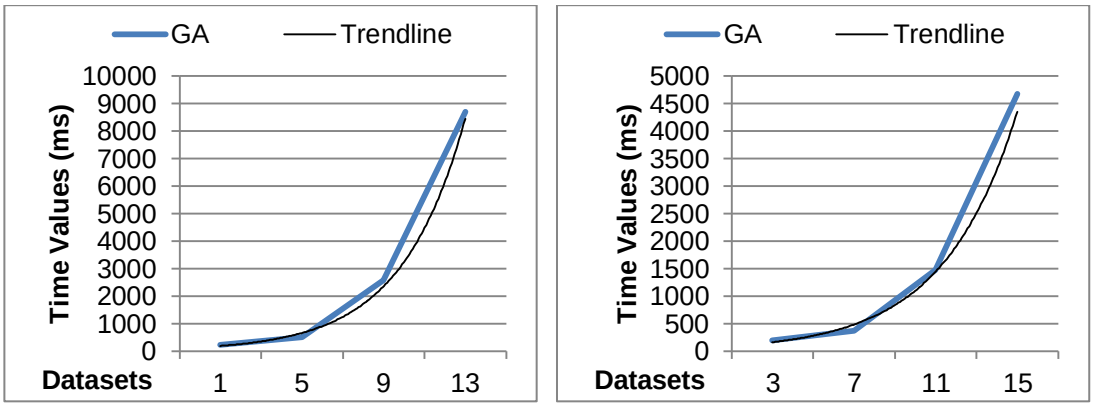


Figure 20: GA Timelines (K=2, Dense-Sparse Datasets)

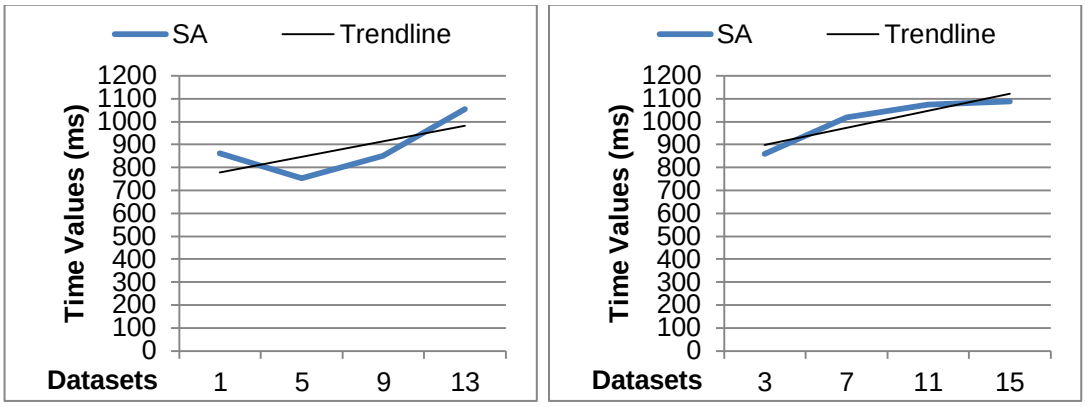


Figure 21: SA Timelines (K=2, Dense-Sparse Datasets)

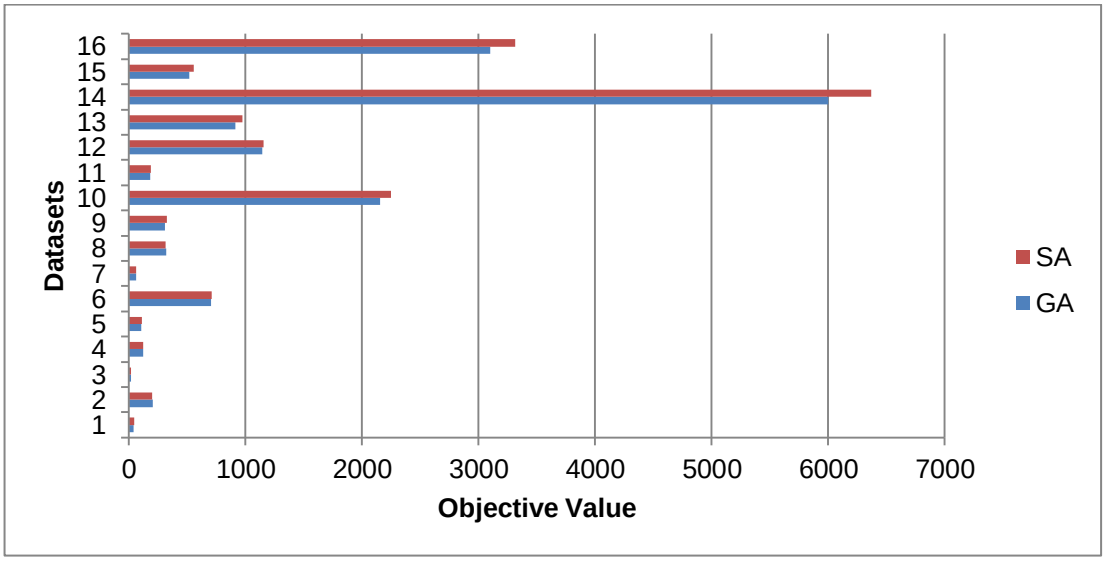


Figure 22: GA vs. SA (Datasets from 1 to 16, K=4)

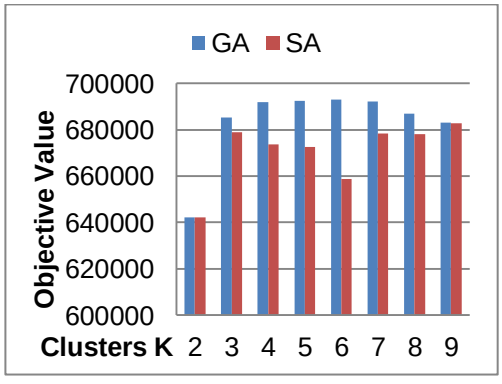


Figure 23: GA vs. SA (17th Dataset)

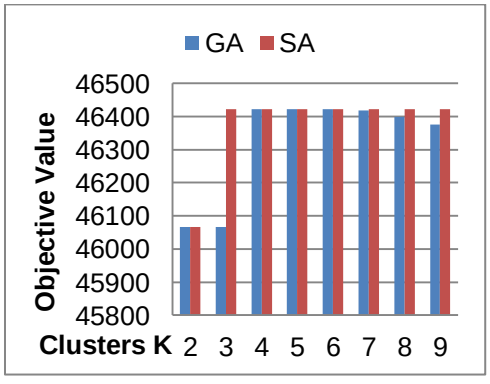


Figure 24: GA vs. SA (18th Dataset)

6.6.2 Comparison between SA and MBH1

Generally, MBH1 performed better than SA. Tables show that for 80x80 sized datasets and K>=3, all the SA results are better. However, important point here is that the elapsed time to find the result is almost ten times more in SA solution. It is not included within experiments for 80x80 sized datasets, but as it can be clearly seen from the results of 10x10, 20x20 and 40x40 sized datasets, if we increase the total running time of the MBH1 algorithm, MBH1 is expected to give better or at least the same results with SA. Figure 25 shows the time differences clearly.

