GRAPH PARTITION PROBLEMS WITH MINIMUM SIZE CONSTRAINTS

By

Xiaoyun Ji

A Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major Subject: Mathematics

Approved by the
Examining Committee:

Rensselaer Polytechnic Institute
Troy, New York

Nov 2004
(For Graduation Dec 2004)

GRAPH PARTITION PROBLEMS WITH MINIMUM SIZE CONSTRAINTS

By

Xiaoyun Ji

An Abstract of a Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major Subject: Mathematics

Rensselaer Polytechnic Institute
Troy, New York

Nov 2004
(For Graduation Dec 2004)

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENT

I would like to express my most sincere gratitude to my advisor Prof. *John E. Mitchell*. I have learned a great deal from him, on matters both mathematical and beyond.

I am also very grateful for the helps and comments I have received from the other members in my thesis committee: Prof. *Kristin Bennett*, Prof. *Franklin Luk*, Prof. *Jong-shi Pang*. I thank Prof. *Jeff Linderoth* on his help with MINTO software. I would also like to take this opportunity to thank the professors and staffs in the math department who helped me in many different ways.

I gratefully acknowledge the financial support from RPI as teaching and research assistants. In particular, this research was supported in part by NSF grant number DMS-0317323.

I owe special thanks to my friends and fellows here at RPI: *Rui Chen, Gang Chen, Nicolas Chevaugeon, Jianyu Deng, Kartik Krishnan, Grigorios Pavliotis, Chengyu Shi, Gehua Yang, Laurel Reilly-Raska,* and all the friends who shared so many good moments with me at Eagle's Republic. I also thank my far away yet close friend on the other side of the globe, *Nan Wang*. They have made my time in graduate school enjoyable well beyond mathematics.

Finally, I would like to dedicate this work to my father *Xiangqi Ji*, my mother *Fengxian Xie* and my brother *Yaonan Ji*. Their love and support has accompanied me on every step I have taken. Despite the enormous physical distance between us, we are always together.

ABSTRACT

In this thesis, we use Integer Programming (IP) to develop exact algorithms for the following graph partition problem with minimum size constraints: given a complete graph $K_n = (V, E)$, with edge weight c_e on each edge, the graph partition problem with minimum size constraints requires partitioning the vertices of K_n into sets that each have at least S vertices, so as to minimize the sum of weights of the partitions. It is similar to the classic graph partition problem, but the additional size constraint makes it much harder to solve.

We consider two different measures to represent the weight of each partition. The Clique Partition Problem with Minimum Size Constraint (CPPMIN) uses the sum of weights of all the edges in each partition. The Minimum Weight Constrained Forest (MWCF) problem uses the weight of the edges in the minimum spanning tree of each partition. Both are NP-hard problems and can be used to model a large variety of practical optimization problems, including micro-aggregation of statistical data, sports team alignment, political districting and telecommunication network design.

We first try to formulate and solve the IP problems using branch-and-cut. For both problems, we discuss the polyhedral structure, derive strong valid inequalities for their corresponding polytopes, and design routines to generate the violated inequalities as cutting planes. Our computational results show that with these strong cutting planes, a branch-and-cut algorithm leads to high quality solutions in a reasonable amount of time.

We then formulate and solve the CPPMIN using branch-and-price-and-cut to compare with the branch-and-cut method. We successfully combine row generation and column generation to yield strong LP relaxations. We discuss the properties of the column generation problem, minimum node-edge-weighted cluster problem, and solve it as an IP. Our computation shows similar performance to the branch-and-cut algorithm.

CHAPTER 1
Introduction

1.1 Scope and Motivation

In this thesis we consider using integer programming to solve graph partition problems with minimum size constraints: given a complete graph $K_n = (V, E)$, with edge weight c_e on each edge, we want to divide the vertices of a graph into sets that each have at least S vertices, so as to minimize the sum of weights of the partitions. We consider two different measures to represent the weight of each partition. The Clique Partition Problem with Minimum size constraint (CPPMIN) uses the sum of weights of all the edges in each partition. The Minimum Weight Constrained Forest problem (MWCF) uses the weight of the edges in the minimum spanning tree of each partition. Both are NP-hard problems and can be used to model a large variety of practical optimization problems, including micro-aggregation of statistical data, sports team alignment, political districting and telecommunication network design.

Because of the wide applications and the difficulty of general graph partition problems, extensive research has been done on general graph partition problems and its variations, including both exact and heuristic algorithms. But surprisingly, we found little literature on partitions with a lower bound on the cluster sizes. Literature on exact algorithms for graph partition problems focuses mostly on clique partition problems with no additional constraints, see Grötschel and Wakabayshi [23, 24]; partitioning with a given number of clusters, see Chopra and Rao[10]; maximization partitioning problems with knapsack upper bound constraints, see Mehrotra and Trick [41]; equipartition, see Conforti, Rao and Sassano[12]; or k-way equipartition, see Mitchell [44, 47]. A similar phenomenon is noticed on knapsack problems. While a lot of attention is paid to the knapsack problems with upper bound constraints, for example quadratic knapsack problem, little literature is available for knapsack lower bound constrained problems. We hope our treatment of this particular graph partition problem with minimum size constraints can lead to a start on this relatively unexplored research area, and eventually provide more options for modelling and solving real-world problems.

Integer programming (IP) has been widely used in solving graph partition problems. The success obtained on solving practical instances of problems such as clique partition problems, equipartition problems, partition problems with knapsack constraints, was a motivation of the study that is carried out in this thesis.

Branch-and-cut, branch-and-price are two common methods for solving IP's. Both proved to be successful on various problems. We would like to test these two different methods on the same problem, CPPMIN, and make a comparison between the two. We hope to achieve a better understanding on the advantages and drawbacks of each method.

The rest of this chapter is organized as follows. In the next section, we define a group of related graph partition problems. Some commonly used terminology from graph theory is also included here. Section 3 contains some basic concepts of polyhedral theory that provide tools to characterize how strong a constraint is. Section 4 defines IP and introduces the methods for solving IP, including branch-and-cut, branch-and-price. Finally in section 5, we give the thesis outline.

1.2 Graph Theory Background

1.2.1 Basic Definitions

We will first give some definitions from graph theory that we use throughout the text. Additional definitions will be given as they are needed.

A **simple graph** $G = (V, E)$ consists of a finite set of nodes (or vertices) V and a set of edges $E \subseteq \{\{u, v\} | u, v \in V\}$ joining different pairs of distinct nodes. When the edges are defined by unordered pairs of nodes, the graph is **undirected**, otherwise it is a **directed** graph. All graphs discussed in this thesis are undirected simple graphs. A graph is called **complete**, if there exist an edge joining each pair of distinct nodes, such a graph is also called a **clique**. A complete undirected graph on of n vertices is denoted by K_n.

Two nodes i and j are **adjacent** if the edge $e_{ij} = (i, j) \in E$, and e_{ij} is said to be **incident** to the vertices i and j. The number of edges incident to a node is called the **degree** of the node. Notice the degree of every vertex in a complete graph K_n is $n-1$.

