Optimization Based Heuristics for the Graph Partitioning Problem

by

Ali Hassanzadeh Kalshani

A Thesis Submitted to the
Graduate School of Sciences and Engineering
in Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in

Industrial Engineering

Koç University

July, 2015

Koç University

Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a M.Sc. thesis by

Ali Hassanzadeh Kalshani

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

Assoc. Prof. Dr. Emre Alper Yıldırım (Advisor)

Prof. Dr. Serpil Sayın

Asst. Prof. Dr. Tınaz Ekim Aşıcı

Date: _____

Dedicated to my parents,
my sisters,
and my wife, to whom I owe the leaping delight.

ABSTRACT

The graph partitioning problem is concerned with partitioning the vertices of a graph into a prespecified number of clusters with given cardinalities so as to minimize the total weight of the edges that have two endpoints in different clusters. This problem has numerous applications in various fields, including parallel processing, power grids, social networks, biomedical networks, and VLSI design. Our focus in this study is on heuristic methods that yield good feasible solutions in a reasonable amount of time, especially for large-scale instances.

Our heuristic methods are based on optimal solutions of linear programming relaxations of different integer linear programming formulations of the graph partitioning problem. In particular, these formulations employ binary variables for determining the assignment of each vertex to each cluster. We treat the optimal value of the relaxed version of each of these binary assignment variables as a measure of the likelihood of the assignment of each vertex to each cluster. Using this approach, we propose two different heuristic methods.

In the first method, we aim to compute a good feasible solution of the graph partitioning problem by solving another linear programming problem whose parameters are given by the optimal values of the binary assignment variables in the linear programming relaxation of the original problem.

The second method is a randomized algorithm, in which we treat the optimal value of each binary assignment variable in the original linear programming relaxation as a probability measure. We randomly assign each vertex to each cluster using the corresponding probabilities without taking into consideration the cardinality constraint on each cluster. We then apply a greedy feasibility restoration procedure in an attempt to satisfy the cardinality constraints of clusters.

Since the quality of the final solutions heavily depends on the quality of the linear programming relaxation of the original integer linear programming problem, we consider adding several classes of valid inequalities in an attempt to strengthen the linear programming relaxation.

We perform computational experiments on several classes of randomly generated graphs with different characteristics. In particular, we study the effects of different formulations and different valid inequalities on the quality of the resulting feasible solutions computed by our heuristic methods. We also compare the performance of our heuristic methods with the well-known Kernighan-Lin heuristic method (KL method). Our computational results reveal that our methods usually take considerably less CPU time than the KL method. For very large instances, solving the linear programming relaxation becomes a computational bottleneck. While our heuristic methods, in general, do not seem to outperform the KL method in terms of the solution quality, using the solutions generated by our heuristic methods as initial solutions for the KL method has the potential to lead to further improvements on several instances.

ÖZETÇE

Çizge bölümleme problemi, bir çizgedeki düğümlerin önceden belirlenmiş eleman sayılı kümelere, uçları değişik kümelerde kalan kenarların toplam ağırlığı en küçük olacak şekilde bölünmesi ile ilgilenir. Çizge bölümleme probleminin paralel programlama, elektrik şebekeleri, sosyal ağlar, biyomedikal şebekeler ve çok geniş ölçekli tümleşik devre tasarımları gibi pek çok değişik alanda sayısız uygulamaları bulunmaktadır. Bu çalışmamızda özellikle büyük ölçekli problemler için kabul edilebilir süreler içinde iyi olurlu çözümler veren sezgisel yöntemlere odaklanıyoruz.

Kullandığımız sezgisel yöntemler, çizge bölümleme probleminin farklı tamsayılı doğrusal programlama formülasyonlarının doğrusal programlama gevşetmelerinden elde edilen en iyi çözümleri girdi olarak almaktadır. Bu formülasyonlar, düğümlerin kümelere atanmasını modelleyen ikili karar değişkenlerine sahiptirler. Biz bu ikili atama değişkenlerinin gevşetmelerinden gelen en iyi değerlerin her birini, düğümlerin kümelere atanma uygunluklarını gösteren bir ölçü olarak ele alıyoruz. Bu yaklaşımı kullanarak iki farklı sezgisel yöntem öneriyoruz.

İlk yöntemde, çizge bölümleme probleminin olurlu iyi çözümlerini, asıl problemin doğrusal programlama gevşetmelerindeki ikili atama değişkenlerinin en iyi değerlerini parametre olarak kullanan ikinci bir doğrusal programlama problemini çözerek hesaplamayı amaçlıyoruz.

İkinci yöntemimiz rastlantısal bir algoritmadır. Bu yöntemde asıl problemin doğrusal programlama gevşetmesindeki her ikili atama değişkeninin en iyi değerini bir olasılık değeri olarak ele alıyoruz. Öncelikle, bu olasılık değerlerini kullanarak, kümelerin eleman sayıları kısıtlarını gözönüne almaksızın, düğümleri kümelere rastlantısal olarak atıyoruz. Sonra, açgözlü bir olurluluk onarımı prosedürü uygulayarak kümelerin eleman sayıları kısıtlarının sağlanmasını gerçekleştiriyoruz.

Son çözümün kalitesi, asıl tamsayılı doğrusal programlama probleminin doğrusal programlama gevşetmesinin kalitesine bağlı olduğu için doğrusal programlama gevşetmesini sıkılaştıran birkaç geçerli eşitsizlikler sınıfını eklemeyi değerlendiriyoruz.

Değişik özelliklere sahip rastgele yaratılmış çizge sınıflarında hesaplamalar gerçekleştiriyoruz. Değişik formülasyonların ve değişik geçerli eşitsizliklerin sezgisel yöntemlerimizin verdiği çözümlerin kaliteleri üzerindeki etkilerini özellikle inceliyoruz. Ayrıca, sezgisel yöntemlerimizin performanslarını, literatürde sıklıkla kullanılmış olan Kernighan-Lin sezgisel yöntemleriyle (KL yöntemi) karşılaştırıyoruz. Hesaplamalarımız, geliştirdiğimiz yöntemlerin genellikle KL yönteminden oldukça kısa işlemci zamanı kullandığını gösteriyor. Çok büyük örneklerde, hesaplamalardaki zorluğu yaratan kısım, doğrusal programlama gevşetmelerini çözmek oluyor. Bizim geliştirdiğimiz sezgisel yöntemler, KL yöntemini çözüm kalitesi olarak aşamasa da, sezgisel yöntemlerimizden gelen çözümleri KL yönteminde başlangıç noktası olarak kullanmak, bazı örnekler için daha iyi sonuçlar elde etmemizi sağlıyor.

ACKNOWLEDGMENTS

My truthful gratitudes go to my supervisor, Associate Professor Emre Alper Yıldırım. I am really happy and fortunate to have worked under his supervision. He was always kind and supportive and working with him made the research truly pleasure for me. His guidances not only helped me frame my future career in academia, but also were invaluable for my life. I appreciate the opportunity to be admitted as a graduate student, guiding me during my M.Sc. studies at Koç University, and also helping me to head for the rest of journey in my academic career.

I acknowledge the financial support provided by TUBITAK, grant number 112M870, (The Scientific and Technological Research Council of Turkey) during my graduate studies.

I am grateful to my thesis committee members, Prof. Dr. Serpil Sayın and Asst. Prof. Dr. Tınaz Ekim Aşıcı for reading my thesis and providing valuable feedback.

I thank Seyyed Mojtaba Hosseini specially for being such a great teammate in the course project of the course Network Flows and Optimization, where we implemented Kernighan and Lin's heuristic method. His programming knowledge has always assisted me during the last year.

I also want to express my deepest gratitudes to my family who helped me so far, and believed in me.

Finally, I would like to sincerely thank my wife, for her patience, kindness, support and encouragement. Her unwavering love is undeniably the best thing I have ever felt.

TABLE OF CONTENTS

Chapter 1

INTRODUCTION

In discrete mathematics, a graph consists of *vertices* or *nodes* and *edges*, which connect certain peirs of vertices. In some cases, edges of a graph can have directions, and in that case, we have a *directed graph* and edges are called *arcs*. When there is no direction on the edges of the graph, we have an *undirected graph*.

Since the advent of computers, graphs have been used to solve problems of different types. In fact, we can model a wide variety of real life problems using graphs. As an example, in an optimization problem in which there are cities and roads among cities, we may represent the problem by a graph having vertices and edges as cities and roads, respectively. As another example, the flow of information in an electronic circuit can also be modeled using a graph, where components of the circuit would be the vertices of the graph, and connections among components would be the edges of the graph.

1.1 Preliminaries

One of the important problems in *graph theory*, which has been studied for more than four decades, is *the Graph Partitioning Problem (GPP)*. Given an undirected graph $G = (V, E)$, where V denotes set of vertices and E denotes set of edges of the graph, with weights on the edges, GPP is concerned with partitioning the set of vertices V into a prespecified number of clusters K, such that:

$$V_1 \cup V_2 \cup ... \cup V_K = V;$$

$$V_i \cap V_j = \emptyset \quad \forall i, j : \quad 1 \leq i < j \leq K.$$

$$|V_i| = n_i \quad \forall i = 1, \ldots, N,$$

Assume that number of vertices in cluster k is n_k. In GPP, the objective function is to minimize the total weight of edges which have two endpoints in different clusters. The *cut set* of a partition is the set of all edges that have two endpoints in different clusters. The cardinality of each cluster can be any prespecified number. However, the sum of cardinalities of all clusters should be equal to $|V|$.

The cardinality constraint in this problem ensures that we assign a prespecified number of vertices to each cluster. The number of vertices in each cluster depends on the particular application of GPP. When the number of vertices in each cluster is equal for all clusters, the problem is called *Equipartition Problem*.

1.2 Applications of Graph Partitioning

The graph partitioning problem has a lot of applications in a wide range of fields in science and engineering. One of the important applications of GPP is in *parallel processing*, (see, e.g., [1], [2], and [3]). The underlying graph partitioning problem is the following: When we use a computer to do some computations, all the tasks that are going to be done are represented by the vertices of a graph. These tasks may have relations with each other. Depending on their connections and the effect of each task on the other one, we define edges between these tasks. On the other hand, processors of the computer are represented by clusters. GPP in this application is concerned with assigning tasks to different clusters such that the connection among tasks that are assigned to different clusters is minimized. In fact, when two tasks can be performed independently, we can assign them to separate processors, otherwise they may depend on one another and they should be assigned to a single processor.

Another major application of GPP is in *complex networks*. As an instance of these networks, we can consider *biological networks*. In these types of networks since we deal with a large number of entities, e.g., proteins, we can always construct a graph and

solve the corresponding problems using graph theory. As an example, proteins are represented by the vertices of the graph, and there are different interactions between all kinds of proteins, and connections between them can be represented by the edges of the graph. To simplify the graph and group some proteins that behave similarly to each other, we use graph partitioning (see, e.g., [4]).

GPP has also applications in *Social Networks*. This topic recently has become popular, and task of GPP here is to identify patterns to match people to communities and groups (see, e.g., [5]).

Image processing is also another field that employs graph partitioning. The purpose of graph partitioning in image processing is to partition pixels of an image into a number of clusters, where each cluster corresponds to an object (see, e.g., [6], [7]).

1.3 Contributions

Garey and Johnson [8] showed that GPP is NP-Complete. Therefore, it is unlikely that a polynomial-time algorithm be developed for GPP. In this thesis, we therefore focus on developing heuristic methods for computing good solutions in a reasonable amount of time, especially for large-scale instances.

Our contributions in this thesis are two *optimization based heuristics*. In both of our methods, we take advantage of the Linear Programming (LP) relaxations of two different integer linear programming formulations (ILP). By adding several valid inequalities to these models, we aim to strengthen the LP relaxation. In this case, in addition to using several sets of valid inequalities in the literature, we make use of several lower bounds on the number of edges in the cut set. For instances in which the weight of each edge is equal to one, these lower bounds actually constitute lower bounds on the optimal value of GPP.

In the first algorithm, our goal is to compute a good feasible solution of the graph partitioning problem by solving another linear programming problem whose parameters are given by the optimal values of the binary assignment variables in the LP relaxation of the original problem. The second method is a randomized algorithm,

in which we treat the optimal value of each binary assignment variable in the original LP relaxation as a probability measure. We randomly assign vertices to the clusters using the corresponding probabilities without taking into consideration the cardinality constraint on each cluster. We then apply a greedy feasibility restoration procedure in an attempt to satisfy the cardinality constraints of clusters.

1.4 Outline of the Thesis

In Chapter 2, we review exact formulations and some heuristic approaches in the literature. There are several studies that propose different formulations to GPP using *Quadratic Programming or Mixed Integer Linear Programming*. Some of these studies focus on the *convex hull* of the integer solutions of GPP by defining *valid inequalities* and other logical constraints. On the other hand, some papers focus on developing heuristic algorithms. Kernighan and Lin (KL) [9] introduced one of the most widely used algorithms for the graph partitioning problem. We review the literature of the graph partitioning problem in Chapter 2.