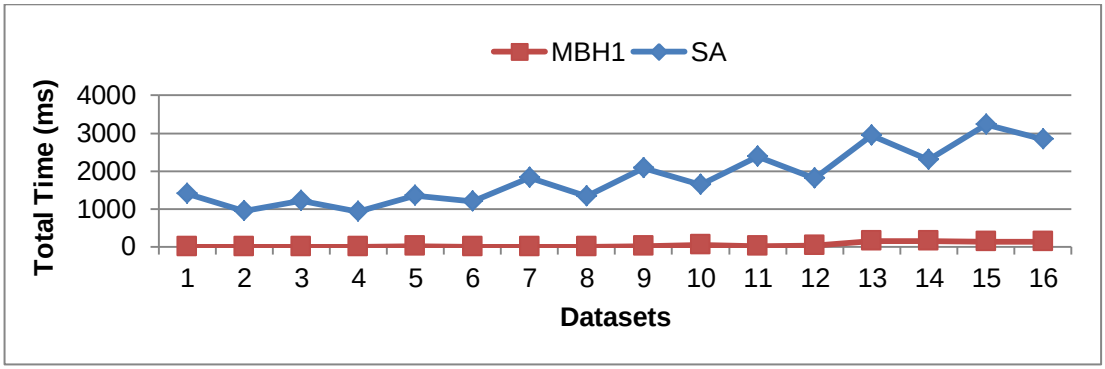


Figure 25: Execution Time Comparison of MBH1 and SA (Datasets 1 to 16, K=9)

6.6.3 Comparison between MBHs

In MBH2-4 we had tried different approaches to the implementation of the algorithm in order to find closer results to the global maximum without striking to local maximums. However experimental results gathered on the tables, showed that our expectations were not satisfied. MBH4 results were closer to the MBH1 results than MBH2 and MBH3. Main reason behind this consequence was that the selection mechanism of MBH4 had been the most similar to MBH1's that has been produced the best results. Besides, in rare cases, less than 3%, (such as K=3 and 16th line), MBH4 produced the best result even better than MBH1, to be pessimistic,

randomness factor might be the reason. To show the difference among MBHs, results of the biggest dataset is used in the chart below.

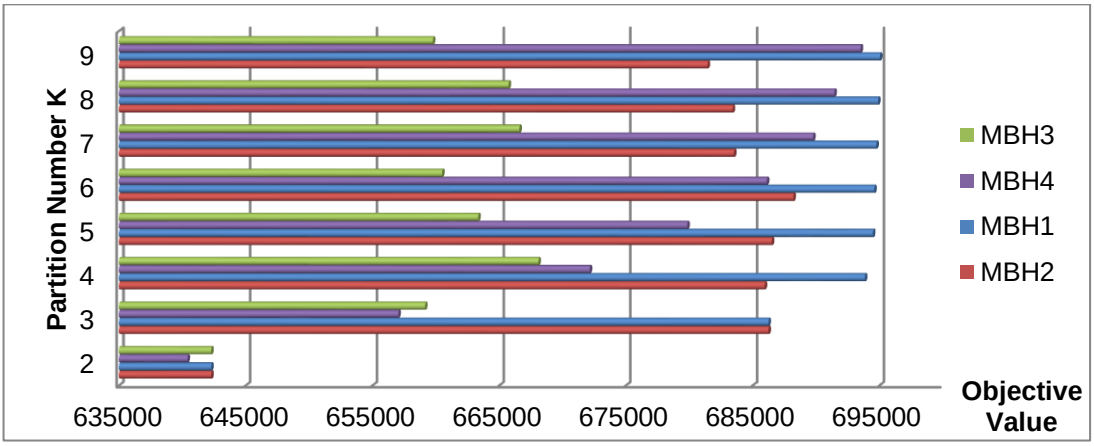
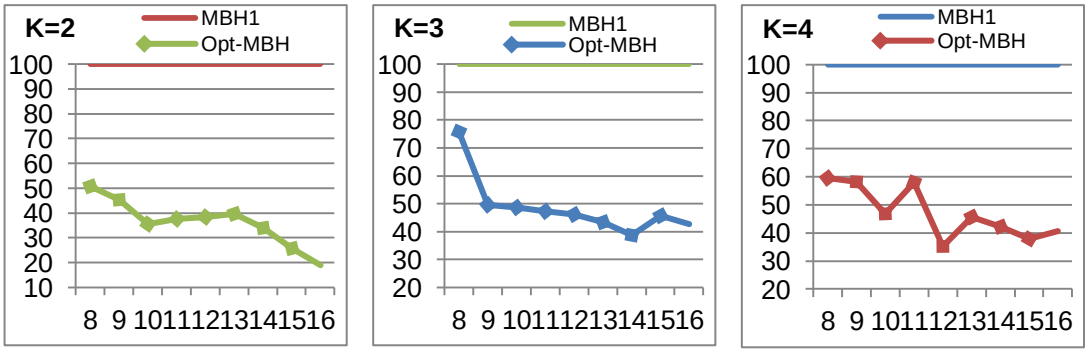


Figure 26: MBH1-4 Comparison (17th Dataset)

6.6.4 Comparison between MBH1 and Opt-MBH

The difference between the best solutions of MBH1 and Opt-MBH was negligible. It was an expected result, since the main parts of the algorithms are the same. The difference is clearly seen in the time values. Especially for bigger datasets, "total execution time" and "time to find the best results" were at least halved in Opt-MBH compared to MBH1. (See Figure 27, Figure 28)



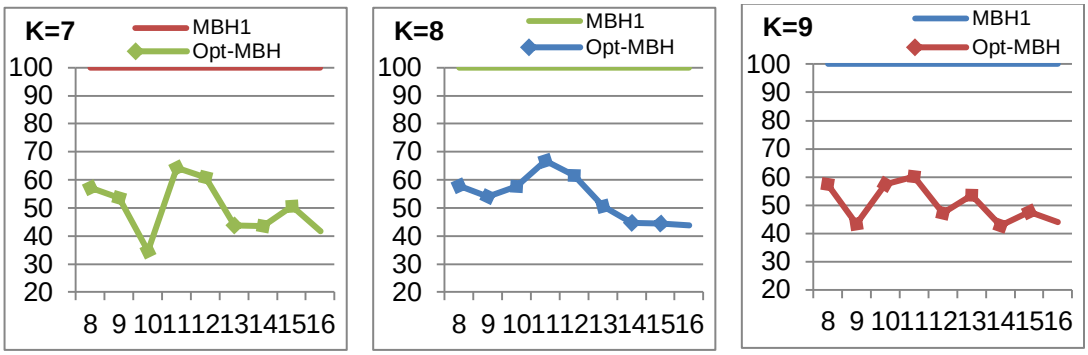


Figure 27: Total Time Comparisons in terms of Percentages (Datasets 8 to 16)

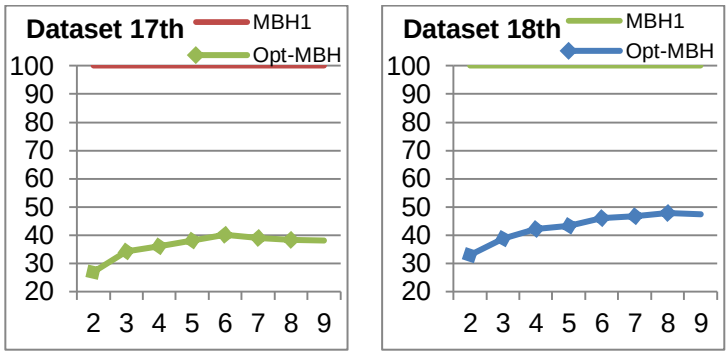


Figure 28: Total Time Comparisons on Real World Datasets (K = 2 to 9)

Detailed outputs of the algorithms are shown in the tables below.