Throughout this thesis, we are going to use the following notation for some special edge sets. Given a graph $G=(V,E)$, let V_1 and V_2 be two disjoint subsets of V, we define

$$\begin{aligned}
\delta(V_1) &= \{uv \in E | u \in V_1, v \notin V_1\} \\
\delta(V_1, V_2) &= \{uv \in E | u \in V_1, v \in V_2\} \\
\delta(V_1, V_2, \ldots, V_k) &= \{uv \in E | u \in V_i, v \in V_j, 1 \le i < j \le k\} \\
E(V_1) &= \{uv \in E | u, v \in V_1\} \\
E(V_1, V_2) &= \{uv \in E | u, v \in V_1\} \cup \{uv \in E | u, v \in V_2\} \\
E(V_1, V_2, \ldots, V_k) &= \cup_{i=1\ldots k}\{uv \in E | u, v \in V_i\}
\end{aligned}$$

A node sequence $v_0, v_1, \ldots, v_k$ with $k \ge 1$ is a $v_0 - v_k$ **walk** if $(v_{i-1}, v_i) \in E$ for $i = 1, \ldots, k$. Node v_0 is the origin of the walk and node v_k is the destination. Nodes $v_1, \ldots, v_{k-1}$ are intermediate nodes. The walk has length k. The walk can also be represented by its edge sequence: $e_1, \ldots, e_k$, where $e_i = (v_{i-1}, v_i)$. A walk is called a **path** if there are no node repetitions. A $v_0 - v_k$ walk is **closed** if $v_k = v_0$. A closed walk is a **cycle** if $k \ge 3$ and $v_0, v_1, \ldots, v_k$ is a path. A graph is **acyclic** if it contains no cycles. The **length** of a cycle is the number of edges in the cycle.

Two vertices u and v in V are **connected** in $G=(V,E)$ if there exists a $u-v$ path in G. Two vertices are in the same **component** of G if they are connected. $G=(V,E)$ is **connected** if it has exactly one component.

For $V' \subseteq V$, The graph $G'=(V',E')$ is a **subgraph** of G if $V' \subseteq V$ and $E' \subseteq E$. G' is the subgraph **induced** by V' if $E' = E(V')$. G' is a **spanning subgraph** if $V' = V$.

An acyclic graph is called a **forest**. A connected forest is a **tree**. A **spanning tree** of $G=(V,E)$ is a spanning subgraph that is a tree.

Proposition 1.2.1. *Let $G=(V,E)$ be a graph on n nodes. The following statements are equivalent:*

- *G is a tree.*

- *There is a unique path between each pair of nodes.*
- *G contains $n-1$ edges and is connected.*
- *G contains $n-1$ edges and is acyclic.*
- *G is acyclic and connected.*

1.2.2 Graph Partitioning Problems

With the above definitions we here formalize the Clique Partition Problem and its variations.

Given a graph $G = (V, E)$, a **clustering** of G is a dividing of V into $\Pi = \{V_1, V_2, \ldots, V_k\}$, where $V = \cup_{i=1..k} V_i$ and $V_i \cap V_j = \emptyset, \forall 1 \leq i < j \leq k$. $V_1, V_2, \ldots, V_k$ are the components in the clustering, sometimes referred to as clusters or partitions. When G is a complete graph, we also call these components subcliques. We refer to the edge set $E(V_1, V_2, \ldots, V_k)$ as the **partition set** or simply **partition**, we refer to the edge set $\delta(V_1, V_2, \ldots, V_k)$ as the **multi-cut set** or simply **multi-cut**. When V is partitioned into just 2 subsets, V_1 and V_2, the resulting cut set $\delta(V_1, V_2)$ is simply called a **cut** of G. Notice here $E(G) = \delta(\Pi) \cup E(\Pi)$, $\delta(\Pi) \cap E(\Pi) = \emptyset$.

Clique Partition Problem: given a complete graph $G = (V, E) = K_n$, each edge is associated with an edge weight c_e. The Clique Partition Problem (CPP) is to partition the vertices V into sets of **any number and any size**, so as to minimize the total weight of the edges that have both endpoints in the same subclique.

Equipartition Problem: given a graph $G = (V, E)$, each edge is associated with an edge weight c_e. The Equipartition problem is to partition the vertices into **two sets of equal size**, so as to minimize the total weight of the edges that have both endpoints in the same subclique.

K-way Equipartition Problem: given a graph $G = (V, E)$, each edge is associated with an edge weight c_e. The k-way equipartition problem is to partition the vertices into k **sets of equal size**, so as to minimize the total weight of the edges that have both endpoints in the same subclique.

K-way Partition Problem: given a graph $G = (V, E)$, each edge is associated with an edge weight c_e. The k-way partition problem is to partition the vertices into **no more than** k **sets**, so as to minimize the total weight of the edges that have both endpoints in the same subclique.

Capacitated Partitioning problem: given a complete graph $G = (V, E) = K_n$, each edge is associated with an edge weight c_e, each vertex is associated with a vertex weight d_v. The problem is to partition the vertices V into sets that **satisfy certain weight restriction on vertex weights**, so as to minimize the total weight of the edges that have endpoints in different subcliques.

Maxcut Problem: given a graph G=(V,E), each edge is associated with a non-negative edge weight c_e, we want to divide the vertices into **2 partitions** so as to **maximize** the total weight of the edges that have endpoints in **different subsets**.

Like many other graph optimization problems, all of the above partition problems are NP-hard, this means that any exact algorithm known that can solve one of these problems needs CPU time that grows exponentially with the number of nodes in the graph unless $P = NP$. Thus one can only expect to solve such problems exactly for graphs with a limited number of nodes. The various techniques we are going to introduce, in particular, the integer programming approach, is trying to push up these limits. For a detailed introduction on complexity theory, including P and NP, we refer to Nemhauser and Wolsey [51]. Sipser [61] discusses this subject from a computer science point of view.

1.3 Polyhedral Theory

In this section, we are going introduce some basic definitions and properties in polyhedral theory that are used in this thesis. Readers can refer to Nemhauser and Wolsey [51] for a more comprehensive treatment of these subjects.

1.3.1 Basic Definitions

Definition 1.3.1. A set $S \in \mathbb{R}^n$ is a **subspace** of $\mathbb{R}^n$ if every linear combination of points in S is also in S.

Definition 1.3.2. A set of points $x^1, \ldots, x^k \in \mathbb{R}^n$ is **linearly independent** if the unique solution to $\sum_{i=1}^{k} \lambda_i x^i = 0$ is $\lambda_i = 0$ for all $i \in \{1, \ldots, k\}$.

Definition 1.3.3. A set of points $x^1, \ldots, x^k \in I\!R^n$ is **affinely independent** if the unique solution to $\sum_{i=1}^{k} \lambda_i x^i = 0, \sum_{i=1}^{k} \lambda_i = 0$, is $\lambda_i = 0$ for all $i \in \{1, \ldots, k\}$.

Proposition 1.3.1. *A set of points $x^1, \ldots, x^k \in I\!R^n$ is affinely independent if and only if the vectors $x^2 - x^1, x^3 - x^1, \ldots, x^k - x^1$ are linearly independent.*

Definition 1.3.4. Given a scalar α and a vector $a \in I\!R^n$, the set $\{x : a^T x \leq \alpha\}$ is a **halfspace**.