In Chapter 3, we exploit the structure of graphs in order to find lower bounds on the size of the cut set in the graph partitioning problem. Next, we review two exact formulations. We focus on linear programming (LP) relaxations of these formulations. The LP relaxation is obtained by relaxing integer variables so that they can take fractional values. We also discuss several sets of *valid inequalities*. A valid inequality is an additional constraint that cuts off some portion of the feasible region of the LP relaxation without eliminating feasible solutions to the original ILP formulation.

In Chapter 4, we propose two heuristic algorithms. In the first method, using an optimal solution of the LP relaxation, we solve another linear programming formulation (LP) in an attempt to obtain decent feasible solutions. In the second approach, which is a randomized algorithm, we again use fractional values of the LP relaxation to obtain good solutions by randomly assigning vertices to the clusters and then restoring feasibility of this assignment in a greedy manner. Both of our algorithms can compute reasonably good feasible solutions even for large instances in a short amount of time.

However, the quality of the solution returned by this algorithm heavily depends on the solution of the LP-relaxation, which we try to strengthen by adding additional valid inequalities.

In Chapter 5, we perform an extensive computational study to better understand effect of each model, lower bound, valid inequality, and also each algorithm in instances of the problem with different characteristics. Finally, in Chapter 6, we conclude this thesis with some future research directions.

Chapter 2

LITERATURE REVIEW

The Graph Partitioning Problem (GPP) has an extensive literature. In this chapter, we review the literature in three groups. The first group of studies propose optimization formulations or study the convex hull of integer solutions of GPP. The second group of studies develop heuristic approaches for this problem. Finally, in the third group, we review several works that consider variations of the GPP that we study in this thesis. In the following sections, we review literature related to each group.

2.1 Exact Formulations

Conforti et al. [10] study the problem of partitioning the vertices of a graph into two disjoint subsets, and the objective function is to minimize the size of *cut set*. Recall that the cut set for specific solution is the set of edges which have two endpoint in different clusters. They have two assumptions that reduce the problem from a general GPP to a specific type of graph partitioning problem. First, they consider *equipartition*, i.e., all the clusters should contain the same number of vertices. Second, they focus on *Bipartitioning*, i.e., they have only two clusters in the problem. In the second paper by Conforti et al. [11], they study the convex hull of integer solutions of this problem. They introduce several facet-defining inequalities for the convex hull of integer solutions. A facet-defining inequality is a valid inequality that defines a facet of the convex hull of integer feasible solutions.

Another comprehensive study on the Graph Partitioning Problem is a paper by Ferreira et al. [12]. Their problem of interest is more general. They assume weights and capacities on the vertices, and they call it "the node capacitated graph partitioning problem". This paper introduces three different optimization formulations.

They also present new sets of facet-defining inequalities for the node capacitated graph partitioning problem. However, for each instance of GPP, the number of these inequalities is $O(2^N)$, where N is number of vertices of the graph. This is a potential drawback of including these constraints in the model.

Lisser et al. [13] propose a binary quadratic programming formulation of this problem and they study different relaxations of this model in order to obtain good approximate solutions. They consider the general partition case, in which we can have clusters of different sizes. When the number of vertices is not a multiple of the number of clusters, they add dummy vertices to the graph in order to formulate it as an *equipartition* problem.

Delling and Werneck [14] propose useful lower bounds for the minimum graph bisection problem, which is again a special case of general GPP. They identify *edge-based packing bounds*, and comment on how one can compute those lower bounds. Continuing this work, Delling et al. [15] take advantage of previously proposed lower bounds and develop a branch and bound algorithm to solve the graph bisection problem.

An interesting formulation of the GPP is introduced by Sorensen [16]. In this formulation, only one set of binary variables for the edges is defined. Therefore, this formulation has fewer variables in comparison with other formulations. The main disadvantage of this ILP formulation is the longer running time. Two sets of facet-defining inequalities are introduced in [16]. Similar to the valid inequalities in [12], the number of valid inequalities grows exponentially with the number of vertices of the graph.

Among all the previously proposed formulations, we use two models introduced by Pardalos and Feng [17] in our experiments. They consider general GPP and also the problem of partitioning *Bipartite Graphs*. For each problem, they present one binary quadratic programming formulation and three ILP formulations. Our experiments are based on two ILP formulations introduced in this paper.

There are also some efforts to take advantage of semidefinite programming relax-

ation of the original quadratic programming formulation to solve GPP. However, all works in this area are dedicated to the *Graph Bisection Problem*, i.e., we have only two clusters. Sotirov [18] and Karisch et al. [19] are two important studies in this direction.

2.2 Heuristic Approaches

A large number of studies focus on developing various algorithms to approximately solve GPP using different methodologies. One of the widely used algorithms is due to Kernighan and Lin (KL) [18]. First, we discuss this heuristic procedure in detail. Then, we briefly review other relevant algorithms.

Kernighan and Lin [18] introduce an improvement type of a heuristic, in which they start with an arbitrary partition. They start with *bipartitioning* and then they extend it to *K-way partitioning*. In the bipartitioning case, where there are only two clusters, they consider all possible exchanges of pairs of vertices between the two clusters, and check whether those exchanges improve the objective function. They pick the pair which leads to the largest improvement in the objective function. They continue until there is no further improvement by exchanging vertices of the graph. When the number of clusters in GPP is more than two, they consider all possible pairs of clusters and run the bipartition algorithm until there is no further improvement. After each pair, they update the graph and clusters and consider the next pair of clusters. In practice, several passes over all combinations of pairs of clusters are required to reach the final solution.

Two main studies that employ the KL algorithm are [20] and [21]. These two studies propose a three-phase algorithm to partition a *hypergraph*, which is more general than a graph. In a hypergraph, instead of edges, we have *hyperedges* that connect more than two vertices. They reduce the original graph first, using some coarsening schemes. Then, they perform partitioning in smaller graphs, and then they try to refine the original graph by bringing back the eliminated vertices and hyperedges. There are ongoing studies to improve each of these three phases. Safro et al. [22]

also use the idea of coarsening scheme for the GPP. A coarsening-based methodology mainly consists of three phases given by coarsening, partitioning, and refinement. This study is concerned with the coarsening phase and they work on similarity of smaller graphs with the original ones, which is highly important in refining the graph after partitioning.

Another study by Yazici and Aykanat [23] uses previous ideas in [20] to solve the constrained min-cut replication problem. They consider an imbalance ratio for each feasible solution of this problem, which is not allowed in the problem of GPP in this thesis, i.e., sizes of clusters are allowed to be different than a prespecified size by a certain proportion. The inputs are a maximum replication capacity and a K-way hypergraph partition. Note that this initial solution and also final solution can be infeasible in the GPP variant that we study in this thesis. In order to solve this partitioning problem, they mix an ILP formulation with Dulmage-Mendelsohn decomposition [24]. Dulmage-Mendelsohn decomposition is a partition of the vertex set into clusters such that two adjacent vertices should be in the same cluster if and only if they are paired with each other in a perfect matching of the graph.

Goldschmidt and Hochbaum [25] study the K-cut problem which deals with finding a partition of an edge weighted graph into K nonempty clusters such that the total edge weight between clusters is minimum. This problem is NP-complete for an arbitrary K (where K is greater than or equal to 3) and its version involving fixing a vertex in each cluster is NP-hard even for $K = 3$. They present a polynomial algorithm for fixed K. In the problem they work on, there is no restriction on the sizes of the clusters.

Guttmann and Hassin [26] work on the minimum K-cut problem in which the number of clusters is fixed. Another assumption is that the sizes of clusters are not given. They demonstrate that for fixed K, it is possible to obtain in polynomial time an approximation of at most three times the optimal value.

We can find several greedy algorithms for GPP in [27]. The quality of the solution in this method is important, however, the most important fact is to obtain locally optimal solutions in a very short amount of time.

Alpert et al. [28] work on large instances of circuits represented as graphs. They introduce a multilevel tool to partition graphs using a simplified procedure in each step. They continue until the size of the graph is smaller than a prespecified threshold value. Their clustering algorithm is matching-based, and they conduct computational experiments experiments only for the bipartitioning case.

2.3 Other Related Literature

There are also several other papers using different concepts, such as data mining methodologies, game theory, and others. Buluc et al. [29] surveys the literature on GPP.

The Graph Partitioning Problem has a wide range of applications in image processing and content-based image clustering. Qiu [6] and Zha et al. [30] develop some methodologies for partitioning bipartite graphs. In the study by Li et al. [31], superpixels, which are segments of an image, play an important role for partitioning contents of the image into clusters using some clues. Hopcroft et al. [32] also study bipartite graphs, and they introduce a polynomial-time algorithm for the maximum matching problem in bipartite graphs, which can be turned into GPP using some assumptions and modifications.

After KL, Fiduccia and Mattheyses build upon the ideas of Kernighan and Lin. The paper by Fiduccia and Mattheyses [33] propose a heuristic method to partition a *network*, which is defined to be more general than a graph. A network consists of vertices and nets that connect a number of vertices. A *net* is generalization of an *edge* in graphs. Note that the gain of changing clusters is defined as the difference in the objective function and we are interested in reducing the value of the objective function. In order to improve the starting partition, they want to choose a single vertex from one side by looking at gains of moving all vertices of one side to the other side. This way they change sizes of the clusters frequently but they have a balancing ratio criterion for cardinalities of clusters to control the feasibility of solutions. This balancing criterion ensures that the number of vertices in each cluster should belong to

an interval. They continue moving vertices between clusters until there is no possible improvement.

Partitioning bipartite graphs has also applications in clustering documents and words. Rege et al. [34] and Dhillon [35] are two studies on clustering documents and words, taking advantage of bipartite graph partitioning tools. They treat documents and words as two sets of vertices of a bipartite graph. Each of these two papers uses different methods to partition bipartite graphs formed by documents and words.

Another important study is due to Balas and Padberg [36]. The graph partitioning problem can be turned into the set partitioning problem after small modifications. For instance, suppose that we have some options for partitioning the vertex set into a prespecified number of clusters and our goal is to pick the best one among them. In this paper, the authors discuss other relevant problems and ways these problems can be turned into one another. For instance, they provide mathematical justification that the set partitioning problem can be converted, under certain circumstances, into set covering and set packing problems. There are also other problems in this paper that are related to the set partitioning problem such as node covering, node packing, and also clique covering problems.

Lukes [37] studies GPP when the input graph is in the form of a tree. This assumption simplifies partitioning the graph into a number of clusters. He provides a linear time algorithm in the number of vertices to solve GPP in trees.

Max-flow/Min-cut problem is one the famous network optimization problems and we have the well-known *Ford-Fulkerson* algorithm [38] to solve these problems. By eliminating balancing constraint on the sizes of the clusters in bipartitioning case, the problem becomes similar to a max-flow problem. One of recent studies on the Max-Flow problem is done by Orlin [39] and he introduces an algorithm to solve the Max-Flow problem in $O(mn)$ time, where m and n denote number of edges and vertices of the graph. Suppose, in the bipartition case, that we fix a vertex as the source node, and we solve the minimum $s-t$ cut problem considering all other vertices of the graph as sink node. Then we pick the best bipartition. This way, even though

we cannot control the cardinalities of two clusters, we can partition the graph into two clusters in an efficient way.

In this thesis we use two optimization formulations and consider their LP relaxations. We aim to strengthen the LP relaxations by adding several valid inequalities and lower bounds to the original ILP formulations. We use an optimal solution of the LP relaxation to obtain approximate solutions by our heuristic approaches. Our main goal is to solve the graph partitioning problem in large instances in a reasonable amount of time, approximately.

Chapter 3

THE GRAPH PARTITIONING PROBLEM

In Chapter 1, we provided a brief description of the Graph Partitioning Problem (GPP), followed by the review of the literature for this problem in Chapter 2. In this chapter, we first define the problem in detail. Then, we discuss the mathematical models introduced in the literature. Next, we present valid inequalities and lower bounds on the size of the cut set, in an attempt to strengthen the LP relaxation. The last part of this chapter is concerned with a characterization of the optimal solution of the LP relaxation of on of the ILP formulations.

3.1 Problem Definition

We are given an edge-weighted undirected graph denoted by $G = (V, E)$. V denotes the set of vertices of the graph, $V = \{1, 2, 3, \ldots, N\}$, and E denotes the set of edges of the graph, $E = \{(i, j) |$ there is an edge between vertices i *and* $j\}$. The weights on the edges of the graph can be represented using a matrix C. Each element of this matrix, c_{ij}, is the weight of the edge $(i, j) \in E$. When there is no edge between vertices i and j, we define $c_{ij} = 0$. The entries of this matrix can be any nonnegative number.