Table 10: Comparison of All Results (K=2)

K=2	LP	GA	SA	MBH1	MBH2	MBH3	MBH4	OptMBH
1	65	35	35	35	35	35	35	35
2	295	177	171	177	177	177	177	177
3	16	14	14	14	14	14	14	14
4	122	120	114	120	120	120	120	120
5	230	96	96	96	95	95	96	96
6	1341	591	578	591	591	591	591	591
7	75	49	49	49	49	49	49	49
8	363	261	253	261	255	259	261	261
9	932	273	275	276	268	262	276	275
10	5162	1652	1678	1683	1628	1657	1684	1684
11	283	143	146	147	139	138	145	147
12	1570	940	932	944	933	927	944	944
13	3826	798	809	815	779	778	814	815
14	21248	4993	5092	5108	4849	4802	5109	5106
15	1159	438	440	444	418	421	444	444
16	6366	2647	2680	2697	2546	2557	2692	2692
17	1127021	642282	642291	642291	642291	642291	640416	642291
18	66416	46066	46066	46066	46066	46066	46066	46066

Table 11: Time to Find the Best Solutions (K=2)

First(ms)	GA	SA	MBH1	MBH2	MBH3	MBH4	OptMBH
1	212	863	0	0	2	0	0
2	231	587	0	2	2	0	2
3	187	860	0	0	0	0	3
4	229	662	0	0	0	0	0
5	502	752	0	3	0	0	0
6	643	591	0	6	6	2	0
7	372	1019	0	2	2	2	0
8	440	728	2	2	0	0	0
9	2573	850	2	24	11	11	2
10	2883	611	5	35	27	9	2
11	1470	1073	6	5	17	8	3
12	1623	774	6	12	11	13	6
13	8707	1056	36	78	94	56	14
14	11179	755	50	91	83	48	14
15	4672	1089	20	64	58	34	13
16	5001	1017	48	118	105	51	5
17	923707	7454	746	2643	4276	69743	437
18	33924	1406	14	51	45	123	11

Table 12: Execution Time of Algorithms (K=2)

Total(ms)	GA	SA	MBH1	MBH2	MBH3	MBH4	OptMBH
1	1281	962	2	3	2	5	0
2	1403	713	3	6	2	3	3
3	1262	961	0	2	3	2	5
4	1150	805	3	3	2	5	0
5	2320	858	6	11	11	6	5
6	2846	749	6	11	6	6	3
7	1526	1134	8	6	8	3	2
8	1657	881	6	9	6	5	3
9	6289	984	17	32	33	20	8
10	7020	842	22	42	42	28	8
11	3268	1206	20	25	28	19	8
12	4523	975	25	34	31	27	10
13	20458	1301	98	195	221	145	39
14	20892	1103	119	212	207	126	41
15	8802	1306	97	142	142	98	25
16	9201	1315	106	150	153	118	20
17	1115816	12321	41116	85184	102133	111279	11023
18	271807	2136	878	1479	1502	3391	289

Table 13: Comparison of All Results (K=3)

K=3	LP	GA	SA	MBH1	MBH2	MBH3	MBH4	OptMBH
1	65	43	43	43	39	40	43	43
2	295	205	199	205	199	205	205	205
3	16	16	16	16	16	16	16	16
4	122	122	120	122	122	122	122	122
5	230	107	108	108	97	97	108	108
6	1341	712	706	719	637	670	719	719
7	75	57	58	59	53	54	59	59
8	363	302	295	307	293	303	302	307
9	932	307	315	316	262	270	314	316
10	5162	2044	2108	2115	1829	1827	2115	2111
11	283	174	179	178	156	152	178	179
12	1570	1105	1116	1119	986	960	1120	1131
13	3826	895	945	936	754	759	941	939
14	21248	5817	6073	6041	4992	5040	6055	6026
15	1159	514	535	531	430	428	527	530
16	6366	3068	3235	3202	2613	2624	3215	3223
17	1127021	685281	678842	686261	686261	659163	657050	686261
18	66416	46066	46422	46422	46356	46421	46422	46422

Table 14: Time to Find the Best Solutions (K=3)

First(ms)	GA	SA	MBH1	MBH2	MBH3	MBH4	OptMBH
1	379	1114	0	2	3	0	0
2	309	710	0	2	0	0	0
3	239	1025	0	0	2	0	2
4	231	780	2	0	2	0	2
5	1006	934	2	2	5	2	9
6	1386	788	6	5	3	0	0
7	668	1309	2	2	0	0	2
8	699	977	2	2	2	5	2
9	2977	1260	3	9	9	16	13
10	3728	869	9	3	6	8	10
11	1966	1481	6	9	6	11	8
12	2336	1027	14	5	9	17	6
13	14140	1557	55	30	47	75	31
14	16875	1178	64	42	34	62	32
15	6254	1685	39	28	24	58	33
16	6134	1482	80	22	24	105	40
17	1071273	10518	4855	24991	38005	137484	1407
18	33924	2321	16	661	415	146	15

Table 15: Execution Time of Algorithms (K=3)

Total(ms)	GA	SA	MBH1	MBH2	MBH3	MBH4	OptMBH
1	1249	1229	2	5	3	3	6
2	1316	853	0	2	0	3	5
3	1094	1131	0	3	2	5	2
4	1069	1022	3	3	3	0	24
5	2225	1048	8	2	5	8	14
6	2814	989	8	5	3	13	2
7	1546	1449	5	5	3	8	3
8	1724	1142	6	5	5	8	5
9	6185	1452	28	19	17	28	14
10	6922	1203	30	16	16	34	14
11	3295	1647	27	17	16	30	13
12	4116	1317	30	15	16	30	14
13	20132	1863	133	76	80	175	58
14	20171	1691	148	69	62	168	58
15	8574	1972	129	62	60	129	59
16	8917	1952	128	61	64	144	55
17	1105766	14602	57415	71857	58461	249778	19580
18	271807	3451	1033	1025	959	3412	401

Table 16: Comparison of All Results (K=4)