Definition 1.3.5. A **polyhedron** is a finite intersection of halfspaces.

Definition 1.3.6. The **dimension of a subspace** is the maximum number of linearly independent vectors in it.

Proposition 1.3.2. *Every affine space is a translation of a subspace. Further the subspace is uniquely defined by the affine space.*

Definition 1.3.7. The **dimension of an affine space** is the dimension of the corresponding subspace.

Definition 1.3.8. Let $S = \{x^1, \ldots, x^k\}$ be a set of points in $I\!R^n$.

The **convex hull** of S is the set of all convex combinations of points in S, i.e.,

$$conv(S) = \{\sum_{i=1}^{k} \lambda_i x^i : \sum_{i=1}^{k} \lambda_i = 1, x^i \in S, \lambda_i \in R, \lambda_i \geq 0, i = 1, \ldots, k\}$$

The **affine hull** of S is the set of all affine combinations of points in S, i.e.,

$$conv(S) = \{\sum_{i=1}^{k} \lambda_i x^i : \sum_{i=1}^{k} \lambda_i = 1, x^i \in S, \lambda_i \in R, i = 1, \ldots, k\}$$

Definition 1.3.9. The **dimension of a polyhedron** is the dimension of its affine hull.

Definition 1.3.10. Let P be a polyhedron. Let H be the hyperplane $H := \{x : a^T x = \alpha\}$. Let $Q = P \cap H$. If $a^T x \geq \alpha$ for all $x \in P$, then Q is a **face** of P.

Definition 1.3.11. Let P be a polyhedron of dimension d. A face of dimension $d-1$ is a **facet**. A face of dimension 1 is an **edge**. A face of dimension 0 is a **vertex**.

Refer to Figure 1.1 for a graph illustration of facets. R is a set of 8 discrete points on a plane, represented as black dots. Figure 1.1 gives 2 facets for the convex hull of R.

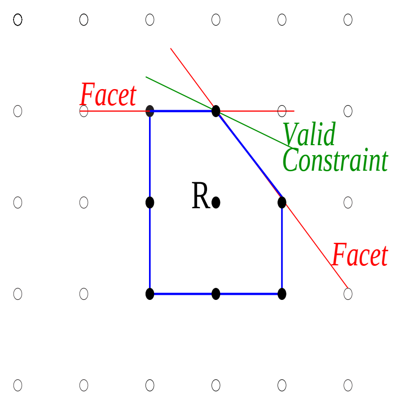

Figure 1.1: Facets for a Polyhedron

Theorem 1.3.1. *(See, for example, Nemhauser and Wolsey [51], Theorem 3.5)*

(i) A full-dimensional polyhedron P has a unique (to within scalar multiplication) minimal representation by a finite set of linear inequalities. In particular, for each facet F_i of P there is an inequality $a_i^T x \leq b$ (unique to within scalar multiplication) representing F_i and $P = \{x \in R^n : a_i^T x \leq b_i \text{ for } i = 1, ..., t\}$.

(ii) If $dim(P) = n - k$ with $k > 0$, then $P = \{x \in R^n : a_i^T x = b_i \text{ for } i = 1, ..., k, a_i^T x \leq b_i \text{ for } i = k+1, ..., k+t\}$. For $i = 1, ..., k$, (a_i^T, b_i) are a maximal set of linearly independent rows of $(A^=, b^=)$, and for $i = k+1, ..., k+t$, (a_i^T, b_i) is any inequality from the equivalence class of inequalities representing the facet F_i.

Definition 1.3.12. The **support** of an inequality $\pi^T x \leq \pi_0$ is given by the set $\{j \in \{1, \ldots, n\} : \pi_j \neq 0\}$.

Suppose that the x variables are in one-to-one correspondence with the edges of a graph G. Given the inequality $\pi^T x \leq \pi_0$, let S be the support of this inequality. The subgraph of G whose edges are indexed by the elements of S is called the **support graph** of the inequality $\pi^T x \leq \pi_0$.

1.3.2 Describing Polyhedra by Extreme Points and Extreme Rays

Let $P := \{x \in I\!R^n : Ax \leq b\}$ where A is a $m \times n$ matrix, x is an n-vector, b is an m-vector. Assume $rank(A) = n$ and $P \neq \emptyset$.

Definition 1.3.13. A point $x \in P$ is an **extreme point** of P if there do not exist points $x^1, x^2 \in P$ and a scalar λ (with $x^1 \neq x^2$ and $0 < \lambda < 1$) such that $x = \lambda x^1 + (1-\lambda)x^2$.

Definition 1.3.14. Let $P^0 := \{r \in I\!R^n : Ar \leq 0\}$. Any $r \in P^0 \backslash \{0\}$ is a ray of P^0. A point $r \in I\!R^n$ is a **ray** of P if and only if for any point $x \in P$ the set $\{y \in I\!R^n : y = x + \lambda r, \lambda \geq 0\} \subseteq P$. A ray r of P is an **extreme ray** if there do not exist rays $r^1, r^2 \in P^0$ and a scalar μ (with $r^1 \neq \lambda r^2$ for any $\lambda > 0$ and $0 < \mu < 1$) such that $r = \mu r^1 + (1-\mu)r^2$.

Theorem 1.3.2. *The polyhedron $P \subseteq I\!R^n_+$ can be represented as*

$$\begin{aligned} P = \{x \in I\!R^n_+ : \quad & x = \sum_{i \in K} \lambda_k x^k + \sum_{j \in J} \mu_j r^j \\ & \textit{with } \sum_{k \in K} \lambda_k = 1, \lambda_k \geq 0 \quad \forall k \in K, \\ & \mu_j \geq 0 \quad \forall j \in J \qquad \} \end{aligned}$$

where $\{x^k\}_{k \in K}$ is the set of extreme points of P and $\{r^j\}_{j \in J}$ is the set of the extreme rays of P.

1.4 Integer Programming Formulation and Algorithms

A Linear Programming (LP) problem can be defined as the problem of optimizing a linear function over a polyhedron. Such a problem can be written in matrix form as

$$\begin{aligned} \min \quad & c^T x \\ \text{subject to} \quad & Ax \leq b \end{aligned} \tag{1.1}$$

where $A \in I\!R^{m\times n}, b \in I\!R^m$ and $c \in I\!R^n$. If the variables are constrained to be integers, we have an Integer Programming (IP) problem.

$$\begin{aligned} \min \quad & c^T x \\ \text{subject to} \quad & Ax \leq b \\ & x \in Z^n \end{aligned} \tag{1.2}$$

If the variables are constrained to take values in {0,1}, we have a 0-1 IP problem. In this section, we are going to introduce the methods used for solving IP problems.