The problem of interest is to partition the vertex set of G into K disjoint clusters to minimize the total weight of the edges which have two endpoints in different clusters. Assume that number of vertices in cluster k is n_k. Below are three important characteristics of such clusters:

$$V_1 \cup V_2 \cup ... \cup V_K = V,$$

$$V_i \cap V_j = \emptyset \quad \forall i, j \;\; 1 \leq i < j \leq N,$$

$$|V_i| = n_i \quad \forall i = 1, \ldots, N,$$

where $V_1, V_2, V_3, ..., V_K$ represent the vertices in each of the K clusters.

3.2 Mathematical Model

As discussed in Chapter 2, there are several studies that propose exact solution approaches to the graph partitioning problem. Among them, we use two ILP formulations proposed in [17]. The first one is referred to as the *XZ-formulation* which has only binary variables. The second one, which is referred to as the *XTS formulation* is obtained after linearizing a binary quadratic programming model. To understand the XTS-formulation better, we review the quadratic programming model as well.

3.2.1 XZ Formulation

All parameters and assumptions for this formulation are given in the beginning of this chapter. First we define the decision variables.

$$x_{ik} = \begin{cases} 1 & \text{if vertex } i \text{ belongs to cluster } k, \qquad \forall i = 1, \ldots, N,\ k = 1, \ldots, K \\ 0 & \text{Otherwise.} \end{cases}$$

$$z_{ij} = \begin{cases} 1 & \text{if edge } (i, j) \text{ is cut}, \qquad \forall (i, j) \in E \\ 0 & \text{Otherwise.} \end{cases}$$

Next we present the first integer linear programming formulation.

$$min \sum_{(i,j)\in E} c_{ij} z_{ij} \tag{3.1}$$

(XZ) s.t.

$$z_{ij} \geq x_{ik} - x_{jk}, \qquad \forall (i,j) \in E, k = 1, \ldots, K, \tag{3.2}$$

$$z_{ij} \geq x_{jk} - x_{ik}, \qquad \forall (i,j) \in E, k = 1, \ldots, K, \tag{3.3}$$

$$\sum_{k=1}^{K} x_{ik} = 1, \qquad \forall i = 1, \ldots, N, \tag{3.4}$$

$$\sum_{i=1}^{N} x_{ik} = n_k, \qquad \forall k = 1, \ldots, K, \tag{3.5}$$

$$x_{ik} \in \{0,1\}, \quad \forall i = 1, \ldots, N, \ \ , k = 1, \ldots, K\,, \tag{3.6}$$

$$z_{ij} \in \{0,1\}, \quad \forall (i,j) \in E \tag{3.7}$$

In this model, the objective function (3.1) minimizes the total weight of the edges in the cut set of the partition. The constraints (3.2) and (3.3) establish the relationship between two sets of binary variables x_{ik} and z_{ij}. They ensure that when vertices i and j belong to different clusters, i.e., $|x_{ik} - x_{jk}| = 1$, the edge between them should be in the cut, i.e., $z_{ij} = 1$. By (3.4), each vertex should be assigned to only one cluster, while the constraints (3.5) ensure that the number of vertices in cluster k should be $n_k \quad \forall k = 1, \ldots, K$. Finally, the last two constraints declares the range of decision variables.

3.2.2 X-Quadratic Formulation (XQ)

Fan and Pardalos [17] develop another integer linear programming model. This is based on a binary quadratic programming formulation, which is then linearized. The quadratic programming formulation is as follows: Suppose that L is the Laplacian matrix for our graph, $G = (V, E)$. In graph theory, the Laplacian matrix is a matrix representation of a graph. In face given the Laplacian matrix of a graph, we can construct the graph. One can compute the Laplacian matrix of a graph by subtracting the adjacency matrix of the graph from the diagonal matrix consisting of degrees of vertices. For a graph with weights on the edges, let C denote the matrix of weights on the edges. Then, the Laplacian matrix is given by:

$$L = diag\left(\sum_{j=1}^{N} c_{1j},\ \ldots,\ \sum_{j=1}^{N} c_{Nj}\right) - C,$$

where the term *diag* refers to a *diagonal matrix* in which all entries except the main diagonal are zero. Therefore, the entries of the laplacian matrix are given by:

$$L_{ij} = \begin{cases} \sum_{j=1}^{N} c_{ij} - c_{ij} & \text{if } i = j, \\ -c_{ij} & \text{if } i \neq j. \end{cases}$$

The decision variable $X = (x_{ik})_{N\times K}$ is the same binary variable as in the *XZ-formulation* represented as a matrix. The *trace* operator in the objective function of model below simply sums over all entries of main diagonal of the matrix $(X^T LX)$.

$$min \quad \frac{1}{2} trace(X^T LX)$$

(XQ) s.t.

$$\sum_{k=1}^{K} x_{ik} = 1 \qquad \forall i = 1, \ldots, N,$$

$$\sum_{i=1}^{N} x_{ik} = n_k \qquad \forall k = 1, \ldots, K,$$

$$x_{ik} \in \{0, 1\} \;\; \forall i = 1, \ldots, N; \;\; k = 1, \ldots, K.$$

The objective function in this model minimizes the total weight of the edges in the cut set. The first two constraints are the same as constraints (3.4) and (3.5) in the XZ formulation. The last constraint defines the range of decision variables.

3.2.3 XTS Formulation

We use methods presented in [40] and [41] to linearize the binary quadratic programming formulation (XQ). Let us first define the decision variables of this model.

$$x_{ik} = \begin{cases} 1 & \text{if vertex } i \text{ belongs to cluster } k,, \qquad \forall i = 1, \ldots, N, \;\; k = 1, \ldots, K \\ 0 & \text{Otherwise.} \end{cases}$$

In [40] and [41], two new sets of continuous variables are introduced as follows: For each $i = 1, ..., N$; $k = 1, ..., K$, $(s_{ik})_{N \times K} \geq 0$, and $(t_{ik})_{N \times K} \geq 0$. Therefore the linearized formulation is given as follows:

$$min \;\; \sum_{i=1}^{N} \sum_{k=1}^{K} s_{ik} \tag{3.8}$$

(XTS) s.t.

$$\sum_{k=1}^{K} x_{ik} = 1, \qquad \forall i = 1, \ldots, N, \tag{3.9}$$

$$\sum_{i=1}^{N} x_{ik} = n_k, \qquad \forall k = 1, \ldots, K, \tag{3.10}$$

$$\sum_{j=1}^{N} L_{ij} x_{jk} - t_{ik} - s_{ik} + C' = 0, \qquad \forall i = 1, \ldots, N; \;\; k = 1, \ldots, K, \tag{3.11}$$

$$t_{ik} \leq C' - x_{ik}, \qquad \forall i = 1, \ldots, N; \;\; k = 1, \ldots, K, \tag{3.12}$$

$$x_{ik} \in \{0,1\}, \ \ \forall i = 1,\ldots,N; \ k = 1,\ldots,K, \tag{3.13}$$

$$t_{ik} \geq 0, \ \ \forall i = 1,\ldots,N; \ k = 1,\ldots,K, \tag{3.14}$$

$$s_{ik} \geq 0, \ \ \forall i = 1,\ldots,N; \ k = 1,\ldots,K. \tag{3.15}$$

C' is defined in this model as $C' = max_{i=1}^{N} \sum_{j=1}^{N} c_{ij}$. By *Theorem 2* in [17], the formulations XQ and XTS are equivalent. It can be shown that minimizing the objective function (3.8) is equivalent to minimizing the objective function in XQ. The difference between the two objective functions is given by a constant term. Indeed, for any optimal solution (X^0, S^0, T^0) of XTS we have

$$\sum_{i=1}^{N}\sum_{k=1}^{K} s_{ik} = trace((X^0)^T L(X^0)) + C' \times N$$

Therefore, the objective function in (XTS) also minimizes the total weight of the edges in the cut set of the partition. The constraints (3.9) and (3.10) take care of assignment of vertices to the clusters and the cardinality constraint of the clusters. The constraints (3.11) and (3.12) establish the relationships among all three variables x_{ik}, s_{ik}, and t_{ik}. Finally, we have non negativity and integrality constraints given by (3.13), (3.14), and (3.15).

3.2.4 Comparison of Different Models

As we discussed in Chapter 2, there are a number of other formulations proposed in the literature. Ferreira et al. [12] developed four optimization formulations for a special version of this problem, the so-called node capacitated graph partitioning problem. Sorensen [16] formulated GPP using fewer number of binary variables, but its solution time is considerably longer than previous models with more variables.

Among the formulations proposed in [17], we choose to use these two models since their computational experiments indicate that these two formulations works better than the other formulation presented in their paper.

3.2.5 Symmetry Breaking Constraints (SB)

When we have equal number of vertices in each cluster, there is a symmetry in the final solution, because we can change the labels of the clusters without affecting the objective function value. In order to eliminate this symmetry and reduce the amount of work required to be done by the solver in an optimization problem, we use symmetry breaking constraints. For instance, for GPP this can be achieved by the following approach:

By relabeling clusters if necessary, there is always an optimal solution in which vertex 1 belongs to cluster 1. Similarly there is an optimal solution in which the second vertex belongs to either the first or the second cluster. This way, we can write constraints for the first $K-1$ vertices. Using the same x_{ik} variables as before, SB constraints are as follows:

$x_{11} = 1,$

$x_{21} + x_{22} = 1,$

.

.

.

$\sum_{k=1}^{K-1} x_{(K-1),k} = 1.$

3.3 Valid Inequalities

Several sets of valid inequalities have been introduced for the convex hull of integer solutions of GPP. While some valid inequalities are easy to implement and effective in solving GPP, in the literature, we see some other valid inequalities that are defined for every subset of V. Therefore, the number of such constraints grows exponentially with the size of the problem. In the following paragraphs, we review different valid inequalities that have been introduced in the literature. First, we present the so-called triangle inequalities.

3.3.1 Triangle Inequalities (TI)

In a graph, suppose that there is a triangle, i.e., there are three vertices each of which is connected by an edge to each of the other two vertices. Furthermore, assume that we consider the bipartition case. There are two cases for the edges of this triangle: either none of them is cut or only two edges should be cut. It follows that these four sets of inequalities are valid for bipartitioning problem.

$$z_{ij} + z_{il} + z_{jl} \leq 2$$

$$z_{ij} \leq z_{il} + z_{jl}$$

$$z_{jl} \leq z_{ij} + z_{il}$$

$$z_{il} \leq z_{ij} + z_{jl}$$

Suppose that we have vertices, $i, j, l \in V$, and three edges among those vertices, $(i, j), (i, l), (j, l)$. The first constraint ensures that all three edges cannot be cut simultaneously. The next three constraints imply that if any two edges are not cut, the third one cannot be cut either. Even though these four sets of inequalities are valid only for the bipartitioning case, excluding the first inequality, the other three constraints are feasible for the general K-way partitioning problem.

We therefore consider including the following valid inequalities in our models:

$$z_{il} \leq z_{ij} + z_{jl}, \qquad \forall (i, l) \in E, s.t. \exists j : (i, j) \in E, (j, l) \in E$$

We use TI to denote this inequality in the results in Chapter 5.

3.3.2 Cut Inequalities (CI)

Considering two clusters and two vertices, if each of these two vertices belongs to a different cluster, the edge between them should be cut. Here is the corresponding

valid inequality:

$$z_{ij} \geq x_{ik} + x_{jl} - 1; \qquad \forall (i,j) \in E, \qquad \forall k,l \in K : k \neq l$$

We use CI to denote this inequality in the results in Chapter 5.

3.4 Lower Bounds on the Size of the Cut set

Graphs can be either *connected* or *disconnected*. A connected graph has only one connected component and every vertex of the graph is reachable from every other vertex by a path consisting of edges. If a graph has more than one connected component, we say that the graph is disconnected. We use *connectedness* of graphs to obtain lower bounds on the size of the cut set. Another concept that we use is the degree of each vertex denoted by deg_i, which is given by the number of neighbors of vertex i. In other words, deg_i is number of edges incident at vertex i, $\quad i = 1, \ldots, N$.

3.4.1 First Lower Bound

Let us denote the number of connected components of a graph by T. This lower bound is concerned with the relationship between the number of connected components and the number of clusters. Suppose that we have a connected graph and the number of connected components of the graph is denoted by T.

First, we state several results regarding the relationships between the number of vertices, number of edges, and number of connected components.

Lemma 3.4.1. *In any connected graph $G = (V, E)$, we have:*

$$|E| \geq |V| - 1$$

This result is one of the widely used and famous results in graph theory (see e.g., [42]).

Next, we establish a relation between the number of connected components, number of vertices, and edges of the graph.

Proposition 3.4.2. *In any graph $G = (V, E)$;*

$$T \geq max\{|V| - |E|, 1\},$$

where T denotes the number of connected components of the graph.

Proof. Suppose we have T connected components and let $G_t = (V_t, E_t)$ denote the connected graph corresponding to the connected component t, $\quad t = 1, \ldots, T$. Using Lemma 3.4.1, we have

$$|E_T| \geq |V_t| - 1, \quad t = 1, \ldots, T.$$

Therefore,

$$\sum_{t=1}^{T} |E_T| \geq \sum_{t=1}^{T} |V_t| - T,$$

which implies that

$$T \geq |V| - |E|.$$

Since $T \geq 1$, the result follows.