K=4	LP	GA	SA	MBH1	MBH2	MBH3	MBH4	OptMBH
1	65	43	43	43	43	43	43	43
2	295	205	200	205	201	199	205	205
3	16	16	16	16	16	16	16	16
4	122	122	121	122	122	122	122	122
5	230	108	110	110	98	98	110	110
6	1341	704	712	723	681	646	722	723
7	75	60	61	61	55	54	61	61
8	363	320	315	325	305	311	321	322
9	932	310	327	327	277	269	326	327
10	5162	2158	2249	2265	1898	1936	2270	2264
11	283	181	187	187	163	162	187	186
12	1570	1146	1157	1163	1041	1036	1165	1162
13	3826	915	977	965	774	781	966	963
14	21248	6005	6373	6305	5273	5298	6321	6326
15	1159	520	557	552	456	449	549	551
16	6366	3100	3315	3276	2734	2779	3271	3277
17	1127021	691752	673602	693867	685959	668109	672162	693867
18	66416	46422	46422	46422	46396	46401	46422	46422

Table 17: Time to Find the Best Solutions (K=4)

First(ms)	GA	SA	MBH1	MBH2	MBH3	MBH4	OptMBH
1	493	1058	0	2	2	2	0
2	410	833	0	3	8	0	2
3	279	1019	2	0	2	3	2
4	264	824	0	3	0	0	0
5	1253	1047	2	6	0	3	2
6	1176	924	3	6	5	5	2
7	902	1407	5	5	2	0	0
8	927	1059	0	5	3	6	2
9	3640	1438	15	8	9	14	6
10	4933	1022	14	11	9	28	9
11	2164	1619	9	3	5	11	3
12	2499	1184	17	5	11	20	5
13	13210	1803	87	42	55	105	41
14	16823	1388	102	28	31	116	36
15	7072	1981	72	33	37	83	42
16	8145	1758	65	25	44	80	36
17	1091079	12056	12513	27648	30336	304839	5828
18	76771	2926	20	732	582	181	19

Table 18: Execution Time of Algorithms (K=4)

Total(ms)	GA	SA	MBH1	MBH2	MBH3	MBH4	OptMBH
1	1240	1145	3	3	3	6	3
2	1285	999	3	5	9	2	3
3	1080	1157	5	2	5	5	2
4	1082	1112	0	6	2	8	3
5	2186	1173	11	9	3	9	5
6	2657	1111	8	9	12	15	8
7	1549	1546	8	8	5	8	5
8	1669	1268	8	5	5	8	5
9	6061	1656	33	17	25	36	19
10	6476	1381	33	22	20	44	15
11	3193	1816	30	17	17	33	17
12	3961	1504	36	19	19	36	13
13	19481	2150	161	87	87	220	74
14	20024	1995	181	75	75	198	77
15	8555	2293	148	69	68	151	56
16	8803	2295	150	70	69	173	61
17	1099614	16558	72328	60705	71178	479881	26027
18	270077	4084	1181	1222	1109	3273	496

Table 19: Comparison of All Results (K=5)

K=5	LP	GA	SA	MBH1	MBH2	MBH3	MBH4	OptMBH
1	65	42	43	43	39	41	43	43
2	295	205	201	205	200	201	205	205
3	16	16	16	16	16	16	16	16
4	122	122	121	122	122	122	122	122
5	230	107	111	112	100	97	112	112
6	1341	707	712	723	660	658	723	723
7	75	61	62	63	56	55	63	63
8	363	321	315	325	311	309	325	325
9	932	311	326	325	271	275	325	326
10	5162	2178	2283	2274	1988	1940	2275	2268
11	283	182	190	190	164	164	189	190
12	1570	1155	1173	1172	1042	1027	1168	1172
13	3826	915	982	967	789	773	968	967
14	21248	5991	6437	6373	5288	5315	6369	6374
15	1159	521	562	555	453	454	554	553
16	6366	3135	3354	3284	2768	2775	3281	3274
17	1127021	692469	672649	694499	686524	663357	679852	694419
18	66416	46422	46422	46422	46398	46402	46422	46422

Table 20: Time to Find the Best Solutions (K=5)

First(ms)	GA	SA	MBH1	MBH2	MBH3	MBH4	OptMBH
1	602	1114	0	2	0	0	0
2	499	852	0	2	2	2	0
3	275	1119	0	2	0	0	2
4	271	838	2	0	0	0	0
5	1296	1103	11	6	5	2	2
6	1359	949	6	0	0	0	0
7	917	1484	6	6	2	2	2
8	880	1059	5	2	3	5	0
9	4143	1560	27	10	3	31	12
10	5664	1201	22	13	15	31	11
11	2426	1757	17	11	6	11	8
12	2499	1415	22	11	10	12	6
13	16900	2037	86	45	45	141	56
14	18152	1571	134	39	45	100	42
15	6808	2265	92	22	23	88	47
16	7926	1964	100	45	30	120	51
17	1076501	13502	30397	42452	28036	856056	12237
18	87671	3276	23	599	546	165	18

Table 21: Execution Time of Algorithms (K=5)

Total(ms)	GA	SA	MBH1	MBH2	MBH3	MBH4	OptMBH
1	1269	1223	5	3	2	3	3
2	1312	1031	0	5	2	5	2
3	1039	1272	0	5	0	2	5
4	1095	1137	3	3	0	0	0
5	2167	1254	11	13	9	9	8
6	2618	1167	11	6	6	14	9
7	1532	1636	8	8	5	6	5
8	1702	1268	8	6	8	10	8
9	5998	1791	42	21	20	44	25
10	6539	1577	44	25	22	48	17
11	3240	2012	31	19	20	38	17
12	3864	1752	34	17	20	41	22
13	19146	2417	184	91	107	247	87
14	19867	2259	209	83	83	225	87
15	8398	2554	172	70	76	178	91
16	8847	2491	167	80	72	183	72
17	1102302	18130	79756	67558	83761	1321474	30342
18	269723	4832	1313	1398	1253	3092	568

Table 22: Comparison of All Results (K=6)

K=6	LP	GA	SA	MBH1	MBH2	MBH3	MBH4	OptMBH
1	65	43	43	43	43	41	43	43
2	295	205	200	205	199	201	205	205
3	16	16	16	16	16	16	16	16
4	122	122	120	122	122	122	122	122
5	230	108	111	112	100	103	112	112
6	1341	703	710	723	643	665	723	723
7	75	61	62	63	56	58	63	63
8	363	317	317	325	311	308	325	325
9	932	302	326	325	265	269	326	326
10	5162	2199	2282	2271	1990	2014	2275	2263
11	283	182	190	191	169	170	189	190
12	1570	1158	1181	1179	1044	1047	1177	1176
13	3826	912	982	970	770	774	962	966
14	21248	5950	6460	6386	5318	5307	6362	6338
15	1159	519	559	555	454	455	553	552
16	6366	3144	3348	3287	2756	2739	3290	3264
17	1127021	692849	658789	694609	688229	660508	686128	694507
18	66416	46422	46422	46422	46387	46421	46422	46422

Table 23: Time to Find the Best Solutions (K=6)