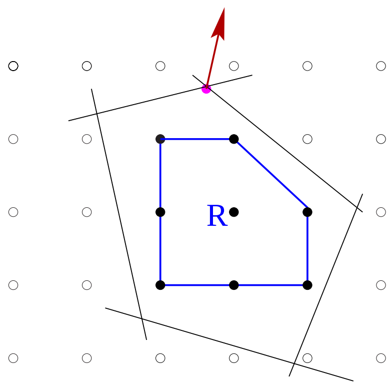

Figure 1.2: IP and LP

Let R denote the feasible region of problem (1.2), i.e. $R := \{x \in Z^n : Ax \leq b\}$. If an efficient representation of the convex hull of R, $conv(R)$, can be found, then the IP problem can be solved by solving the LP relaxation on $conv(R)$, since the optimal solution of an LP lies on the extreme points and the extreme points of $conv(R)$ are in R. But such a $conv(R)$ is not easily found. In most cases $conv(R)$ has an exponential number of facets, this is true not only for NP-hard problems, but also for some polynomially solvable problems, such as the matching problem, see Edmonds and Johnson [16]. In many cases a concrete description of $conv(S)$ is not known; even if an exponential number of facets are allowed, i.e. we don't know all families of facet-defining inequalities. This is often, though not always, the case for NP-hard problems. Figure 1.2 illustrates the relationship between the IP feasible region R, $conv(R)$ and the feasible region of an LP relaxation. The integer feasible region R is represented as black dots.

Below, we are going to give a brief introduction on some of the common methods used for solving IP problems. For each method, there are extensive amount of research and papers on it. The readers can refer to Mitchell [45, 46, 43], Lee and Mitchell [36] for surveys on cutting plane methods, branch-and-bound methods, and branch-and-cut methods, Barnhart [5] for a survey on branch-and-price, Savelsbergh [59] for a review of branch-and-price and its application on general assignment problems.

1.4.1 Cutting Plane Method

The motivation for a cutting plane method is that a complete description for the IP feasible region usually involves too many constraints to handle efficiently, and many of them won't be binding at the optimal solution anyway, so instead of adding them all in at the beginning, we can add in only those violated ones. Thus we have a sequence of LP relaxations of the IP problem. The relaxations are gradually improved to give better approximations to $conv(R)$, at least in the neighborhood of the optimal solution.

As illustrated in Figure 1.3-1.5, the basic steps of a cutting plane algorithm is:

- Solve the linear programming relaxation.
- If the solution to the relaxation is feasible in the integer programming problem, STOP with optimality.
- Otherwise, find one or more cutting planes that separate the optimal solution to the relaxation from the convex hull of feasible integer points, and add a subsets of these constraints to the relaxation.
- Return to first step.

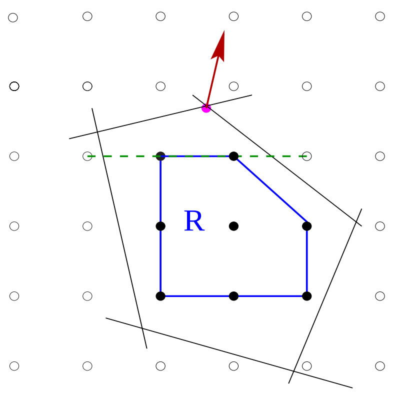

Figure 1.3: Cutting Plane Algorithm I: Solve initial relaxation, look for cutting planes

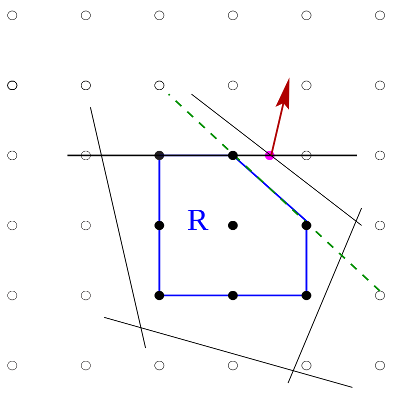

Figure 1.4: Cutting Plane Algorithm II: Add cutting plane, reoptimize, look for new cutting planes

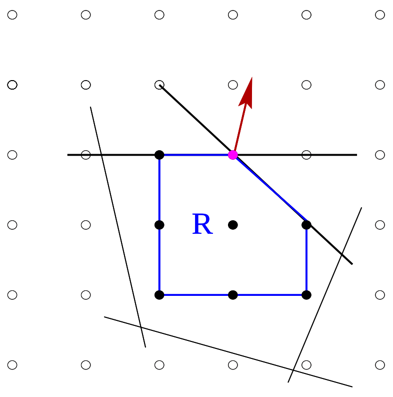

Figure 1.5: Cutting Plane Algorithm III: Add cutting plane, reoptimize, get an integer optimal solution

One way of generating general cutting planes involves combining together inequalities from the current description of the linear programming relaxation. Gomory cuts generated from simplex tableau is one of them and it was first proposed by Gomory [21, 22], then generalized to *Chvátal-Gomory cutting planes* by V. Chvátal in [11]. These types of cutting planes are often not strong and converge slowly to the IP feasible region since they do not utilize the special structure of the underlying problems. But because of the same reason, one does not need to write specialized routines to find the cutting planes. They are usually already implemented in general IP solvers.

Stronger cutting planes can be generated by investigating the polyhedral structure of convex hull of the set of integer feasible points that correspond to a specific problem. Current research is focused on developing such cutting plane algorithms for a variety of hard combinatorial optimization problems, and on solving large instances of IP problems using these methods. Chapters 2 and 3 of this thesis fall into this category of research. Jünger and et al. [35] contains a survey of problems that have been solved using strong cutting plane algorithms.

Typically in these algorithms, first a partial polyhedral description of the convex hull of the set of integer feasible points is determined. This description will usually contain families of facets of certain types. But most of the constraints will be left out of the LP relaxation at the beginning. Then if an optimal solution to an LP relaxation is infeasible, a subproblem, called the *separation problem*, is solved to try to identify violated inequalities in a class. If one or more violated inequalities are found, some are added to the LP to cut off the infeasible solution. Then the LP is re-optimized.

When no violated cutting planes can be found, we are going to resort to Branch-and-bound, which is the other basic method for integer programming problems.

1.4.2 Branch-and-Bound

The branch-and-bound method is based on the general principle of divide-and-conquer. The idea is to iteratively divide the feasible region (branching) to form subproblems of the original IP. Each subproblem is solved - either exactly or approximately – to obtain a lower bound on the subproblem objective value. It is essentially a smarter enumeration method. The advantage over a brute force exhaustive search method lies in the fact that if a lower bound for the objective value of a given subproblem is greater than the objective value of a known integer feasible solution, then the optimal solution of the original integer program problem cannot lie in the region associated with the given subproblem, thus this subproblem is pruned out, therefore reducing the searching space.

1.4.3 Branch-and-Cut

Combining cutting plane and branch and bound together leads us to the branch and cut algorithm: at each node of the branch-and-bound tree, a cutting plane method is used. An illustration of branch-and-cut algorithm is shown as a tree in Figure 1.6.

We should notice here that either in a pure cutting plane method, or a branch-and-cut scheme, the LP relaxations only need to be solved approximately, since we are just using the LP solution to generate cuts. This suggest the use of the interior point method to solve the LP relaxation, since it can gradually tighten up the degree of accuracy while a simplex method can not. The interior point solutions also lead to tighter cutting planes. On the other hand, for most LP problems, it is still faster to solve them using simplex method. Therefore, the question on how to choose between simplex and interior point method does not have a straight forward answer. Previous research on this includes an interior point cutting plane algorithm in Mitchell [42], and a combined interior point and simplex cutting plane algorithm in Mitchell and Borchers [48].