□

Combining these results, we can obtain the following lower bound on the size of the cut set.

Proposition 3.4.3. *Let K denote the number of clusters in our problem, and we have T connected components in the graph. Then*

$$\sum_{(i,j)\in E} z_{ij} \geq max\{K - T, 0\},$$

where z_{ij} is a binary variable which takes one if the edge (i, j) is cut, and zero otherwise.

Proof. Assume, to the contrary, that the size of the cut set is strictly less than $K - T$. After partitioning the graph, suppose we treat each cluster as a single vertex and if there is at least one edge between each pair of clusters, we put an edge in the new

graph.

We therefore have a new graph $G' = (V', E')$ which has K vertices and $|E'|$ edges. Clearly, $|E| \geq |E'|$. Let us assume that the number of connected components of G' is T'. It is easy to see that the size of the cut set in the original problem is greater than or equal to $|E'|$. By Proposition 3.4.2:

$$T' \geq K - |E'| \geq K - \sum_{(i,j)\in E} z_{ij} > T,$$

which implies that

$$T' > T.$$

Note that, for each component in G', there is at least one connected component in the original graph G. Therefore the number of connected components in G, given by T cannot be less than the number of connected components in G', denoted by T'. This is a contradiction. It follows that:

$$\sum_{(i,j)\in E} z_{ij} \geq max\{K - T, 0\}$$

□

Therefore, $LB_1 = max\{K - T, 0\}$, where K is the number of clusters and T is the number connected components of the graph.

3.4.2 Second Lower Bound

For the second lower bound, we consider each connected component of the graph and obtain a lower bound for the size of the cut set for that specific connected component and then we add them up.

We can use Lemma 3.4.4 to compute second lower bound. Note that M_t is number of vertices in connected component t of the graph.

Lemma 3.4.4. *Suppose that cluster sizes $n_1, \ldots, n_k$ satisfy $n_1 \geq n_2 \geq \ldots \geq n_k$. Let*

$$LB_2(t) = min\left\{ l : \sum_{k=1}^{l} n_k \geq M_t \right\} - 1, \quad t = 1, \ldots, T.$$

Then, the size of the cut set for each connected component t is at least $LB_2(t)$. Therefore,

$$\Longrightarrow LB_2 = \sum_{t=1}^{T} LB(t)$$

is a lower bound on the size of the cut set.

Proof. The reasoning for this lower bound is as follows. For each connected component of a graph, depending on the sizes of the clusters, we can obtain the minimum number of edges that should be cut for that specific connected component. We can use proof by induction here. Suppose that for a specific connected component, the number of vertices of that connected component, i.e., M_t, is greater than or equal to the size of the largest cluster, i.e., n_1. In this case, it follows that we should cut at least one edge from that connected component to do the partitioning. Repeating this procedure for other sizes of clusters in a sorted list, we can prove Lemma 3.4.4. □

3.4.3 Third Lower Bound

Recall that the number of adjacent vertices of each vertex is called the degree of that vertex.

Proposition 3.4.5. *Suppose that cluster sizes $n_1, \ldots, n_k$ satisfy $n_1 \geq n_2 \geq \ldots \geq n_k$. For each vertex i in the graph, we have a lower bound on the number of edges incident at i that are cut.*

$$LB_i = max\{deg_i - n_1 + 1, 0\}.$$

This is true because in the worst case, suppose that the vertex i belongs to first cluster whose size is n_1. This means after doing the partitioning, it can have at most $n_1 - 1$ neighbors in that cluster. On the other hand, it has deg_i neighbors in the original

graph. Therefore, the difference between these two numbers is the lower bound on the number of edges cut, among the edges incident at the vertex i.

For this lower bound, we use the Proposition 3.4.5 iteratively. These bounds are called *Degree-based* bounds. The procedure for computing this lower bound is as follows.

Input: Graph $(G = (V, E))$, Cluster sizes $(n_1, ..., n_K)$

Output: A lower bound on the size of the cut set

1 Compute degrees for all the vertices of the graph;
2 Sort the vertices of the graph in non-increasing order of their degrees;
3 Pick the first vertex in the list and call it vertex i and compute the lower bound for the vertex i using 3.4.5, (LB_i);
4 **if** $LB_i = 0$ **then**
5 | Stop the algorithm and report the summation of $LB_i's$ obtained in the previous steps as the overall lower bound on the size of the cut set, LB_3.
6 **else**
7 | Remove the vertex i and its edges from the graph and go the the step 1.
8 **end**

Algorithm 1: Computing third lower bound

Consider the graph in Figure 3.1 as an example:

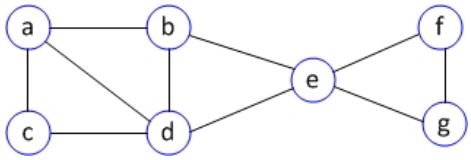

Figure 3.1: An example for the third lower bound

Here is a comparison with respect to different values of n_1:

	Degree-based bound
$n_1 = 4$	1
$n_1 = 2$	$3 + 2 + 1 = 6$

In this table, we compute the lower bound for each vertex using the formula presented in the second step of the procedure above. After sorting the vertices in non-increasing order of their degrees, it is clear that vertices d, e both have the largest degrees. Pick d arbitrarily and we compute the lower bound for that vertex:

When $n_1 = 4$, $LB(d) = max\{4 - 4 + 1, 0\} = 1$.

When $n_1 = 2$, $LB(d) = max\{4 - 2 + 1, 0\} = 3$.

In the first case, where $n_1 = 4$, after removing vertex d and its edges, the remaining vertices has a lower bound of zero. Therefore, the overall lower bound is equal to one for this case.

For the second case, where $n_1 = 2$, after removing vertex d and its edges from the graph, we update the degrees and sort them again. Then, we pick the vertex with largest degree, which is e.

$$LB(e) = max\{3 - 2 + 1, 0\} = 2.$$

Again, we remove the vertex e and its edges from the graph. After updating the degrees again, we pick vertex a since it has the largest degree.

$$LB(a) = max\{2 - 2 + 1, 0\} = 1.$$

After that, there is no positive lower bound for the remaining vertices. We can add these three values to obtain the lower bound on the number of edges that should be cut, which is equal to $3 + 2 + 1 = 6$.

3.4.4 Fourth Lower Bound

This lower bound is obtained using a simple counting argument. The reasoning behind this lower bound on the size of the cut set is as follows. We consider the maximum

possible number of edges that we can have in each cluster after partitioning and then we subtract that number from the cardinality of edge set E. If the result is a positive value, that would be our lower bound.

$$LB_4 = |E| - \sum_{k=1}^{K} \binom{n_k}{2}$$

A proof for this lower bound can be found in [16].

As an example, let us consider the same graph in Figure 3.1 and assume that we want to partition this graph into three clusters of size $\{3, 2, 2\}$. Hence, the corresponding lower bound is

$$LB = 10 - \binom{3}{2} - \binom{2}{2} - \binom{2}{2} = 10 - 5 = 5.$$

Each of these lower bounds is easy to compute and implement. We compute each lower bound in our experiments, and we impose the largest one as a lower bound on the size of the cut set. Note that the lower bounds translate into lower bounds on the optimal value if each edge has a weight of one.

3.5 LP Relaxation of the XZ formulation

In this section, we give a characterization of the optimal solution and the optimal value of the XZ formulation.

Lemma 3.5.1. *The LP relaxation of XZ formulation has an optimal value of zero for any instance of GPP.*

Proof. Let us construct the following feasible solution to the LP relaxation. Assume that we set

$$x_{ik} = \frac{n_k}{N} \quad \forall i = 1, \ldots, N\,; \;\; k = 1, \ldots, K.$$

One can easily verify that using these values we can set all z_{ij} variables to zero since the right hand side of constraints (3.2) and (3.3) in this model are always zero.

$$z_{ij} \geq x_{ik} - x_{jk} = \frac{n_k}{N} - \frac{n_k}{N} = 0, \qquad \forall (i,j) \in E, k = 1, \ldots, K,$$

$$z_{ij} \geq x_{jk} - x_{ik} = \frac{n_k}{N} - \frac{n_k}{N} = 0, \qquad \forall (i,j) \in E, k = 1, \ldots, K,$$

$$\sum_{k=1}^{K} x_{ik} = \sum_{k=1}^{K} \frac{n_k}{N} = \frac{\sum_{k=1}^{K} n_k}{N} = 1 \qquad \forall i = 1, \ldots, N,$$

$$\sum_{i=1}^{N} x_{ik} = \sum_{i=1}^{N} \frac{n_k}{N} = N \times \frac{n_k}{N} = n_k \qquad \forall k = 1, \ldots, K.$$

Therefore, we constructed a feasible solution to the LP relaxation and each z_{ij} variable has a value of zero. Since we have a minimization problem and all the z_{ij} variables are greater than or equal to zero, the objective function cannot have a negative value. Therefore, the above solution is an optimal solution of the LP relaxation, which implies that the optimal solution of the LP relaxation is zero. □

Chapter 4

PROPOSED METHODS: OPTIMIZATION BASED HEURISTICS

Our main goal is to devise effective heuristic algorithms to obtain a good partition of large graphs. In our heuristic algorithms, we use the optimal solution of the LP relaxation, strengthened by adding different valid inequalities, as an input to two algorithms explained below. In both algorithms, we denote by w_{ik} the optimal solution of the LP relaxation. We treat w_{ik} as a measure of the likelihood of vertex i belonging to cluster k.

4.1 Optimization Models

We solve the LP relaxation of four different models, using variations of the models XZ and XTS. *SB* is the abbreviation for Symmetry Breaking constraints. *LB* is the lower bound constraint (i.e., size of the cut set is greater than or equal to this value). We also use two valid inequalities, TI (Triangle Inequalities) and CI (Cut Inequalities). Since solving the LP relaxation is part of our algorithms, we review five different LP models below. Note that SB can only be used for the equipartition case.

4.1.1 Model 1: XZ+SB+LB

For each $i = 1, ..., N;\ k = 1, ..., K$, we define $x_{ik} \in [0, 1]$. Similarly, for each edge $(i, j) \in E$, $z_{ij} \in [0, 1]$.

$$min \sum_{(i,j)\in E} c_{ij} z_{ij}$$

$(XZ - Rel)$ s.t.

$$z_{ij} \geq x_{ik} - x_{jk}, \qquad \forall (i,j) \in E, k = 1, \ldots, K,$$

$$z_{ij} \geq x_{jk} - x_{ik}, \qquad \forall (i,j) \in E, k = 1, \ldots, K,$$

$$\sum_{k=1}^{K} x_{ik} = 1, \qquad \forall i = 1, \ldots, N,$$

$$\sum_{i=1}^{N} x_{ik} = n_k, \qquad \forall k = 1, \ldots, K,$$

$$\sum_{(i,j) \in E} z_{ij} \geq LB,$$

$$\sum_{k=1}^{i} x_{ik} = 1, \qquad \forall i = 1, \ldots, N, \quad \& \quad i < K,$$

$$x_{ik} \geq 0 \;\; \forall i = 1, \ldots, N, \;\; k = 1, \ldots, K,$$

$$z_{ij} \geq 0 \;\; \forall (i,j) \in E.$$

4.1.2 Model 2: XZ+SB+LB+CI

In this model, we only add cut inequalities (CI) to the previous model.

$$s_{ij} \geq x_{ik1} + x_{jk2} - 1; \qquad \forall i, j = 1, ..., N \qquad \forall k_1, k_2 = 1, ..., K : k_1 \neq k_2$$

4.1.3 Model 3: XZ+SB+LB+TI

We add triangle inequalities instead of the cut inequalities.

$$z_{il} \leq z_{ij} + z_{jl}, \qquad \forall (i,l) \in E, s.t. \exists j : (i,j) \in E, (j,l) \in E.$$

4.1.4 Model 4: XZ+SB+LB+CI+TI

In this model, we add both cut inequalities and triangle inequalities.

4.1.5 Model 5: XTS+SB

Again in this model, for each $i = 1, ..., N$; $k = 1, ..., K$, we define $x_{ik} \in [0, 1]$, $s_{ik} \geq 0$, and $t_{ik} \geq 0$.

Here is the LP relaxation of the XTS formulation:

$$min \quad \sum_{i=1}^{N} \sum_{k=1}^{K} s_{ik}$$

$(XTS - Rel)$ s.t.

$$\sum_{k=1}^{K} x_{ik} = 1, \qquad \forall i = 1, \ldots, N,$$

$$\sum_{i=1}^{N} x_{ik} = n_k, \qquad \forall k = 1, \ldots, K,$$

$$\sum_{j=1}^{N} L_{ij} x_{jk} - t_{ik} - s_{ik} + C' = 0, \qquad \forall i = 1, \ldots, N, \ k = 1, \ldots, K,$$

$$t_{ik} \leq C' - x_{ik}, \qquad \forall i = 1, \ldots, N, \ k = 1, \ldots, K,$$

$$\sum_{k=1}^{i} x_{ik} = 1, \qquad \forall i = 1, \ldots, N, \quad \& \quad i < K,$$

$$x_{ik}, t_{ik}, s_{ik}, \geq 0 \ \ \forall i = 1, \ldots, N, \ k = 1, \ldots, K.$$

These five formulations are employed and their optimal solutions are used to compute reasonable solutions of the GPP.