First(ms)	GA	SA	MBH1	MBH2	MBH3	MBH4	OptMBH
1	633	1083	2	2	0	2	6
2	454	885	3	0	2	0	3
3	298	1106	2	0	3	2	2
4	298	914	0	0	0	3	2
5	1323	1195	2	8	5	0	0
6	1493	967	3	5	3	2	2
7	986	1415	2	0	2	0	0
8	981	1148	0	3	5	3	5
9	4473	1763	22	9	6	26	6
10	5485	1318	23	13	19	33	13
11	2265	1866	23	3	14	16	6
12	2933	1513	24	8	5	16	9
13	16558	2321	98	36	72	139	49
14	18739	1779	117	56	61	128	59
15	7356	2490	120	48	50	98	33
16	8067	2192	100	55	34	139	34
17	1083564	14901	37280	58204	65923	1683597	15514
18	96079	3427	22	676	651	130	27

Table 24: Execution Time of Algorithms (K=6)

Total(ms)	GA	SA	MBH1	MBH2	MBH3	MBH4	OptMBH
1	1226	1215	5	2	3	2	8
2	1248	1075	5	6	2	2	5
3	1048	1231	5	2	3	3	3
4	1097	1175	3	2	3	8	3
5	2147	1354	9	9	11	14	11
6	2554	1253	11	8	6	14	5
7	1532	1637	5	6	5	5	6
8	1710	1415	6	5	5	11	6
9	6040	2019	48	26	24	47	25
10	6594	1715	45	22	23	50	23
11	3140	2085	41	22	22	39	27
12	3855	1844	38	20	22	45	27
13	19290	2775	214	111	124	295	106
14	19637	2527	257	104	102	272	117
15	8571	2872	209	89	92	222	94
16	8950	2779	200	90	90	222	69
17	1106928	19804	101092	82610	107366	2609329	40564
18	269928	5291	1523	1568	1485	3473	701

Table 25: Comparison of All Results (K=7)

K=7	LP	GA	SA	MBH1	MBH2	MBH3	MBH4	OptMBH
1	65	42	43	43	41	41	43	43
2	295	203	202	205	200	201	205	205
3	16	16	16	16	16	16	16	16
4	122	122	120	122	122	122	122	122
5	230	107	112	112	97	102	112	112
6	1341	708	717	723	658	645	723	723
7	75	61	62	63	56	56	63	63
8	363	319	318	325	300	294	325	325
9	932	307	325	326	269	266	324	326
10	5162	2204	2286	2272	1992	1954	2266	2271
11	283	181	190	189	163	165	190	190
12	1570	1148	1191	1175	1068	1042	1165	1182
13	3826	905	981	969	755	757	968	969
14	21248	5929	6447	6384	5237	5221	6376	6367
15	1159	513	559	556	452	448	553	553
16	6366	3108	3339	3294	2777	2742	3280	3239
17	1127021	692093	678232	694770	683541	666606	689780	694814
18	66416	46418	46422	46422	46389	46401	46422	46422

Table 26: Time to Find the Best Solutions (K=7)

First(ms)	GA	SA	MBH1	MBH2	MBH3	MBH4	OptMBH
1	655	1144	2	3	0	0	0
2	546	911	0	3	0	9	3
3	310	1101	3	5	0	0	0
4	315	948	0	3	3	0	0
5	1320	1243	2	6	5	3	3
6	1524	1030	3	8	5	0	3
7	927	1679	2	6	5	2	0
8	839	1170	2	6	5	2	2
9	5248	1799	36	14	19	33	17
10	5529	1482	45	14	2	22	8
11	2435	1983	17	2	9	11	8
12	2736	1671	20	14	11	30	6
13	16653	2599	153	44	59	122	47
14	19541	1955	150	52	41	137	67
15	7313	2855	94	50	34	114	59
16	8164	2370	109	53	55	147	70
17	1080305	15692	41396	38439	48663	2972624	30076
18	182518	3966	37	866	755	181	30

Table 27: Execution Time of Algorithms (K=7)

Total(ms)	GA	SA	MBH1	MBH2	MBH3	MBH4	OptMBH
1	1223	1278	5	3	2	2	2
2	1259	1104	2	5	3	13	8
3	1047	1243	6	5	2	0	3
4	1083	1248	5	3	3	0	2
5	2175	1409	16	13	11	14	11
6	2540	1307	11	9	11	14	6
7	1541	1880	6	6	10	8	5
8	1666	1479	11	9	8	9	6
9	5959	2092	53	26	31	54	29
10	6447	1872	67	23	28	59	23
11	3162	2257	39	20	28	44	25
12	3928	2065	44	29	24	45	27
13	19369	3030	254	119	131	334	111
14	19920	2788	291	109	112	300	126
15	8366	3231	237	95	100	253	120
16	8992	3097	221	100	97	239	92
17	1090965	21519	118233	92036	121606	5864530	46114
18	269874	5876	1692	1753	1660	3764	789

Table 28: Comparison of All Results (K=8)

K=8	LP	GA	SA	MBH1	MBH2	MBH3	MBH4	OptMBH
1	65	42	43	43	41	41	43	43
2	295	203	201	204	200	201	205	205
3	16	16	16	16	16	16	16	16
4	122	122	121	122	122	122	122	122
5	230	106	110	112	103	96	112	112
6	1341	698	715	723	663	639	723	723
7	75	60	61	63	56	59	62	63
8	363	319	318	325	300	309	325	325
9	932	303	326	325	265	266	324	325
10	5162	2159	2280	2274	1975	1989	2268	2273
11	283	180	189	190	167	163	190	190
12	1570	1144	1187	1175	1081	1043	1173	1180
13	3826	897	981	966	748	743	965	966
14	21248	5931	6452	6351	5199	5195	6339	6342
15	1159	510	559	553	448	450	549	553
16	6366	3104	3351	3284	2722	2745	3275	3255
17	1127021	686809	678076	694906	683431	665756	691455	694767
18	66416	46398	46422	46422	46393	46420	46422	46422

Table 29: Time to Find the Best Solutions (K=8)

First(ms)	GA	SA	MBH1	MBH2	MBH3	MBH4	OptMBH
1	767	1346	2	3	5	2	2
2	640	980	0	2	6	0	0
3	292	1206	2	0	2	3	0
4	334	939	0	0	0	0	3
5	1424	1323	3	3	8	2	5
6	1566	1089	5	8	6	8	3
7	930	1780	0	8	5	3	6
8	1033	1268	2	3	6	5	3
9	4370	1860	25	5	14	27	20
10	5772	1605	42	22	17	50	19
11	2432	2156	13	6	11	12	16
12	3053	1799	14	14	19	17	17
13	18380	2688	133	45	66	209	67
14	18402	2168	173	67	62	117	86
15	8053	3094	86	67	58	93	80
16	8387	2676	165	59	59	147	61
17	1080644	17241	58430	57571	55168	3812741	24239
18	160207	4317	35	1133	584	131	33

Table 30: Execution Time of Algorithms (K=8)