1.4.4 Branch-and-Price

The philosophy of branch-and-price is similar to that of branch-and-cut except that the procedure focuses on column generation rather than row generation. In fact, pricing and cutting are complementary procedures for tightening an LP relaxation. In branch-and-price, sets of columns are left out of the LP relaxation because there are too many columns to handle efficiently and most of them will

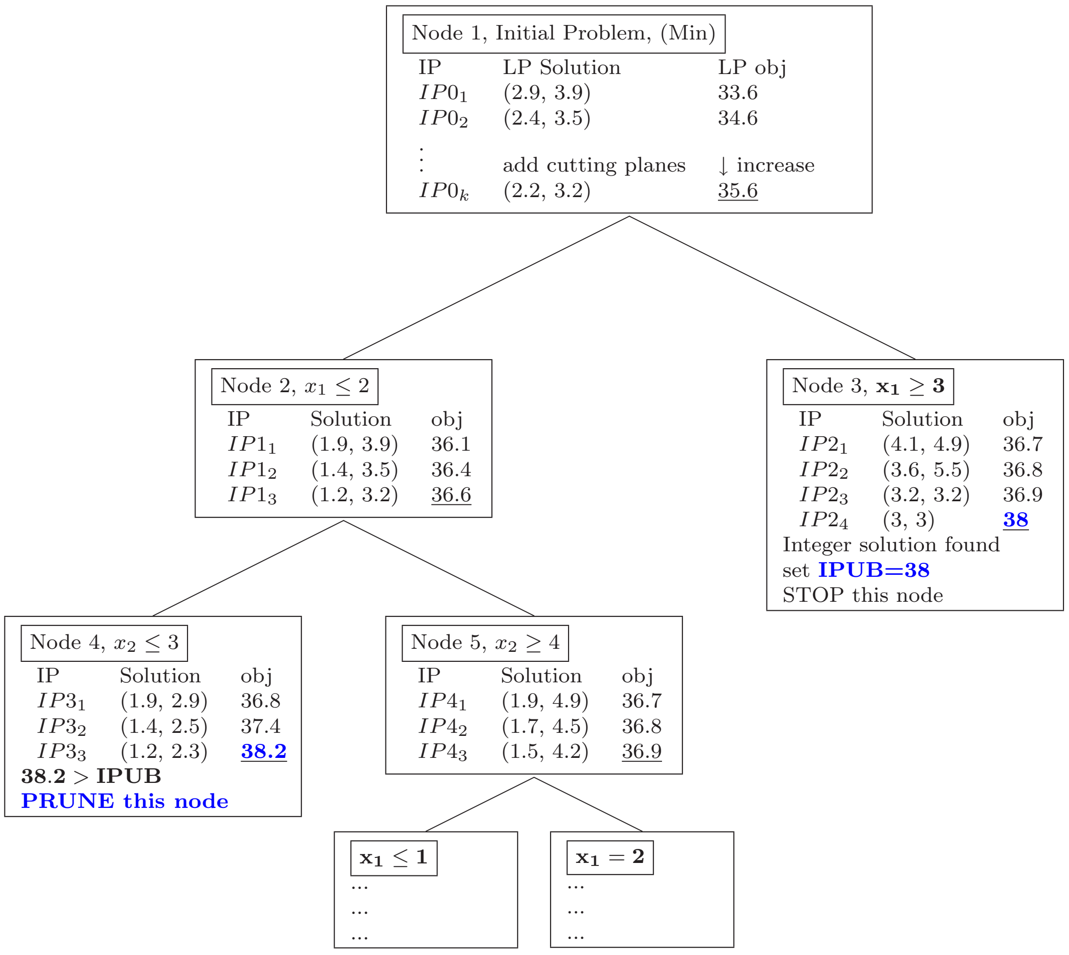

Figure 1.6: Branch-and-Cut Method

have their associated variable equal to zero in an optimal solution anyway. Then to check the optimality of an LP solution, a subproblem, called the pricing problem, which is a separation problem for the dual LP, is solved to try to identify columns to enter the basis. If such columns are found, the LP is re-optimized. Branching occurs when no columns price out to enter the basis and the LP solution does not satisfy the integrality condition.

Branch-and-price, which is also a generalization of branch-and-bound with LP relaxations, allows column generation to be applied throughout the branch-and-bound tree. The fundamental difficulties in applying column generation techniques for linear programming in integer programming solution method mainly lies in three aspects: how to find a good algorithm for the pricing subproblems; how to do branching effectively in branch and price scheme; and how to stabilize the dual variable to reduce oscillation in the degenerate case. We will discuss these aspects in detail in Chapter 4 when we apply the branch-and-price method to our problems. Branch-and-price can also be illustrated as a tree in Figure 1.7.