4.2 Two-Stage Algorithm

By solving the LP relaxation of the original ILP formulations, we usually obtain fractional values in an optimal solution. Our goal is to obtain an integer solution. The motivation for having a second stage formulation after solving the LP relaxation is to use those fractional values to extract good feasible solutions.

Using fractional values of the LP relaxation, denoted by w_{ik}, we formulate another

optimization model to extract a feasible solution for the original problem. In fact, w_{ik} is a measure of the likelihood for each vertex i belonging to the cluster k. We call this model, *Second-stage formulation (SS).*

$$x_{ik} = \begin{cases} 1 & \text{if vertex } i \text{ belongs to cluster } k, \qquad \forall i = 1, \ldots, N,\ k = 1, \ldots, K \\ 0 & \text{Otherwise.} \end{cases}$$

$$max \sum_{i=1}^{N} \sum_{k=1}^{K} w_{ik} x_{ik}$$

$$(SS) \qquad \text{s.t.}$$

$$\sum_{k=1}^{K} x_{ik} = 1, \quad \forall i = 1, \ldots, N,$$

$$\sum_{i=1}^{N} x_{ik} = n_k, \quad \forall k = 1, \ldots, K,$$

$$x_{ik} \in \{0, 1\}, \quad \forall i = 1, \ldots, N,\ k = 1, \ldots, K.$$

It can be shown that starting from the optimal solution of the original ILP model as the coefficients of the objective function, we obtain the same optimal solution using this two-stage method. This fact shows that we can use this algorithm to approach the optimal solution of GPP.

This problem is actually an *assignment problem* and since the coefficient matrix is totally unimodular, it can be solved in a very short amount of time, even for large instances, because the LP relaxation always has an integer optimal solution.

Note that any integer optimal solution of this model is feasible for the original problem because the constraints ensure feasibility of the solution for the XZ formulation or the XTS formulation. However, it is not necessarily optimal for the original model, unless each $w_{ik} \in \{0, 1\}, \quad i = 1, \ldots, N; k = 1, \ldots, K$.

This is an optimization based method to obtain an approximate solution. However, the quality of the final solution is related to the coefficients in the objective function. Since we obtain those parameters from the LP relaxation of original ILP model, an

attempt to strengthen the LP relaxation is likely to improve the solution of the second stage model (SS).

As we discussed, since the coefficient matrix is totally unimodular, it takes a very short amount of time to find the optimal solution of the second stage model. Our computational experiences show that, even for large instances of GPP, (where we have thousands of vertices), the second stage model can be solved only in a few seconds, which is a distinctive advantage of this model.

4.3 Randomized Algorithm

Similar to the Two-Stage heuristic, we use fractional values of the solution of the LP relaxation and now treat them as probabilities. Note that, for each vertex i, we have $\sum_{k=1}^{K} w_{ik} = 1$. Therefore, w_{ik} values can be viewed as a probability measure. Then, we use a randomized algorithm to obtain good feasible solutions for the original problem. Algorithm 1 shows the procedure of our randomized algorithm.

Input: Graph $(G = (V, E))$, Cluster sizes $(n_1, ..., n_K)$

Output: A feasible solution for GPP, $(C_1, C_2, ..., C_K)$

1 Solve the LP relaxation;
2 Denote the optimal solution by w;
3 $C_k \longleftarrow \emptyset, \quad \forall k = 1, ..., K$;
4 **for** *vertex* $i = 1$ *to* N **do**
5 $\quad C_k \longleftarrow C_k \cup \{i\}$ with probability w_{ik};
6 **end**
7 $E \longleftarrow \emptyset$; $S \longleftarrow \emptyset$;
8 **for** $k = 1$ *to* K **do**
9 $\quad$ Compute the value r_k for each cluster as such: $r_k = n_k - |C_k|$;
10 $\quad$ **if** $r_k < 0$ **then**
11 $\quad\quad E \longleftarrow E \cup \{k\}$
12 $\quad$ **end**
13 $\quad$ **if** $r_k > 0$ **then**
14 $\quad\quad S \longleftarrow S \cup \{k\}$
15 $\quad$ **end**
16 **end**
17 **while** $E \neq \emptyset$ **do**
18 $\quad$ **for** *vertices (i) in* $\cup_{k \in E} C_k$ **do**
19 $\quad\quad$ Compute maximum improvement (gain) of moving vertex i to all the clusters in set $\cup_{k \in S} C_k$. Denote the best gain of vertex i by g_i.
20 $\quad$ **end**
21 $\quad$ Move the vertex with maximum gain, $\max_{\forall i \in E} g_i$, to the cluster that yields the maximum gain and repeat the steps 8 to 16, to update the information.
22 **end**
23 Report the solution.

Algorithm 2: Randomized Algorithm for GPP

In this algorithm, we first solve the LP relaxation of the original ILP formulation. We use LP relaxations of two different ILP formulations in our study. Note that by

adding valid inequalities, we may end up with different variants of formulations.

After solving the LP relaxation, we denote its solution by w. In most cases, w has fractional values rather than integer values. Then, we treat these fractional values as *probabilities of belonging vertices to clusters*. As an example, assume that we have three clusters, and the fractional values for i^{th} row of w is as follows:

	1	2	3
i	0.3	0.1	0.6

In this small example, the probability that vertex i belongs to the first cluster is 0.3, to the second cluster is 0.1, and probability of being in last cluster is 0.6. Using these probabilities, we obtain an assignment of vertices to the clusters. For each vertex we generate a random number between zero and one. If that number is less than 0.3 we assign it to the first cluster, if it belongs to $(0.3, 0.4]$ we assign it to the second cluster, and finally if it is greater than 0.4 we assign it to the third cluster.

However, cardinality constraints on the clusters may be violated after assigning vertices to clusters. Then, using a procedure of moving vertices between clusters appropriately, we restore feasibility of the solution.

It can be shown that starting from the optimal solution of the original ILP model as the probabilities, we obtain the same optimal solution using this randomized method. This fact shows that we can use this algorithm to approach the optimal solution of GPP.

4.4 KL with Our Heuristics

We can use the solution of our heuristic methods as an initial solution to the KL algorithm. Our experiments show that, in most of the cases, we have further improvement in the solutions, in comparison with solution of the KL algorithm.

Chapter 5

COMPUTATIONAL EXPERIMENTS

We conduct experiments using both ILP models, and also our heuristic approaches. All the optimization formulations are coded in OPL integrated with C# language, using CONCERT technology. We use IBM ILOG CPLEX Optimization Studio 12.5.1 to solve the models. For all the algorithms, we used C# .Net programming language and Microsoft Visual Studio Professional 2012, Version 4.5.51219. We generated a user-friendly graphical user interface for the experiments.

Both algorithms use fractional values of the LP relaxation of different models we presented in Chapter 4. We implemented the KL algorithm in order to compare with our heuristic methods. We first describe the KL algorithm in detail. Then we present the results of our computational experiments.

5.1 Kernighan and Lin's Heuristic Method (KL)

Kernighan and Lin [9] propose an efficient heuristic method for partitioning a graph. Their method is an improvement type of a heuristic, in which, starting from an arbitrary partition, pairwise vertex exchanges are performed between two clusters until no further improvement can be achieved in the objective function.

5.1.1 Bipartitioning

For each vertex in a cluster, Kernighan and Lin define internal and external costs, and then for all possible exchanges of vertices between pairs of clusters, they calculate the gain of exchanging them (i.e., the reduction in the value of objective function). Then their method performs the exchanges and obtains a locally optimal solution for that initial partition.

5.1.2 Extending to K-way Partitioning

Kernighan and Lin extend their approach to obtain a method for solving the K-way partitioning problem. They take all possible pairs of clusters in the problem and run the algorithm for the bipartitioning case on each pair of clusters. Obviously at each pass of the algorithm we have $\binom{K}{2}$ number of pairs to be scanned using the bipartitioning approach.

For each pair of clusters, we perform the procedure described in Section 5.1.1, and once the termination criterion is satisfied for that pair of clusters, we move to the next pair in turn. The order of pairs to be examined is important since checking each pair and changes that we make at each step affect the solution obtained with examining next pairs in order. We may have several passes through all the pairs of clusters. We count the number of consecutive pair-wise scans whit no improvement, and when this number is equal to the number of pairs, i.e., $\binom{K}{2}$, we stop the algorithm.

5.2 Computational Results

We conduct experiments for graphs with different characteristics. For this purpose, we generate random graphs with different number of vertices and densities. We define a parameter, denoted by p, which is a number between zero and one. This parameter corresponds to the density of the graph, which is defined as the ratio of the number of the edges to the maximum number of edges given by $\frac{N(N-1)}{2}$ for a graph with N vertices.

In the randomly generated graphs, each pair of vertices i and j is connected by an edge with probability p.

The smallest graph we generated has 20 vertices and 14 edges, and the largest graph has 20,000 vertices and 6,001,973 edges. For graphs with 20, 50, and 100 vertices, we solve the exact ILP models along with heuristic algorithms. For larger instances, we only apply our heuristic methods.

The rest of this chapter is devoted to results of our computational experiments. Note that we used equipartition with number of clusters K chosen from $\{2, 5, 10\}$ in

all the experiments. First, we review the results of exact formulations. The time limit in our experiments is one hour of CPU time. We solve exact models for instances of 20, 50, and 100 vertices, with different p values. For graphs with 20 vertices, we use p values from the set $\{0.1, 0.25, 0.5, 0.75, 0.9\}$. For graphs with 50 vertices, we use p values in the set $\{0.1, 0.25\}$, and finally, for graphs with 100 vertices, we only consider 0.1 as the p value. Then, we compare these results with different heuristic approaches on the same instances. For larger graphs, we cannot solve exact models due to memory problems. For instances with 200, 400, and, 1000 vertices, we apply our heuristic methods using different models and consider the effect of different valid inequalities.

Note that the randomized algorithm is run a total of 10 times and the best solution is reported in an attempt to reduce the effect of randomness. Since the KL algorithm also uses random initial solutions, we run the KL algorithm for 10 times using 10 randomly selected initial partitions and report the best solution.

5.2.1 Exact Results

In this section, we review the results of the ILP formulations in samples of size, 20, 50, and 100 vertices with different p values. Note that in all tables of this chapter we used some abbreviations illustrated in Table 5.1:

Table 5.1: Notation used in the experiments

Name	**Explanation**
p	p value
K	Number of clusters
XZ: Base	XZ formulation
XZ: LB	XZ formulation+ lower bounds
XZ: SB	XZ formulation+ symmetry breaking
XZ: CI	XZ formulation+ cut inequalities
XZ: TI	XZ formulation+ triangle inequalities
XZ: CI+TI	XZ formulation+ triangle and cut inequalities
XTS: Base	XTS formulation
XTS: SB	XTS formulation+ symmetry breaking
LB	Lower bound on the size of the cut set
KL	Kernighan and Lin's algorithm
RND	Randomized algorithm
SS	Two-stage algorithm
RND-KL	KL using randomized algorithm
SS-KL	KL using two-stage algorithm
M1	LP relaxation of XZ+SB+LB
M2	LP relaxation of XZ+SB+LB+CI
M3	LP relaxation of XZ+SB+LB+TI
M4	LP relaxation of XZ+SB+LB+CI+TI
M5	LP relaxation of XTS+SB

For each pair of N and p, we generate ten instances randomly, and we report the optimal values, gap information, and CPU times taking average over all these ten instances.

For each number of vertices, we have three tables except the last one, the instances with 100 vertices. The first table presents normalized objective function values. For each instance, we scale the best feasible solution obtained to 100, this can be optimal value, or the best solution found within the time limit. Using this normalization, we can better compare the results of different models. *Therefore closer value to 100, has better objective function value.* In the second table, we present optimality gaps for those cases which were not solved to optimality in one hour of CPU time. The number after each gap information shows the number of instances that hit the limit, and the fractional number is the average optimality gap of those instances. The formula for

the optimality gap is given by:

$$Optimality\ \ Gap = \frac{UB - LB}{UB},$$

where UB and LB denote the upper bound and the lower bound on the objective function obtained by the solver, respectively.

In the third table, we report average CPU time over 10 random instances in seconds.