Total(ms)	GA	SA	MBH1	MBH2	MBH3	MBH4	OptMBH
1	1270	1488	6	3	5	6	3
2	1261	1200	5	6	6	2	0
3	1027	1336	5	3	5	5	2
4	1044	1267	3	5	2	3	5
5	2178	1484	15	9	14	17	11
6	2540	1390	14	14	10	17	8
7	1499	1958	9	9	8	9	9
8	1665	1560	11	6	13	11	6
9	6037	2148	58	27	27	58	31
10	6516	2005	59	31	30	66	34
11	3189	2446	47	20	29	45	31
12	3955	2243	45	29	26	52	28
13	19416	3268	271	124	148	367	137
14	19672	3040	307	115	120	339	137
15	8512	3504	253	106	105	246	112
16	9128	3466	239	106	105	256	105
17	1091275	23096	130676	99924	133186	6623392	49945
18	269564	6588	1811	1889	1806	3772	864

Table 31: Comparison of All Results (K=9)

K=9	LP	GA	SA	MBH1	MBH2	MBH3	MBH4	OptMBH
1	65	42	43	43	41	41	43	43
2	295	204	200	205	201	201	205	205
3	16	16	16	16	16	16	16	16
4	122	122	121	122	122	122	122	122
5	230	105	111	112	101	99	112	112
6	1341	691	718	723	653	648	723	723
7	75	61	62	63	57	56	63	63
8	363	320	315	325	301	308	325	325
9	932	300	326	326	269	265	325	324
10	5162	2178	2275	2274	1962	1958	2264	2274
11	283	182	190	188	168	166	190	188
12	1570	1134	1184	1178	1070	1048	1174	1177
13	3826	896	983	973	740	740	969	967
14	21248	5898	6463	6382	5200	5147	6369	6356
15	1159	507	561	551	444	450	551	555
16	6366	3085	3360	3276	2701	2706	3259	3259
17	1127021	682912	682900	695053	681454	659780	693530	694831
18	66416	46376	46422	46422	46385	46377	46422	46422

Table 32: Time to Find the Best Solutions (K=9)

First(ms)	GA	SA	MBH1	MBH2	MBH3	MBH4	OptMBH
1	796	1402	0	3	2	2	3
2	608	941	0	5	2	0	3
3	337	1215	0	3	0	3	2
4	367	932	0	0	0	0	0
5	1479	1357	10	2	9	2	3
6	1797	1201	5	9	6	12	3
7	1021	1821	5	0	5	6	0
8	1017	1342	0	3	5	6	2
9	4582	2075	22	13	11	16	17
10	5438	1643	41	19	20	45	19
11	2427	2390	22	9	11	33	6
12	3287	1811	33	14	14	28	17
13	17056	2944	147	92	67	217	87
14	18411	2298	152	86	70	234	58
15	7850	3235	128	50	55	139	81
16	8777	2847	136	80	47	169	84
17	1079663	17068	61346	45858	71190	3985041	32776
18	191094	4607	39	814	1204	155	34

Table 33: Execution Time of Algorithms (K=9)

Total(ms)	GA	SA	MBH1	MBH2	MBH3	MBH4	OptMBH
1	1264	1524	5	3	2	2	5
2	1243	1198	2	5	2	6	6
3	1042	1366	3	5	0	6	3
4	1090	1295	8	2	2	0	0
5	2198	1552	19	11	12	20	13
6	2587	1535	17	12	9	19	8
7	1520	2042	11	9	5	14	5
8	1686	1668	11	9	9	14	6
9	5956	2315	69	31	33	66	30
10	6401	2133	66	31	31	75	38
11	3159	2702	47	28	31	53	28
12	3849	2288	50	48	33	56	24
13	19453	3537	289	138	159	398	155
14	20263	3309	348	123	125	374	148
15	8590	3697	265	112	116	282	126
16	9343	3906	259	116	115	278	114
17	1090272	23489	143526	106408	142605	8982386	54711
18	269835	7193	2002	2033	1977	4097	948

6.7 Senator Experiment (18th Dataset)

Our Opt-MBH algorithm had clustered 108 senators and 696 bills into 3 clusters. That is, the gain has increased when the cluster size is increased from 2 to 3. After that, there were no increase in the gain, and therefore, the best clustering result has been found as 3. Figure 29 shows the 2-way, and Figure 30 shows the 3-way partitioning of the bipartite graphs, which are plotted in MATLAB. In these figures columns correspond to the bills and the rows correspond to the senators. The colors (green and red) correspond to the votes of senators on the bills (favor or against). Notice that blue lines have been inserted into these figures in order to make clusters more visible.

The US Senate has 2-party system (with 2 independents, mostly inclined to Democrats), with 100 members. During the 2 years of 111th Senate, the numbers of the members of both parties have changed due to different circumstances. Therefore, the total number of senators has also increased to 108. In two clustering, the clusters were roughly representing the party lines. During 111th Senate, the number of Republicans was 39 in its minimum level, and one of the clusters our system has obtained exactly had that many senators. Of course, there are several Senators voting quite independently from their respective parties. However, even in 3-cluster structure, it has been observed that senators were not clustered forming the 3rd group. Only, a small number of bills have been discovered, which are mostly been rejected by the senators of both parties. The structures of 2 and 3 clusters are as follows:

- in 2-way, 39 senators and 257 bills formed one cluster and 69 senators and 439 bills formed the other one,
- in 3-way, again the number of senators were the same, for the first two clusters and the third cluster had 0 senators, however, 7 bills from the first cluster, and 4 bills from the second cluster had been moved into the third one making it with 0 senators and 11 bills.

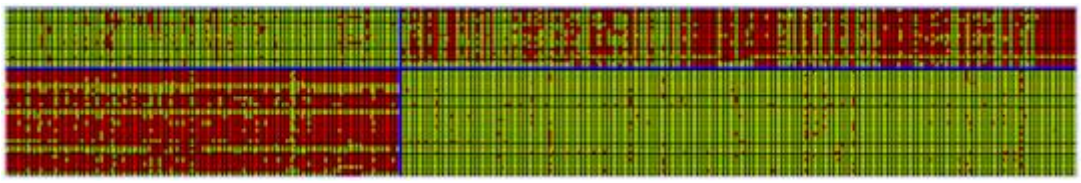


Figure 29: Partitioning of Senators and Bills (K = 2)

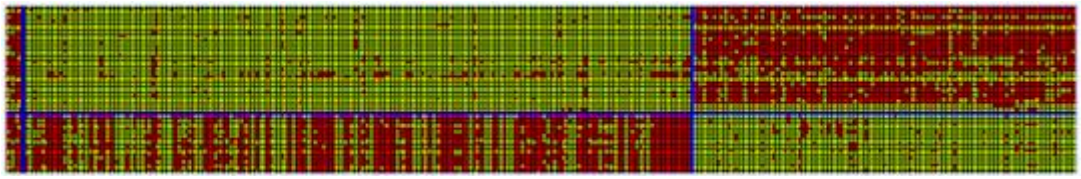


Figure 30: Partitioning of Senators and Bills (K = 3)

6.8 Questionnaire Experiment (17th Dataset)

Table 34: Clusters for Questionnaire Experiment (P: # of persons, Q: # of questions in a cluster)