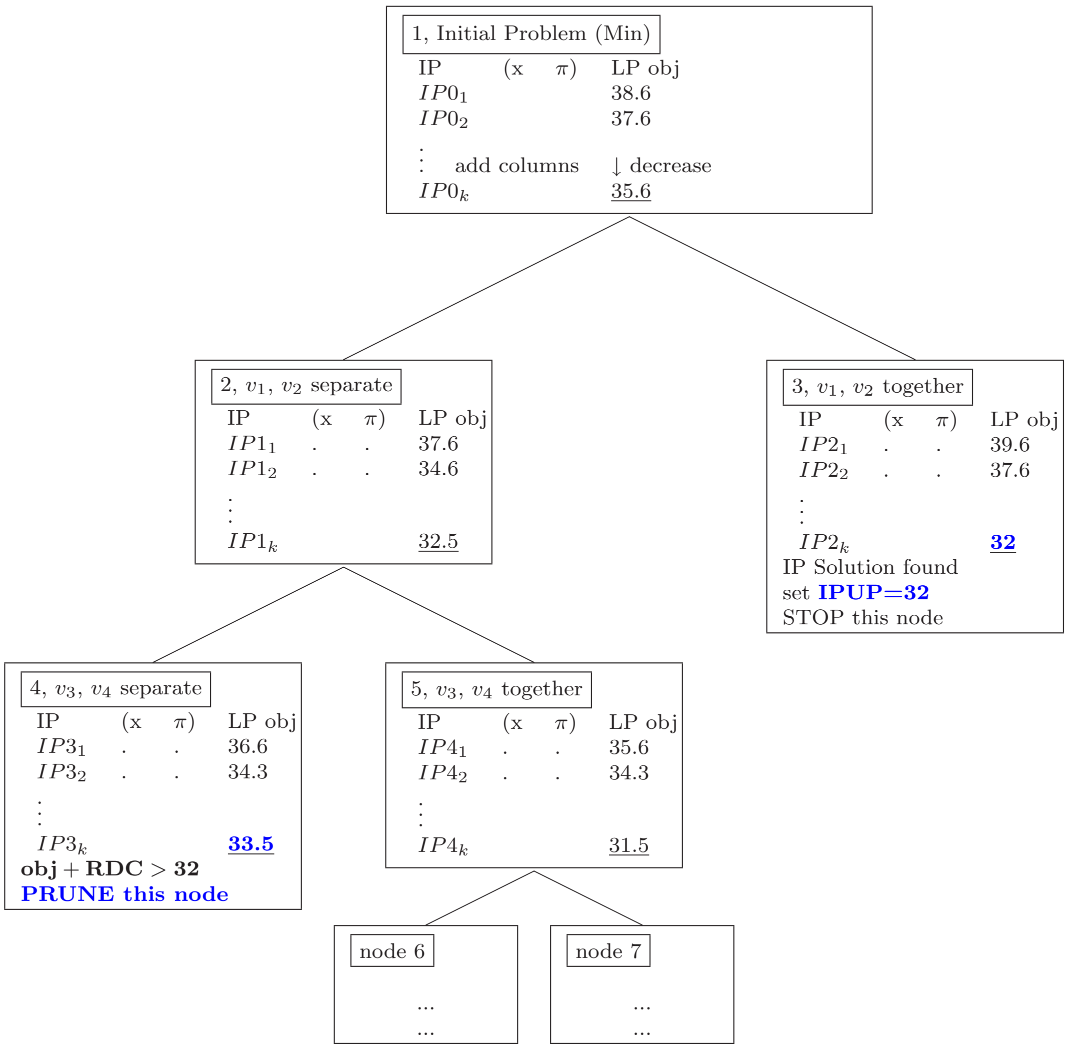

Figure 1.7: Branch-and-Price Method

Among the three aspects mentioned, the most important one is to be able to find a good and fast algorithm for the pricing subproblems because the pricing subproblems are usually hard problems themselves. Various methods have been proposed on this direction, including heuristics, integer programming technologies, branch-and-bound combined with combinatorial methods.

Lübbecke and Desrosiers [38] give a survey on column generation methods used for integer programming, with an emphasis on the dual perspective, which leads to stabilization techniques of the dual solutions.

Finally, combining branch-and-price, branch-and-cut, into branch-and-price-and-cut is also a challenge. Branch-and-price-and-cut usually can provide very strong LP relaxations. But solving the pricing problem after new cutting planes are generated can pose major difficulty, since the cutting planes might destroy the structure of the pricing problem. In Chapter 4, we will explain how we achieve this for our problem.

1.5 Thesis Outline

The remaining chapters of the thesis are organized as follows.

In Chapter 2, we investigate the facial structure of the convex hull of CPPMIN. We discuss the polyhedral structure and introduce new kinds of cutting planes: pigeon cutting planes and flower cutting planes. We will report the computational results with a branch-and-cut algorithm confirming the strength of these new cutting planes.

In Chapter 3, we investigate the facial structure of the convex hull of MWCF. We take the same approach as in Chapter 2 to discuss polyhedral structure to find classes of strong valid inequalities for the branch-and-cut algorithm.

In Chapter 4, we solved the same CPPMIN problem as in Chapter 2 using a branch-and-price-and-cut method. We add cutting planes to strengthen the LP relaxation and solve the pricing problem, the minimum node-edge-weighted cluster problem, as an integer programming problem. We compare the computational results with those in Chapter 2.

Finally, Chapter 5 contains our conclusions and suggestions for future research directions.

CHAPTER 2
CPPMIN using Branch and Cut

2.1 Introduction

Given a graph $G = (V, E)$ with edge weights c_e, and an integer $S < |V|$, the Clique Partition Problem with Minimum clique size requirement (CPPMIN) requires dividing the vertices V into subsets that each have at least S vertices. The objective is to minimize the total weight of the edges that have both endpoints in the same set.

We define a binary variable x_e, which takes the value 1 if edge e is within one subset and 0 if it is across two subsets. Our integer programming formulation for CPPMIN

$$\begin{aligned} \min \quad & \sum_{e \in E} c_e x_e \\ \text{s.t.} \quad & \sum_{e \in \delta(v)} x_e \geq S - 1 \qquad \forall v \in V \qquad (2.1) \\ & x \text{ is the incidence vector of a partition on } G \end{aligned}$$

where $\delta(v)$ denotes the set of edges incident to vertex v. Constraint (2.1) is called the size constraint. We will use $n = |V|$ to denote the number of vertices. With S given, we can define $k = \lfloor n/S \rfloor$, $r = n \bmod S$, $0 \leq r \leq S - 1$. Thus we have $n = kS + r$. We will explain later in section 2.1, what we mean by "x is the incidence vector of a partition on G", and rewrite this above formulation in a more concrete formulation as problem (CPPMIN).

Throughout this paper, we assume G is the complete graph on V. A non-complete graph can easily be converted to a problem on a complete graph by adding the missing edges in with a very big edge weight c_e, or with edge weight 0, depending on the application. We can also just add explicit constraints $x_e = 0$ for the missing edges e in a non-complete graph.

The CPPMIN problem is closely related to the classic partition problem that has no restrictions on the size or number of the partitions – the Clique Partition

Problem (CPP). CPP partitions the vertices V into subcliques, so as to minimize the total weight of the edges that have both endpoints in the same subcliques. Grötschel and Wakabayshi [23, 24] have provided a detailed description with several kinds of cutting planes, especially triangle constraints and 2-partition constraints. Their experiments showed that only using Triangle inequalities and 2-partition inequalities is good enough. A LP relaxation consisting of all triangle inequalities and some 2-partition inequalities turns out to be empirically very effective. In fact, only adding cutting planes of triangle inequalities were sufficient to prove the optimality of a feasible solution in most of the cases.

Variations of CPP have also been studied. One of them is the equipartition problem, which requires dividing the vertices into 2 clusters of equal size. When there is an odd number of vertices, they needed to be divided into to 2 clusters with size difference of one vertex. Conforti, Rao and Sassano [12] discussed the polyhedral theory for this problem. An extension from here is the k-way equipartition problem, which requires dividing the n points into k equally sized clusters. In this problem, the number of the vertices n has to be a multiple of the number of partitions k. This is equivalent to the constraint that each cluster has to be of size n/k. Mitchell [44] discussed the facial structure of the polytope associated with the problem and provided a branch-and-cut algorithm to solve the problem. He used this method to successfully solve a NFL scheduling problem in [47].

But in many instances the number n may not be a multiple of k, so a k-way equipartition is not possible. One way is to solve the k-way partition problem instead, which requires to divide the graph into no more than k clusters. Chopra and Rao [10] investigated this problem on a general graph. They don't assume a complete graph, so they can take advantage of the structure of the graph to be divided. They also discussed the opposite problem of requiring at least k clusters. A special case of the k-way clustering problem is the max cut problem, which has $k = 2$. Various approaches have been studied on this problem. A good survey can be found in Poljak and Tuza [54].

Another way to solve the problem when n is not a multiple of k is to relax the constraint on the size of the clusters and only require that each cluster size is not smaller than $S = \lfloor n/k \rfloor$. This leads to the problem in this paper: CPPMIN, a Clique Partition Problem with Minimum Clique Size requirement. It has applications in the micro-aggregation of statistical data [14], sports team scheduling[47] and many other areas.

It is natural to ask about the opposite problem : Clique Partition Problem with an upper bound on the cluster size. In fact, this can be converted into a k-way equipartition problem by adding dummy vertices. So the k-way equipartition problem and CPPMIN have covered all three kinds of constraints on cluster size for a clique partition problem.