Table 5.2: Normalized Objective Function Values- 20 vertices

p	K	XZ: Base	XZ: LB	XZ: SB	XZ: V1	XZ: V2	XZ: V1+V2	XTS: Base	XTS: SB
0.1	2	100	100	100	100	100	100	100	100
	5	100	100	100	100	100	100	100	100
	10	100	100	100	100	100	100	100	100
0.25	2	100	100	100	100	100	100	100	100
	5	100	100	100	100	100	100	100	100
	10	100	100	100	100	100	100	100	100
0.5	2	100	100	100	100	100	100	100	100
	5	100	100	100	100	100	100	100	100
	10	100	100	100	100	100	100	100	100
0.75	2	100	100	100	100	100	100	100	100
	5	100	100	100	100	100	100	100	100
	10	100	100	100	100	100	100	100	100
0.9	2	100	100	100	100	100	100	100	100
	5	100	100	100	100	100	100	100	100
	10	100	100	100	100	100	100	100	100

Table 5.3: Optimality Gap- 20 vertices

p	K	XZ:Base	XZ: LB	XZ: SB	XZ: V1	XZ: V2	XZ: V1+V2	XTS:Base	XTS: SB
0.1	2	-	-	-	-	-	-	-	-
	5	-	-	-	-	-	-	-	-
	10	-	-	-	-	-	-	-	-
0.25	2	-	-	-	-	-	-	-	-
	5	-	-	-	-	-	-	-	-
	10	-	-	-	0.12 (7)	-	0.03 (1)	-	-
0.5	2	-	-	-	-	-	-	-	-
	5	-	0.001 (1)	0.03 (4)	-	-	0.11 (10)	-	-
	10	0.04 (8)	-	0.1 (10)	0.29 (10)	0.05 (8)	0.30 (10)	0.03 (8)	-
0.75	2	-	-	-	-	-	-	-	-
	5	0.12 (10)	-	0.23 (10)	0.17 (10)	0.19 (10)	0.33 (10)	-	-
	10	0.15 (10)	-	0.18 (10)	0.34 (10)	0.16 (10)	0.42 (10)	0.004 (9)	0.003 (10)
0.9	2	-	-	-	-	-	-	-	-
	5	0.16 (10)	-	0.29 (10)	0.22 (10)	0.26 (10)	0.36 (10)	-	-
	10	0.14 (10)	-	0.21 (10)	0.36 (10)	0.27 (10)	0.55 (10)	0.002 (6)	0.001 (10)

Table 5.4: CPU Times(s)- 20 vertices

p	K	XZ: Base	XZ: LB	XZ: SB	XZ: V1	XZ: V2	XZ: V1+V2	XTS: Base	XTS: SB
0.1	2	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	5	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
	10	**1**	**1**	3	6	**1**	5	3	5
0.25	2	**0**	**0**	**0**	**0**	**0**	**0**	2	1
	5	**7**	14	30	14	**7**	32	41	79
	10	54	**20**	190	2724	55	997	113	97
0.5	2	**0**	2	1	1	1	2	5	3
	5	400	893	2696	1032	787	3600	**194**	233
	10	3340	**2**	3600	3600	3452	3600	3432	2001
0.75	2	3	12	**2**	3	27	115	8	4
	5	3600	111	3600	3600	3600	3600	57	**56**
	10	3600	**2**	3600	3600	3600	3600	3453	3600
0.9	2	6	22	4	6	89	138	**3**	5
	5	3600	**8**	3600	3600	3600	3600	115	200
	10	3600	**2**	3600	3600	3600	3600	2861	3600

In the tables, the boldface entries indicate the ones that have better performance than the others. In the tables representing normalized objective function value, boldface entries represent the models that obtained the best objective function value. In the tables showing CPU times, boldface entries correspond to the model with the shortest CPU time.

From the results of experiments with graphs of 20 vertices, it is clear that when the density of the graph ($p - value$) increases, our lower bounds on the size of the cut set also get better, and in some cases, while other models hit the time limit, the model with lower bounds is solved to optimality within a few seconds in these instances. In addition, we can observe that, in some instances, the XTS formulation works better than the XZ formulation.

For the instances with 50 vertices, and for the values of p greater than or equal to 0.5, we run into memory problems.We therefore limit our experiments in larger sizes of graphs to the smaller p-values.

Table 5.5: Normalized Objective Function Values- 50 vertices

p	K	XZ: Base	XZ: LB	XZ:SB	XZ: V1	XZ: V2	XZ:V1+V2	XTS: Base	XTS:SB
0.1	2	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**
	5	100.37	**100**	102.2727273	100.55	100.35	103.4090909	101.98	104.5454545
	10	102.76	**100.52**	104.8387097	106.01	102.45	106.4516129	101.66	104.0322581
0.25	2	**100**	**100**	**100**	**100**	**100**	**100**	**100**	102.1276596
	5	102.42	102.53	102.8571429	106.07	102.69	107.4285714	**101.73**	100.5714286
	10	104.28	**101.12**	101.8348624	107.43	104.83	108.2568807	101.25	100

Table 5.6: Optimality Gap- 50 vertices

p	K	XZ: Base	XZ: LB	XZ:SB	XZ: V1	XZ: V2	XZ:V1+V2	XTS: Base	XTS:SB
0.1	2	-	-	-	-	-	-	-	-
	5	0.05 (4)	0.04 (4)	0.28 (10)	0.11 (8)	0.06 (6)	0.26 (10)	0.19 (10)	0.2 (10)
	10	0.4 (10)	0.38 (10)	0.56 (10)	0.58 (10)	0.39 (10)	0.56 (10)	0.34 (10)	0.32 (10)
0.25	2	-	-	-	-	-	-	0.17 (10)	0.16 (10)
	5	0.57 (10)	0.56 (10)	0.69 (10)	0.65 (10)	0.58 (10)	0.72 (10)	0.42 (10)	0.39 (10)
	10	0.71 (10)	0.12 (10)	0.77 (10)	0.8 (10)	0.72 (10)	0.92 (10)	0.48 (10)	0.47 (10)

Table 5.7: CPU Times(s)- 50 vertices

p	K	XZ: Base	XZ: LB	XZ:SB	XZ: V1	XZ: V2	XZ:V1+V2	XTS: Base	XTS:SB
0.1	2	**1**	**1**	**1**	**1**	**1**	**1**	52	7
	5	2202	2319	3600	3402	**2155**	3600	3600	3600
	10	**3600**	**3600**	**3600**	**3600**	**3600**	**3600**	**3600**	**3600**
0.25	2	290	268	**89**	285	774	2417	3600	3600
	5	**3600**	**3600**	**3600**	**3600**	**3600**	**3600**	**3600**	**3600**
	10	**3600**	**3600**	**3600**	**3600**	**3600**	**3600**	**3600**	**3600**

When the p value for graphs with 50 vertices increases, solving the model to optimality within the time limit becomes more difficult.

As a remark we can observe that with larger values of p, we have more edges. Therefore we have to define more variables and constraints.

Usually the model with lower bound constraints is the most promising one among all the models.

Table 5.8: Normalized Objective Function Values- 100 vertices

p	K	XZ: Base	XZ: LB	XZ:SB	XZ: V1	XZ: V2	XZ:V1+V2	XTS: Base	XTS:SB
0.10	2.00	**100.49**	100.55	101.49	100.91	100.72	101.88	100.68	101.79
	5.00	105.03	105.67	101.83	111.59	104.75	115.54	101.38	**101.1**
	10.00	111.67	109.56	110.10	119.31	110.07	114.07	101.70	**101.64**

Table 5.9: Optimality Gap- 100 vertices

p	K	XZ: Base	XZ: LB	XZ:SB	XZ: V1	XZ: V2	XZ:V1+V2	XTS: Base	XTS:SB
0.1	2	0.28	0.28	0.26 (10)	0.29	0.29	0.36 (10)	0.33	0.32 (10)
	5	0.8	0.81	0.85 (10)	0.87	0.81	0.88 (10)	0.57	0.58 (10)
	10	0.9	0.81	0.91 (10)	0.95	0.9	0.96 (10)	0.7	0.67 (10)

From the experiments with graphs having 50 and 100 vertices, we can conclude that going beyond these samples, we cannot solve the ILP model to optimality and we always have a positive optimality gap. Also in larger instances, either XZ with lower bounds, or XTS are better than other models in terms of approaching the optimal solution.

5.2.2 Exact vs. Heuristic

In this section, we review results from our heuristic approaches which we ran on the same instances with 20, 50, and 100 vertices. We can compare the objective function values and CPU times.

Comparing objective function values shows that, for large instances, in which we are unable to get an optimal solution within the time limit, heuristic approaches can obtain better solutions. Furthermore, depending on the density of the graph and on the number of clusters, different models and algorithms exhibit different performances. For example, when the number of clusters increases, KL algorithm is faster, but the quality of its solution gets worse.

Table 5.10: Normalized Objective Function Values- 20 vertices

p	K	LB	KL		RND	SS	RND-KL	SS-KL
0.1	2	47.22	126.85	M1	160.19	383.33	116.67	144.44
				M2	145.37	357.41	118.52	120.37
				M3	160.19	344.44	127.78	133.33
				M4	205.13	410.26	205.13	117.00
				M5	137.04	236.11	116.67	122.22
	5	59.93	111.92	M1	139.21	167.90	117.94	116.07
				M2	158.77	188.66	121.36	111.92
				M3	146.88	172.93	117.94	109.45
				M4	137.25	137.25	117.65	117.65
				M5	129.03	152.43	118.18	113.77
	10	98.41	104.04	M1	127.34	156.04	106.55	104.46
				M2	124.25	146.39	108.36	105.85
				M3	127.34	153.72	108.26	107.76
				M4	123.60	168.54	112.36	112.36
				M5	111.96	119.36	103.67	103.45

Table 5.11: Normalized Objective Function Values- 20 vertices

p	K	LB	KL		RND	SS	RND-KL	SS-KL
0.25	2	8.47	100.71	M1	115.05	173.02	104.88	109.72
				M2	115.05	168.99	104.88	106.96
				M3	115.05	174.77	104.88	105.63
				M4	117.30	170.09	105.57	105.57
				M5	121.17	123.54	107.12	107.03
	5	69.51	105.49	M1	122.03	131.27	107.42	105.53
				M2	118.69	134.65	103.73	104.21
				M3	118.65	140.73	105.04	103.01
				M4	112.80	125.00	106.71	103.66
				M5	111.48	114.28	103.67	103.54
	10	99.43	101.05	M1	108.04	116.75	101.59	101.02
				M2	107.44	117.82	101.64	100.58
				M3	108.21	116.39	101.24	101.33
				M4	106.73	111.37	104.41	102.09
				M5	103.89	107.35	100.98	101.63

Table 5.12: Normalized Objective Function Values- 20 vertices

p	K	LB	KL		RND	SS	RND-KL	SS-KL
0.5	2	23.85	101.42	M1	107.70	137.43	102.21	104.22
				M2	108.51	136.51	101.67	101.77
				M3	108.00	137.81	102.20	103.68
				M4	117.33	146.67	101.33	101.33
				M5	106.14	117.33	100.52	101.60
	5	97.75	102.12	M1	111.23	120.26	101.43	102.69
				M2	110.99	117.77	102.21	102.80
				M3	109.82	118.22	102.19	102.30
				M4	111.60	123.35	105.73	102.79
				M5	107.80	107.00	102.23	102.26
	10	100	100	M1	102.31	106.00	100.00	100.00
				M2	101.12	104.79	100.00	100.00
				M3	102.28	106.10	100.00	100.00
				M4	101.12	104.49	100.00	100.00
				M5	100.26	100.95	100.00	100.00

Table 5.13: Normalized Objective Function Values- 20 vertices

p	K	LB	KL		RND	SS	RND-KL	SS-KL
0.75	2	82.67	100.62	M1	106.24	120.18	100.47	100.47
				M2	103.80	117.92	100.99	100.61
				M3	106.68	119.43	101.64	100.89
				M4	106.87	119.08	102.29	100.76
				M5	104.17	106.09	100.79	100.44
	5	100	100.1	M1	102.44	106.80	100.00	100.00
				M2	102.00	107.86	100.20	100.10
				M3	102.75	106.75	100.19	100.19
				M4	103.45	104.31	100.00	100.00
				M5	100.98	102.55	100.00	100.17
	10	100	100	M1	100.23	102.18	100.00	100.00
				M2	100.22	102.00	100.00	100.00
				M3	100.00	102.17	100.00	100.00
				M4	100.00	102.21	100.00	100.00
				M5	100.00	100.00	100.00	100.00

Table 5.14: Normalized Objective Function Values- 20 vertices

p	K	LB	KL		RND	SS	RND-KL	SS-KL
0.9	2	98.28	100.25	M1	102.70	109.14	100.36	100.48
				M2	102.82	109.64	101.23	100.12
				M3	103.43	109.88	100.49	100.61
				M4	105.66	111.95	100.63	100.63
				M5	101.60	102.29	100.84	100.25
	5	100	100	M1	100.21	101.99	100.00	100.00
				M2	100.21	102.06	100.00	100.00
				M3	100.36	102.57	100.00	100.00
				M4	100.00	103.57	100.00	100.00
				M5	100.00	100.91	100.00	100.00
	10	100	100	M1	100.06	100.81	100.00	100.00
				M2	100.00	100.69	100.00	100.00
				M3	100.13	100.69	100.00	100.00
				M4	100.00	100.00	100.00	100.00
				M5	100.00	100.00	100.00	100.00

The *running times* for all of the instances with 20 vertices were almost zero. We therefore did not include the tables for running times of the instances.