Clusters	K=2		K=3		K=4		K=5		K=6		K=7		K=8		K=9	
	P	Q	P	Q	P	Q	P	Q	P	Q	P	Q	P	Q	P	Q
1	7458	33	6320	34	5864	33	6083	34	5941	33	6088	34	6080	34	6074	34
2	114	15	1252	0	981	3	637	1	883	3	611	1	572	1	531	1
3	NA	NA	0	14	727	0	447	0	463	0	417	1	423	1	498	1
4	NA	NA	NA	NA	0	12	405	1	203	1	287	0	162	0	198	0
5	NA	NA	NA	NA	NA	NA	0	12	82	2	130	1	126	1	114	1
6	NA	NA	NA	NA	NA	NA	NA	NA	0	9	39	0	123	0	64	0
7	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	0	11	86	1	52	0
8	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	0	10	41	2
9	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	0	9
10	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
RESULT	642291		^ 686261		^ 693867		^ 694499		^ 694551		^ 694759		^ 694843		^ 694851	

In our fourth experiment, we have used the 17th dataset. Since the data size was very large and dimensions were disproportional, we could not print its results in a figure similar to the one that we have done for the Senate experiment. Opt-MBH algorithm had partitioned this weighted bipartite graph into 9 clusters, as the best clustering structure. Our system tries to partition starting with 2 clusters first, and then increases the number of clusters by one. We tried all the cluster sizes from 2 to

9. We have discovered that the objective value increased for each cluster size as it can be seen from Table 34. At each step, we have observed that difference between results was dwindling, like it was converging to some value (Figure 31). Again, as in the Senate experiment, some clusters had only vertices from one of the partitions of the bipartite graphs. The whole experiment took around 4,5 minutes with our test computer of which specifications expressed in section 6.1.

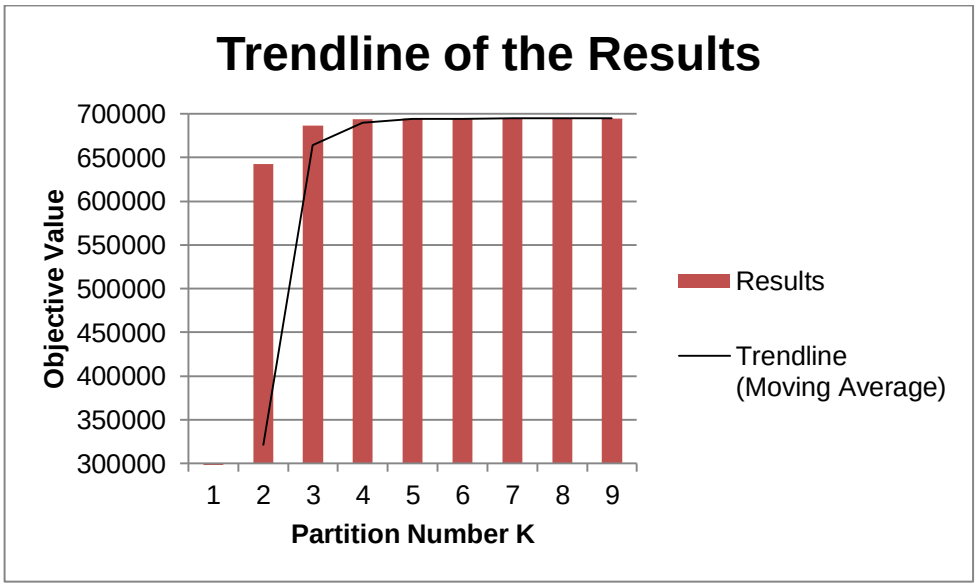


Figure 31: Results of Questionnaire Experiment with Moving Average Trendline

6.9 MBH1 and Opt-MBH on the Same Randomness

In the final experiment, again, MBH1 and Opt-MBH have been compared. However, this time the algorithms have been designed to work with inputs of not only the datasets but also the randomly generated initial solutions. Before the experiment, several random initial solutions had been created by a tool for the given U, V and K values and had been saved on a file. MBH1 and Opt-MBH methods read their initial solutions from the same file instead of generating randomly at that moment of iteration. Therefore, the lines of the below graphs which corresponds to the results of the algorithms exactly follow the same path. "17th dataset with K=9" and "18th dataset with K=8" results are displayed in the figures below. As can be understood

from the minimum values, R=3 has been used in this experiment. Figure 32 contains two graphs which emphasize total execution time of the algorithms and cut points (i.e., vertical lines) in Opt-MBH. Different from Figure 32, in Figure 33 senator dataset has been used with K=8 and approximate saved time is shown. Note that the ratio of total execution time values of MBH1 and Opt-MBH is almost the same as the ratio of pixel values displayed on the chart.