Mehrotra and Trick [41] did research on Clustering problem with Knapsack capacity constraints. In their problem, each vertex has a weight. The total weight of the vertices in the same cluster cannot exceed a capacity. This is a generalized version of the problem requiring each cluster has to be smaller than a certain size. They used a branch and price method to solve the problem. Similarly, we can consider the opposite problem of requiring each cluster to be bigger than a certain weight. It can also be formulated as an integer programming problem in the similar fashion as CPPMIN:

$$\begin{array}{lll} \min & \sum_{e \in E} c_e x_e & \\ \text{s.t.} & \sum_{j \in V} w_j x_{ij} \geq CAP - w_i \qquad \forall i \in V & \\ & x \text{ is the incidence vector of a partition on } G & \end{array} \tag{2.2}$$

where w_i denotes the the weight of vertex i. CAP is the required minimum capacity. This is one of the future works that we would like to look into.

2.1.1 Notations and Definitions

We follow the notations we defined in chapter 1, especially we are going to use the following ones.

Given an undirected graph $G = (V, E)$, let V_1 and V_2 be two disjoint subsets of V, we define the following notation.

$$\begin{aligned}
\delta(V_1) &= \{uv \in E | u \in V_1, v \notin V_1\} \\
\delta(V_1, V_2) &= \{uv \in E | u \in V_1, v \in V_2\} \\
\delta(V_1, V_2, \ldots, V_k) &= \{uv \in E | u \in V_i, v \in V_j, i, j = 1, ..., k, i \neq j\} \\
E(V_1) &= \{uv \in E | u, v \in V_1\} \\
E(V_1, V_2) &= E(V_1) \cup E_(V_2) \\
E(V_1, V_2, \ldots, V_k) &= \cup_{i=1\ldots k} E(V_i)
\end{aligned}$$

A **clustering** of G is a division of V into $\Pi = \{V_1, V_2, \ldots, V_k\}$, where $V = \cup_{i=1..k} V_i$ and $V_i \cap V_j = \emptyset, \forall 1 \leq i < j \leq k$. $V_1, V_2, \ldots, V_k$ are the **components** in the clustering. When G is a complete graph, we also call these components subcliques. The edge set $E(\Pi) = E(V_1, V_2, \ldots, V_k)$ is called the **partition set** or simply **partition**, while $\delta(\Pi) = \delta(V_1, V_2, \ldots, V_k)$ is called the **multi-cut set** or simply **multi-cut**. Notice that $E(G) = \delta(\Pi) \cup E(\Pi)$, $\delta(\Pi) \cap E(\Pi) = \emptyset$. The partition and the multi-cut are just two ways of representing the same clustering problem. They are both used often in the literature. Next, we define the corresponding incidence vectors.

Definition 2.1.1. Given a graph $G = (V, E)$ and a clustering $\Pi = (V_1, V_2, \ldots, V_k)$ on G. Let $|E| = m$, $x_\Pi \in \{0, 1\}^m$ is called the **incidence vector of partition** Π if the entries in x_Π satisfy

$$x_e = \begin{cases} 1 & e \in E(\Pi) \\ 0 & \text{otherwise} \end{cases}$$

$y_\Pi \in \{0, 1\}^m$ is called the **incidence vector of multi-cut** Π if the entries in y_Π satisfy

$$y_e = \begin{cases} 1 & e \in \delta(\Pi) \\ 0 & \text{otherwise} \end{cases}$$

For any $U \subseteq V$, define $\mathbf{x}(\mathbf{U}) = \sum_{i,j\in U} x_{ij}$, $\mathbf{y}(\mathbf{U}) = \sum_{i,j\in U} y_{ij}$.

According to the above definition $x_\Pi + y_\Pi = e$, where e is a vector of ones.

In the following, we repeat a few definitions in integer programming. They will be used in the theorems and proofs in the later sections.

Definition 2.1.2. A set of points $x^1, \ldots, x^k \in I\!R^n$ is **affinely independent** if the unique solution to $\sum_{i=1}^{k} \lambda_i x^i = 0, \sum_{i=1}^{k} \lambda_i = 0$, is $\lambda_i = 0$ for all $i \in \{1, \ldots, k\}$.

Proposition 2.1.1. *A set of points $x^1, \ldots, x^k \in I\!R^n$ is affinely independent if and only if the vectors $x^2 - x^1, x^3 - x^1, \ldots, x^k - x^1$ are linearly independent.*

Definition 2.1.3. Let P be a polyhedron of dimension d. A face of dimension $d-1$ is a **facet**.

Definition 2.1.4. Given an inequality $\pi^T x \leq \pi_0$, the **support** of this inequality is defined as the set $J = \{j \in \{1, \ldots, n\} : \pi_j \neq 0\}$. If the x variables corresponds to the edges of a graph G, the **support graph** of this inequality is the subgraph of G, whose edges are indexed by the elements of the support J.

2.2 Polyhedral Theories

We define the corresponding polytope $R(G,S)$ and $\bar{R}(G,S)$ for problem (2.1) as follows:

$$\begin{aligned}
R(G,S) \;&:=\; conv\{x \in \{0,1\}^n : \sum_{e\in\delta(v)} x_e \geq S-1 \quad \forall v \in V \\
&\qquad \text{x is the incidence vector of a partition on G}\} \\
\bar{R}(G,S) \;&:=\; \{x \in [0,1]^n : \sum_{e\in\delta(v)} x_e \geq S-1 \quad \forall v \in V\}
\end{aligned}$$

Obviously $R \subseteq \bar{R}$. Once we solve the problem on R, we are done, but since we don't know a concrete description for R, we will start from $\bar{R}$.

We also need to define some other related polytopes:

$$\begin{aligned}
P(G) &:= conv\{x \in \{0,1\}^n : && x \text{ is the incidence vector of a clique partitioning} \\
& && \text{of Graph G }\} \\
P_{BC}(G) &:= conv\{y \in \{0,1\}^n : && y \text{ is the incidence vector of a 2-way cut of Graph G}\} \\
P_{EC}(G) &:= conv\{y \in \{0,1\}^n : && y \text{ is the incidence vector of an equicut of Graph G}\} \\
P_{EP}(G) &:= conv\{x \in \{0,1\}^n : && x \text{ is the incidence vector of an equipartition} \\
& && \text{of Graph G}\}
\end{aligned}$$

2.2.1 Polyhedral Theory for $P(G)$, $P_{BC}(G)$, $P_{EC}(G)$ and $P_{EP}(G)$

We are going to summarize some results for the above polytopes. Since Clique Partition Problem (CPP) is the basic problem all the other variations are based on, we'll start with its corresponding polytope $P(G)$. Grötschel and Wakabayashi [23, 24] described a simplex-based cutting plane algorithm for CPP. The most important facets they used there are Triangle Inequalities and 2-partition equalities.

Theorem 2.2.1. *([24]) $P(G)$ is full dimensional, i.e., $dim(P(G)) = |E|$.*

Theorem 2.2.2. *([24], Theorem 3.1) Each **Triangle Inequality***

$$x_{ij} + x_{il} - x_{jl} \leq 1 \quad \forall 0 \leq i \neq j \neq l \leq |V| \tag{2.3}$$

defines a facet of the clique partitioning polytope.