Examining Table 5.10 through 5.15, we can verify that using solutions of the two heuristics as an input to the KL algorithm further improves the solution of the KL algorithm. Boldface values in the tables show that in most cases, the best solution is among the last two columns, i.e., KL using two-stage and randomized algorithms.

One important observation from the above tables is the effect of lower bounds on the instances with value of p greater than or equal to 0.5. As an example, consider Tables 5.13 and 5.14. From these tables, for larger p values, lower bounds can even be equal to the optimal solution of the problem.

Table 5.15: Normalized Objective Function Values- 50 vertices

p	K	LB	KL		RND	SS	RND-KL	SS-KL
0.1	2	4.13	118.49	M1	178.23	230.96	**115.83**	116.16
				M2	178.23	239.65	**115.83**	115.43
				M3	178.23	227.54	**115.83**	116.16
				M4	180.74	240.57	119.26	115.98
				M5	152.72	145.61	116.03	119.67
	5	8.47	113.22	M1	156.48	162.43	114.10	111.55
				M2	155.55	159.42	114.71	113.50
				M3	155.60	160.84	113.37	113.57
				M4	155.17	158.22	111.16	111.97
				M5	145.07	138.08	112.52	**109.03**
	10	44.21	108.75	M1	139.37	148.99	107.96	109.75
				M2	136.67	148.21	110.26	109.78
				M3	137.58	150.11	109.16	109.48
				M4	136.91	145.54	108.02	109.08
				M5	128.27	125.03	107.75	**107.07**

Table 5.16: Normalized Objective Function Values- 50 vertices

p	K	LB	KL		RND	SS	RND-KL	SS-KL
0.25	2	0.96	104.36	M1	127.11	151.81	102.96	104.11
				M2	127.11	152.66	102.96	104.31
				M3	127.11	151.41	102.96	103.90
				M4	124.36	148.82	**101.04**	102.27
				M5	125.03	119.66	102.58	102.97
	5	44.14	102.54	M1	120.25	130.40	103.51	**101.98**
				M2	121.32	128.11	102.75	102.86
				M3	120.79	127.94	102.83	102.51
				M4	119.94	126.67	102.30	102.24
				M5	119.32	116.50	103.05	102.56
	10	89.57	101.73	M1	113.68	120.31	101.89	101.51
				M2	112.33	116.97	**100.78**	102.37
				M3	114.44	120.95	101.81	101.50
				M4	112.73	119.90	101.87	101.69
				M5	111.49	109.09	102.08	101.51

Table 5.17: CPU Times(s)- 50 vertices

p	K	KL		LP-Rel	RND	SS	RND-KL	SS-KL
0.1	2	0.03	M1	0	0	0	0.02	0.02
			M2	0	0	0	0.02	0.02
			M3	0	0	0	0.02	0.02
			M4	0.00	0	0	0.02	0.02
			M5	0	0	0	0.02	0.02
	5	0.01	M1	0.02	0	0	0.01	0.01
			M2	0.18	0	0	0.01	0.01
			M3	0.02	0	0	0.01	0.01
			M4	0.29	0	0	0.01	0.01
			M5	0.04	0	0	0.01	0.01
	10	0.01	M1	0.05	0	0	0.01	0.01
			M2	1.46	0	0	0.01	0.01
			M3	0.05	0	0	0.01	0.01
			M4	1.48	0	0	0.01	0.01
			M5	0.18	0	0	0.01	0.01

Table 5.18: CPU Times(s)- 50 vertices

p	K	KL		LP-Rel	RND	SS	RND-KL	SS-KL
0.25	2	0.03	M1	0	0	0	0.02	0.03
			M2	0	0	0	0.02	0.02
			M3	0.09	0	0	0.02	0.02
			M4	0.04	0	0	0.02	0.02
			M5	0	0	0	0.02	0.02
	5	0.01	M1	0.03	0	0	0.01	0.02
			M2	0.78	0	0	0.01	0.01
			M3	0.66	0	0	0.01	0.01
			M4	1.05	0	0	0.01	0.01
			M5	0.05	0	0	0.01	0.01
	10	0.01	M1	0.29	0	0	0.01	0.01
			M2	2.02	0	0	0.01	0.01
			M3	0.69	0	0	0.01	0.01
			M4	2.07	0	0	0.01	0.01
			M5	0.47	0	0	0.01	0.01

From Tables 5.16, 5.17, and 5.18 we can draw similar conclusions as in instances with 20 vertices. *The boldface values belong to method in which we use solution of the two heuristics as input to the KL algorithm.*

While the CPU time of the LP relaxation may increase in these samples, the KL algorithm and the two heuristic methods can quickly find a solution.

Table 5.19: Normalized Objective Function Values- 100 vertices

p	K	LB	KL		RND	SS	RND-KL	SS-KL
0.1	2	0.71	108.78	M1	151.97	171.07	108.47	107.72
				M2	151.97	170.26	108.47	**106.55**
				M3	151.97	171.95	108.47	108.89
				M4	150.52	172.47	107.46	106.90
				M5	152.06	130.61	109.31	109.06
	5	1.51	102.86	M1	136.33	137.26	**101.88**	103.69
				M2	136.33	140.03	**101.88**	103.15
				M3	136.33	138.67	**101.88**	104.03
				M4	136.35	139.06	102.37	103.31
				M5	133.72	121.55	103.36	102.34
	10	21.27	102.26	M1	125.51	129.77	102.68	102.35
				M2	125.01	129.39	103.04	102.16
				M3	126.53	129.80	102.02	102.10
				M4	129.28	135.05	100.44	104.51
				M5	123.03	117.53	102.04	**100.88**

Table 5.20: CPU Times(s)- 100 vertices

p	K	KL		LP-Rel	RND	SS	RND-KL	SS-KL
0.1	2	0.3	M1	0.01	0	0	0.29	0.32
			M2	0.01	0	0	0.29	0.28
			M3	0.23	0	0	0.29	0.27
			M4	0.06	0.00	0	0.35	0.36
			M5	0.02	0.00	0	0.31	0.23
	5	0.13	M1	0.13	0.01	0	0.13	0.13
			M2	1.12	0.00	0	0.13	0.13
			M3	0.60	0.00	0	0.13	0.12
			M4	1.24	0.01	0	0.15	0.15
			M5	0.42	0.01	0	0.12	0.11
	10	0.05	M1	2.54	0.01	0	0.05	0.05
			M2	5.75	0.01	0	0.05	0.06
			M3	3.04	0.01	0	0.05	0.06
			M4	6.68	0.01	0	0.06	0.07
			M5	2.17	0.01	0	0.06	0.05

Similarly, for the instances with 100 vertices, among all the methods, the most promising one is KL using our heuristic methods. Even for the instances with 100 vertices, the running time of the heuristics is always less than a second.

Since the p value of the graphs is 0.1, lower bounds cannot help in these cases.

5.2.3 Heuristic Results

In this section, we review some tables for approximate solutions by our two heuristic algorithms. We present the results for instance with 200, 400, and 1000 vertices. For graphs with larger than 2000 vertices, solving the LP relaxation becomes a bottleneck. As an example, we tried an instance with 2000 vertices, and the computer ran out of memory.

Similar to what we observed in the previous section, our results show that using the KL algorithm with initial solutions coming from our heuristics, may improve the solution, and also in certain cases it reduces the running time of KL as well.

For graphs with 400 and 1000 vertices, we only used M1 (LP relaxation of XZ by adding symmetry breaking constraints and also lower bounds).

Table 5.21: Normalized Objective Function Values- 200 vertices

p	K	LB	KL		RND	SS	RND-KL	SS-KL
0.1	2	0.14	102.03	M1	130.89	141.45	102.39	101.48
				M2	130.89	142.19	102.39	101.72
				M3	130.89	141.88	102.39	102.63
				M4	130.10	138.69	101.99	101.42
				M5	131.16	116.48	**101.06**	101.78
	5	0.32	101.24	M1	122.29	125.55	101.85	100.90
				M2	122.29	125.35	101.85	101.03
				M3	122.29	124.23	101.85	101.31
				M4	121.75	125.65	100.82	100.86
				M5	122.48	114.11	101.51	**100.80**
	10	11.91	100.91	M1	117.34	120.39	101.09	101.13
				M2	117.47	120.88	101.21	100.86
				M3	117.70	122.18	100.79	101.17
				M4	117.64	123.44	101.22	101.31
				M5	117.62	112.74	101.12	**100.57**

Table 5.22: CPU Times(s)- 200 vertices

p	K	KL		LP	RND	SS	RND-KL	SS-KL
0.1	2	5.18	M1	0.14	0.02	0	4.56	5.08
			M2	0.17	0.02	0	4.57	5.09
			M3	0.30	0.02	0	4.58	5.32
			M4	0.40	0.02	0	4.96	5.62
			M5	0.12	0.02	0	4.36	4.38
	5	2.18	M1	4.84	0.02	0	1.79	2.08
			M2	4.36	0.02	0	1.75	2.03
			M3	4.77	0.02	0	1.80	2.07
			M4	7.24	0.02	0	2.29	2.16
			M5	4.34	0.02	0	1.99	1.65
	10	0.68	M1	14.63	0.03	0	0.64	0.61
			M2	42.49	0.03	0	0.59	0.73
			M3	23.80	0.03	0	0.71	0.67
			M4	69.02	0.03	0	0.71	0.71
			M5	30.87	0.03	0	0.63	0.64

From Table 5.21, we can observe that the quality of solutions of our heuristics in comparison to the KL algorithm is not promising. However, in almost all cases, RND-KL and SS-KL give better solutions than the KL algorithm.

From Table 5.22, we observe that CPU times start to increase especially for solving the LP relaxation.

One interesting observation about the KL algorithm is that when the number of clusters increases, the running time of the algorithm decreases. Our results show that the KL algorithm in bipartitioning case has more promising objective function values in comparison with K-way partitioning.

Table 5.23: Normalized Objective Function Values- 400 vertices

p	K	LB	KL		RND	SS	RND-KL	SS-KL
0.10	2.00	0.03	**100.43**	M1	117.30	125.65	100.75	100.53
	5.00	0.08	100.23	M1	115.80	118.02	100.39	**100.05**
	10.00	6.27	100.29	M1	112.91	114.94	**100.12**	100.44

Table 5.24: CPU Times(s)- 400 vertices

p	K	KL	LP- M1	RND	SS	RND-KL	SS-KL
0.10	2	96.03	1.32	0.19	0.00	104.90	100.56
	5	39.07	23.38	0.13	0.00	36.37	40.41
	10	11.48	144.79	0.16	0.02	11.80	11.50

Similar to the instances with 200 vertices, the quality of the solution of the KL algorithm in instances with fewer number of clusters is better. For instance, in Table 5.23, we observe that the solution of KL algorithm is the best one. However, for other instances in which the number of clusters are 5 and 10, the best ones are SS-KL and RND-KL, respectively.

Table 5.25: Normalized Objective Function Values- 1000 vertices

p	K	LB	KL		RND	SS	RND-KL	SS-KL
0.10	2	0.00	100.56	M1	112.61	114.78	**100.00**	100.19
	5	0.01	**100.00**	M1	110.37	111.37	100.09	100.13
	10	3.04	**100.00**	M1	108.34	108.57	100.02	100.02

Table 5.26: CPU Times(s)- 1000 vertices

p	K	KL	LP- M1	RND	SS	RND-KL	SS-KL
0.10	2	5638.53	48.16	0.94	0.00	4178.18	4647.56
	5	39.07	2038.62	1.06	0.06	2031.03	3185.91
	10	11.48	3086.70	1.18	0.11	573.43	513.27

From Table 5.26, in the case where we have two clusters, the running time of KL starting from our heuristic methods is better than the running time of the KL algorithm. Hence, in larger instances, we can see the reduction in the running time of the KL algorithm using the solution of two heuristic methods as an input.

5.2.4 Clusters with Different Sizes

Our experiments so far were for the equipartition case. We conducted experiments using same graphs with 20 vertices as in the previous sections. Instead of two equal size clusters, i.e., $\{10, 10\}$, we set the sizes of clusters as $\{4, 16\}$. Tables below show the results for both exact models and our heuristics. Note that since CPU times for heuristics are almost zero, we did not report them. Similar to the previous experiments, these are also averaged over 10 different instances. Since the problem in these experiments can be solved to optimality by ILP formulation, we scaled all the optimal solutions to 100. Therefore we report only the normalized objective function values of our heuristic methods.