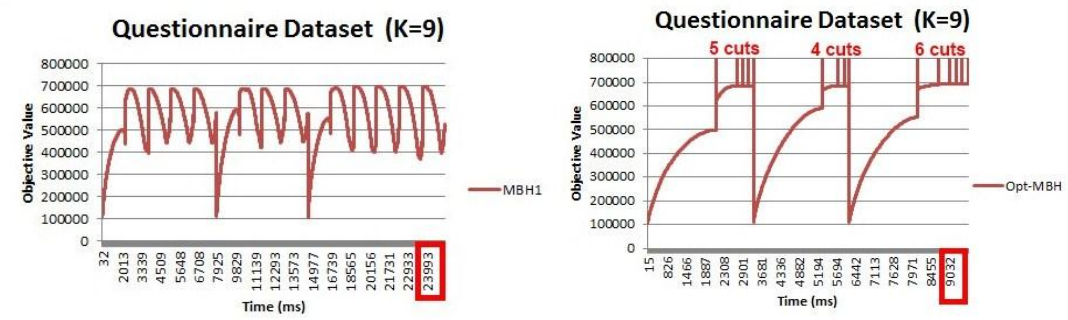


Figure 32: Comparison of Results Using the Same Randomness

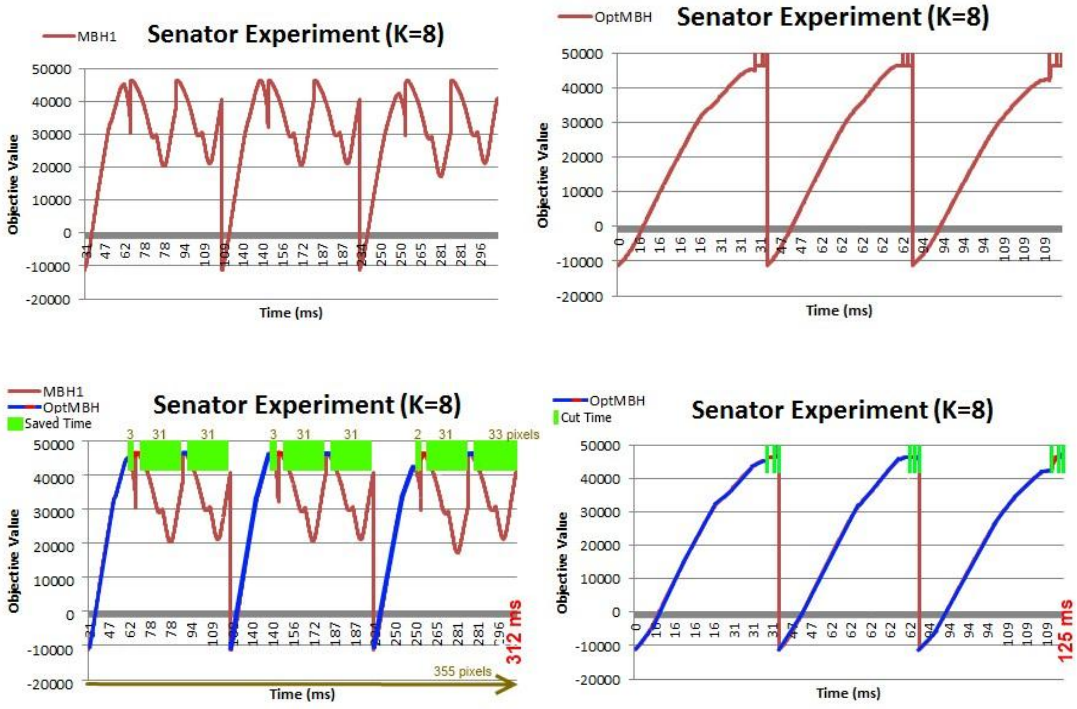


Figure 33: Results on Senator Dataset with K=8

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 Conclusions

This work extends previous work on 2-way clustering of signed bipartite graphs to k-way clustering of signed or weighted bipartite graphs. This problem appears in social networks in many different forms.

In this study, for *k-way partitioning of the signed bipartite graphs* problem, mathematical methods, generic algorithms and various move-based heuristics have been developed. We have shown that our approaches are quite effective through experiments on not only randomly generated data, but also real world data. Size, density and values of the datasets used in experiments were varied in order to generate various conditions.

We can conclude that, optimized move-based heuristic algorithm is at the top among all algorithms both for the best result and execution time. As presented in experimental results, it is quite impressive that bipartite graph data in 7572x48 sized dataset is partitioned into two distinct blocks only in 437 ms after the start of execution. As a product of the study we can say that, as far as we know, the most efficient algorithm to the given problem is produced.

7.2 Future Work

Some future works may be adapting methods used for unsigned partitioning problems for our problem and analyzing the parts emerged after the partitioning process is done.

It would be great to study with some real datasets that have more than 2 natural clusters and work together with social analysts to check the quality of the clusters that have been found.

REFERENCES

[1] Salvatore, J. Bipartite Graphs and Problem Solving. University of Chicago, 2007.

[2] Andrej, M., and Doreian, P. Partitioning signed two-mode networks. Journal of Mathematical Sociology 33, pages 196–221, 2009.

[3] Banerjee, S., Sarkar, K., Gokalp, S., Sen, A., and Davulcu, H. Partitioning Signed Bipartite Graphs for Classification of Individuals and Organizations.

[4] Bansal, N., Blum, A., and Chawla, S. Correlation clustering. In MACHINE LEARNING, pp. 238–247, 2002.

[5] Charikar, M., Guruswami, V., and Wirth, A. Clustering with qualitative information. In Proceedings of the 44th Annual, 2003. IEEE FOCS.

[6] Sen, A., Deng, H., and Guha, S. On a graph partition problem with application to VLSI layout. Inf. Process. Lett. 43(2), 87–94, 1992.

[7] Dhillon, I.S. Co-clustering documents and word using bipartite spectral graph partitioning. In Proceedings of the KDD, 2001. IEEE.

[8] Zaslavsky, T. Frustration vs. clusterability in two-mode signed networks (signed bipartite graphs), 2010.

[9] Zha, H., He, X., Ding, C., Simon, H., and Gu, M. Bipartite graph partitioning and data clustering. In Proceedings of the 10th International Conference on Information and Knowledge Management, pp. 25–32, 2001. ACM.

[10] Fiduccia, C.M., and Mattheyses, R.M. A Linear-Time Heuristic for Improving Network Partitions. In Design Automation, pp. 175-181, 1982.

[11] Kernighan, B.W., and Lin, S. An Efficient Heuristic Procedure for Partitioning Graphs. In Bell System Technical, vol.49, pp. 291-307, 1970.

[12] Bui, T.N., and Moon, B.R. Genetic Algorithm and Graph Partitioning. In Computers, vol.45, no.7, pp. 841-855, 1996. IEEE.

[13] Yang, B., Cheung, W.K., and Liu, J. Community Mining from Signed Social Networks. In Knowledge and Data Engineering, vol.19, no.10, pp.1333-1348, 2007. IEEE.

[14] Doreian, P., Batagelj, V., and Ferligoj, A. Generalized Blockmodeling of Two-Mode Network Data. In Social Networks, vol.26, pp.29-53, 2004.

[15] Danzig, G.B. Reminiscences about the origins of linear programming, in: A. Bachem et al. (eds.) Mathematical Programming—The State of the Art, Bonn 1982, Springer-Verlag, Berlin, 78–86, 1983.

[16] Dantzig, G.B. Programming in a linear structure, Econometrica 17 73–74, 1949.

[17] Wood, M.K. and Dantzig, G.B. Programming of interdependent activities, I, General Discussion, Econometrica 17 193–199, 1949.

[18] Dantzig, G.B. Programming of interdependent activities, II, Mathematical model, Econometrica 17 200–211, 1949.

[19] Dantzig, G.B. Linear Programming and Extensions, Princeton University Press, Princeton, New Jersey, 1963.

[20] Khachiyan, L.G. A polynomial Algorithm in Linear Programming, Doklady Akademiia Nauk SSSR 244:S, p. 1093-1096, translated in Soviet Mathematics Doklady 20:1 (1979), p. 191-194, 1979.

[21] Karmarkar, N. A new polynomial-time algorithm for linear programming, Combinatorica, 4(4): 373-395, 1984.

[22] Maros, I. Computational Techniques of the Simplex Method. *Kluwer Academic Publishers*, Norwell, MA, USA, 2002.

[23] Meggido, N. Pathways to the optimal set in linear programming, Progress in Mathematical Programming Interior-point and related methods, pp. 131–158, 1988.

[24] Holland, J. H. Adaption in Natural and Artificial Systems. University of Michigan Press, 1975.

[25] Goldberg, D. E. Genetic Algorithms in Search, Optimization and Machine Learning. Addison - Wesley, Reading MA, 1989.

[26] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. Optimization by simulated annealing. Science 220, 671-680, 1983.

[27] Dueck, G., and Scheuer, T. Threshold Accepting: A General Purpose Optimization Algorithm Appearing Superior to Simulated Annealing. J. Comp. Phys.90, 161-175, 1990.

[28] Ingber, L. Simulated Annealing: Practice versus Theory. Math. Comput. Modeling 18, 29-57, 1993.

[29] Metropolis, N., Rosenbluth, A. W., Rosenbluth, M., Teller, A. H., and Teller, E. Equation of State Calculations by Fast Computing Machines. J. Chem. Phys. 21, 1087-1092, 1953.

[30] Otten, R. H. J. M. and Van Ginneken, L. P. P. P. the Annealing Algorithm. Boston, MA: Kluwer, 1989.