Table 5.27: Normalized Objective Function Values- 20 vertices, clusters (4,16)

p	K	LB	KL		RND	SS	RND-KL	SS-KL
0.1	2 (4,16)	47.37	152.63	M1:LB	189.47	347.37	157.89	168.42
				M1:LB+TI	163.16	300.00	152.63	168.42
				M1:LB+CI	184.21	342.11	152.63	163.16
				M1:LB+CI+TI	152.63	315.79	**142.11**	173.68
				M2	242.11	400.00	194.74	173.68
0.5	2(4,16)	11.64	108.36	M1:LB	106.87	136.12	101.79	104.18
				M1:LB+TI	106.57	135.82	102.39	103.88
				M1:LB+CI	105.67	142.69	101.79	106.57
				M1:LB+CI+TI	106.57	142.39	103.28	103.88
				M2	105.07	135.22	101.79	**101.49**

We can make the following observations.

Since clusters have different sizes, we cannot use symmetry breaking constraints. Lower bounds get worse as the sizes of clusters deviate from the equipartition case. KL algorithm, even for two clusters tends to behave worse in comparison with its performance in the equipartition case. Randomized algorithm gives better solution than KL algorithm and two-stage algorithm for the instances with the p value of 0.5. However using the solutions of our heuristic methods as an initial solution for the KL algorithm seems to be the most promising direction.

5.2.5 Very Large Instances

Table 5.28 presents a comparison between our two algorithms in large scale. Note that we have not any LP solution here, and we simply used the fractional values as given by Lemma 3.4.1 as an input to our heuristic methods. Note that this solution is the optimal solution of the LP relaxation when using the XZ formulation without adding any other constraints.

Table 5.28: Large instances

N	p	K	LB	RND	RND-Time	SS	SS-Time
2000	0.2	2	1	198809	5.51	199727	0.03
		5	3091	318225	6.48	319889	0.08
		10	200764	358737	7.35	360155	0.14
5000	0.05	2	1	310463	41.64	312750	0.02
		5	4	497016	77.29	499796	0.27
		10	9	560289	84.13	561913	0.56
10000	0.05	2	1	1245402	203.22	1250743	0.03
		5	4	1993645	303.26	1997886	0.58
		10	9	2243494	449.77	2248600	1.37
20000	0.03	2	1	2992020	1695.41	2999751	0.08
		5	4	4792141.00	1546.01	4801119	7.04

Chapter 6

CONCLUSIONS AND FUTURE WORK

In this thesis, we reviewed the extensive literature of graph partitioning problem and we used existing optimization formulations to propose our heuristic methods. We proposed two heuristic algorithms using the solution of the LP relaxation of these optimization formulations. We also implemented the KL algorithm, which is one of the widely used algorithms for solving graph partitioning problems, in an attempt to compare with our results. We conducted computational experiments using graphs with different characteristics. Based on these experiments, we discuss our observations.

Exact models can only obtain an optimal solution for instances with at most 50 vertices. To illustrate, using one of hour CPU time limit, an instance of GPP with 50 vertices and density of 0.5 stops running due to memory problems.

We observed that when there is no additional constraints for the LP relaxation of the XZ formulation, its optimal value is always zero and that the resulting optimal solution is not very useful for our heuristics. In this case, valid inequalities play an important role to improve the quality of the solution returned by the LP relaxation. However, one drawback of using the LP relaxation in large instances again is the memory problem. Since we have a large number of variables and constraints, by incorporating new valid inequalities we further increase the number of constraints in our problem. That is why after a certain threshold, personal computers cannot handle the LP relaxation.

We used several sets of lower bounds on the size of the cut set as valid inequalities. Since in our experiments, the edge weights are all equal to one, our lower bounds function as a lower bound on the optimal value of the objective function. An important observation regarding the efficiency of these lower bounds is as follows. When the

number of edges of an instance increases, i.e., as the *p-value* gets larger, lower bounds perform well in terms of reducing the running time or the optimality gap of the original ILP formulation. Especially in dense graphs, lower bounds are more effective.

Two-stage algorithm is a fast method to obtain a feasible solution for GPP. However, the quality of its output depends on the optimal solution of the LP relaxation. We can potentially obtain very good approximations if we can sufficiently strengthen the LP relaxation.

The randomized algorithm takes advantage of randomness. Therefore, if we run this algorithm in a repeated manner, the chance of getting better solutions increases. Similar to the two-stage algorithm, the quality of the solution of the LP relaxation is a critical issue for the performance of the randomized algorithm.

Despite the fact that we cannot outperform the KL algorithm in most cases, using the solution of our heuristic approaches as an initial solution to the KL algorithm, we can outperform the KL algorithm in several instances. Therefore our methods can be used to construct a good starting solution for the KL algorithm.

6.1 Future Research Directions

1. We can somehow strengthen our randomized algorithm by iteratively improving the solutions by generating random solutions at each step carefully. The crucial part of the randomized algorithm presented in this thesis is the input probabilities, and the algorithm assigns vertices to the clusters based on these probabilities. We can update the probabilities using the incumbent solution at each step by taking a convex combination of the previous probabilities and the best solution out of ten new random solutions. Such an approach can be helpful in terms of finding solutions in different parts of the feasible region, thereby increasing our chances to compute a good feasible solution.

2. Since the quality of the solution returned by both algorithms heavily depends on the solution of the LP relaxation, using better valid inequalities and lower

bounds may have a positive effect on the performance of these two algorithms.

3. We have other versions of the graph partitioning problem and our methods can be applied to those versions as well. In some cases, instead of specifying sizes of the clusters, we have an upper bound on the number of vertices in each cluster. Alternatively, we may have no restriction on the sizes of the clusters. We can study application of our heuristic methods in these problems.

BIBLIOGRAPHY

[1] E. G. Boman, K. D. Devine, and S. Rajamanickam, "Scalable matrix computations on large scale-free graphs using 2d graph partitioning," in "International Conference for High Performance Computing, Networking, Storage and Analysis, SC'13, Denver, CO, USA - November 17 - 21, 2013," (2013), pp. 50:1–50:12.

[2] A. Buluç and K. Madduri, "Graph partitioning for scalable distributed graph computations," in "Graph Partitioning and Graph Clustering - 10th DIMACS Implementation Challenge Workshop, Georgia Institute of Technology, Atlanta, GA, USA, February 13-14, 2012. Proceedings," (2012), pp. 83–102.

[3] S. Chu and J. Cheng, "Triangle listing in massive networks and its applications," in "Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 21-24, 2011," (2011), pp. 672–680.

[4] B. H. Junker and F. Schreiber, *Analysis of Biological Networks*, vol. 2 (John Wiley & Sons, 2011).

[5] S. Fortunato, "Community detection in graphs," Complex Networks and Systems Lagrange Laboratory **abs/0906.0612** (2009).

[6] G. Qiu, "Bipartite graph partitioning and content-based image clustering," in "1 st European Conference on Visual Media Production," (2004), pp. 87–04.

[7] B. Peng, L. Zhang, and D. Zhang, "A survey of graph theoretical approaches to image segmentation," Pattern Recognition **46**, 1020–1038 (2013).

[8] M. R. Garey, D. S. Johnson, and L. J. Stockmeyer, "Some simplified NP-complete graph problems," Theor. Comput. Sci. **1**, 237–267 (1976).

[9] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," Bell System Technical Journal **49**, 291–307 (1970).

[10] M. Conforti, M. R. Rao, and A. Sassano, "The equipartition polytope. I: Formulations, dimension and basic facets," Math. Program. **49**, 49–70 (1991).

[11] M. Conforti, M. R. Rao, and A. Sassano, "The equipartition polytope. II: Valid inequalities and facets," Math. Program. **49**, 71–90 (1991).

[12] C. E. Ferreira, A. Martin, C. C. de Souza, R. Weismantel, and L. A. Wolsey, "Formulations and valid inequalities for the node capacitated graph partitioning problem," Math. Program. **74**, 247–266 (1996).

[13] A. Lisser and F. Rendl, "Graph partitioning using linear and semidefinite programming," Mathematical Programming **95**, 91–101 (2003).

[14] D. Delling and R. F. F. Werneck, "Better bounds for graph bisection," in "Algorithms - ESA 2012 - 20th Annual European Symposium, Ljubljana, Slovenia, September 10-12, 2012. Proceedings," (2012), pp. 407–418.

[15] D. Delling, A. V. Goldberg, I. Razenshteyn, and R. F. F. Werneck, "Exact combinatorial branch-and-bound for graph bisection," in "Proceedings of the 14th Meeting on Algorithm Engineering & Experiments, ALENEX 2012, The Westin Miyako, Kyoto, Japan, January 16, 2012," (2012), pp. 30–44.

[16] M. M. Sørensen, "Facet-defining inequalities for the simple graph partitioning polytope," Discrete Optimization **4**, 221–231 (2007).

[17] N. Fan and P. M. Pardalos, "Linear and quadratic programming approaches for the general graph partitioning problem," J. Global Optimization **48**, 57–71 (2010).

[18] R. Sotirov, "An efficient semidefinite programming relaxation for the graph partition problem," INFORMS Journal on Computing **26**, 16–30 (2014).

[19] S. E. Karisch, F. Rendl, and J. Clausen, "Solving graph bisection problems with semidefinite programming," INFORMS Journal on Computing **12**, 177–191 (2000).

[20] Ü. V. Çatalyürek and C. Aykanat, "Patoh (partitioning tool for hypergraphs)," in "Encyclopedia of Parallel Computing," (2011), pp. 1479–1487.

[21] G. Karypis and V. Kumar, "Parallel multilevel series k-way partitioning scheme for irregular graphs," SIAM Review **41**, 278–300 (1999).

[22] I. Safro, P. Sanders, and C. Schulz, "Advanced coarsening schemes for graph partitioning," ACM Journal of Experimental Algorithmics **19** (2014).

[23] V. Yazici and C. Aykanat, "Constrained min-cut replication for k-way hypergraph partitioning," INFORMS Journal on Computing **26**, 303–320 (2014).

[24] A. Dulmage and N. Mendelsohn, "Two algorithms for bipartite graphs," Journal of the Society for Industrial & Applied Mathematics **11**, 183–194 (1963).

[25] O. Goldschmidt and D. S. Hochbaum, "A polynomial algorithm for the k-cut problem for fixed k," Mathematics of operations research **19**, 24–37 (1994).

[26] N. Guttmann-Beck and R. Hassin, "Approximation algorithms for minimum K-cut," Algorithmica **27**, 198–207 (2000).

[27] R. Battiti and A. A. Bertossi, "Greedy, prohibition, and reactive heuristics for graph partitioning," IEEE Trans. Computers **48**, 361–385 (1999).

[28] C. J. Alpert, J. Huang, and A. B. Kahng, "Multilevel circuit partitioning," IEEE Trans. on CAD of Integrated Circuits and Systems **17**, 655–667 (1998).

[29] A. Buluç, H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz, "Recent advances in graph partitioning," Preprint- Lawrence Berkeley National Laboratory **abs/1311.3144** (2013).

[30] H. Zha, X. He, C. H. Q. Ding, M. Gu, and H. D. Simon, "Bipartite graph partitioning and data clustering," tenth international conference on Information and knowledge management **cs.IR/0108018** (2001).

[31] Z. Li, X. Wu, and S. Chang, "Segmentation using superpixels: A bipartite graph partitioning approach," in "2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, June 16-21, 2012," (2012), pp. 789–796.

[32] J. E. Hopcroft and R. M. Karp, "An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs," SIAM J. Comput. **2**, 225–231 (1973).

[33] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," in "Proceedings of the 19th Design Automation Conference, DAC '82, Las Vegas, Nevada, USA, June 14-16, 1982," (1982), pp. 175–181.

[34] M. Rege, M. Dong, and F. Fotouhi, "Co-clustering documents and words using bipartite isoperimetric graph partitioning," in "Proceedings of the 6th IEEE International Conference on Data Mining (ICDM 2006), 18-22 December 2006, Hong Kong, China," (2006), pp. 532–541.

[35] I. S. Dhillon, "Co-clustering documents and words using bipartite spectral graph partitioning," in "Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, San Francisco, CA, USA, August 26-29, 2001," (2001), pp. 269–274.

[36] E. Balas and M. W. Padberg, "Set partitioning: A survey," SIAM Review **18**, 710–760 (1976).

[37] J. A. Lukes, "Efficient algorithm for the partitioning of trees," IBM Journal of Research and Development **18**, 217–224 (1974).

[38] L. R. Ford and D. R. Fulkerson, "Maximal flow through a network," Canadian Journal of Mathematics **8**, 399–404 (1956).

[39] J. B. Orlin, "Max flows in o(nm) time, or better," in "Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013," (2013), pp. 765–774.

[40] W. A. Chaovalitwongse, P. M. Pardalos, and O. A. Prokopyev, "A new linearization technique for multi-quadratic 0-1 programming problems," Oper. Res. Lett. **32**, 517–522 (2004).

[41] H. D. Sherali and J. C. Smith, "An improved linearization strategy for zero-one quadratic programming problems," Optimization Letters **1**, 33–47 (2007).

[42] N. Biggs, E. K. Lloyd, and R. J. Wilson, *Graph Theory, 1736-1936* (Oxford University Press, 1976).