In fact the triangle inequalities with binary constraints would completely describe the incidence vector of the clique partition problem of a complete graph. In other words, the solution of (2.4) are exactly the incidence vectors for the CPP of a complete graph.

$$\begin{aligned}
\min \quad & \sum_{e \in E} c_e x_e \\
\text{s.t.} \quad & x_{ij} + x_{il} - x_{jl} \leq 1 && \forall i,j,l \quad 0 \leq i \neq j \neq l \leq n \qquad (2.4) \\
& x_{ij} \in \{0,1\} && \forall i,j \quad 1 \leq i < j \leq n
\end{aligned}$$

Similarly, we can rewrite our initial formulation for CPPMIN, i.e., problem (2.1), in a more concrete form as the following:

$$\begin{array}{llrr}
\min & \displaystyle\sum_{i,j\in E, i\neq j} c_{ij}x_{ij} & & \\
\text{s.t.} & \displaystyle\sum_{j\in E, j\neq i} x_{ij} \geq S-1 & \forall i \in V & (2.5) \\
& x_{ij} + x_{il} - x_{jl} \leq 1 & \forall i,j,l \quad 0 \leq i \neq j \neq l \leq n & (2.6) \\
& x_{ij} \in \{0,1\} & \forall i,j \quad 1 \leq i < j \leq n &
\end{array}$$

Theorem 2.2.3. *([24], Theorem 4.1)For every nonempty disjoint subsets $U, W \subseteq V$, the* ***2-partition inequality***

$$x(U,W) - x(U) - x(W) \leq min\{|U|, |W|\} \tag{2.7}$$

defines a facet of the clique partitioning polytope, provided $|U| \neq |W|$

Their experiments showed that these two kinds of constraints, especially the triangle constraints give a good approximation for the polytope.

The results below for $P_{BC}(G)$ and $P_{EC}(G)$ are from Barahona and Mahjoub [4], Barahona, Grotschel and Mahjoub [2], Conforti, Rao and Sassano[12]. We will use them for our later proof.

Theorem 2.2.4. *([2]) $P_{BC}(G)$ is full dimensional, i.e., $dim(P_{BC}(G)) = |E|$.*

Theorem 2.2.5. *([12], Lemma 3.4) The dimension of $P_{EC}(K_{2p+1})$ is $\binom{2p+1}{2} - 1$.*

Theorem 2.2.6. *([12], Lemma 3.5) The dimension of $P_{EC}(K_{2p})$ is $\binom{2p}{2} - 2p$.*

Corollary 2.2.7. *The dimension of equipartition polytope $P_{EP}(K_{2p+1})$ is $\binom{2p+1}{2} - 1$. The dimension of equipartition polytope $P_{EP}(K_{2p})$ is $\binom{2p}{2} - 2p$.*

Proof. $\forall y \in P_{EC}$, let $x = e - y$, then $x \in P_{EP}$. So every set of affinely independent vectors in P_{EC} corresponds to a set of affinely independent vectors in P_{EP}, thus the corollary follows. □

2.2.2 Dimension for $R(G,S)$

Theorem 2.2.8. *The dimension of CPPMIN polytope $R(G,S)$ is*

(i) If $S < \frac{n}{2}$, then $dim(R(G,S)) = \binom{n}{2}$.

(ii) If $S = \frac{n}{2}$, then $dim(R(G,S)) = \binom{n}{2} - n$.

(iii) If $S > \frac{n}{2}$, then $dim(R(G,S)) = 0$.

Proof. We are going to prove the theorem according to the three cases.

(i) for $S < \frac{n}{2}$, consider 2 cases:

(a) $n = kS + r$ **is odd**.

Suppose $n = kS + r = 2p + 1$, consider the hyperplane corresponding to an equipartition $H_{ep} = \{x \in R^{\binom{2p+1}{2}} : x(G) = \binom{2p+1}{2} - p(p+1)\}$. From Corollary 2.2.7, we have $dim(H_{ep}) = \binom{2p+1}{2} - 1$. Since $H_{ep} \subset R(G,S)$, $e \in R(G,S)$, $e \notin H_{ep}$, we get $dim(R(G,S)) = \binom{2p+1}{2} = \binom{n}{2}$.

(b) $n = kS + r$ **is even**.

Suppose $kS + r = 2p$, we will consider the case when $S = p - 1$. We can embed the graph $G = K_{2p}$ into a complete graph K_{2p+1} by adding one more vertex v_{2p+1}. Let $m = \binom{2p}{2}$, $m' = \binom{2p+1}{2}$. From (a), we know $dim(R(K_{2p+1}, p)) = \binom{2p+1}{2} = m'$, so there exist $m' + 1$ affinely independent incidence vectors $x^1, \ldots, x^{m'}, x^{m'+1} \in R(K_{2p+1}, p)$. They can be represented as the column vectors in a $m' \times (m' + 1)$ matrix A'. According to the definition of affine independence, system $\begin{bmatrix} e^T \\ A' \end{bmatrix} \lambda = 0$ has a unique solution $\lambda = 0$. So $rank(\begin{bmatrix} e^T \\ A' \end{bmatrix}) = m' + 1$. Now we remove the $2p$ rows in A' that corresponds to the $2p$ edges incident to vertex v_{2p+1}. The remaining matrix $\begin{bmatrix} e^T \\ A \end{bmatrix}$ has full row rank $\binom{2p+1}{2} + 1 - 2p = \binom{2p}{2} + 1$. We can pick $\binom{2p}{2} + 1$ linearly independent column vectors from $\begin{bmatrix} e^T \\ A \end{bmatrix}$ to get $\begin{bmatrix} e^T \\ B \end{bmatrix}$, now $\begin{bmatrix} e^T \\ B \end{bmatrix} \mu = 0$ implies $\mu = 0$, so the $\binom{2p}{2} + 1$ column vectors in B are affinely independent, and they correspond to the incidence vectors in $R(K_{2p}, p-1)$, so we have $dim(R(K_{2p}, p-1)) \geq \binom{2p}{2}$. Since $\binom{2p}{2}$ is already full dimension, we have $dim(R(K_{2p}, p-1)) = \binom{2p}{2} = \binom{n}{2}$.

In the case that $S \leq p-1$, $R(G,S) \supseteq R(G,p-1)$, so $dim(R(G,S)) = \binom{n}{2}$.

(ii) For $S = \frac{n}{2}$, this can only happen when n is an even number, it then becomes an equipartition problem, so $dim(R(G,S)) = \binom{2S}{2} - 2S$ from Conforti and Rao[12],

(iii) For $S > \frac{n}{2}$, only one solution is possible, which is to put all vertices in the same subset. The polytope $R(G,S)$ degenerates to a single point, so $dim(R(G,S)) = 0$.

□

2.3 Cutting Planes

2.3.1 Bound Constraints

Before introducing the theorems to establish the facets for CPPMIN, we'll introduce two lemmas first. Lemma 2.3.1 is almost exactly the same as the lemma 3.2 from Chopra and Rao [10], except that we need to make sure here that every component in the partition satisfies the minimum size requirement. The proof, which we will skip here, also follows the same as in Chopra and Rao[10].

Lemma 2.3.1. *Let $ax \geq b$ be any valid inequality with respect to R(G,S), let j be any vertex in G. Consider the following partitions of G,*

$$\Pi_1 = N_1, N_2 \cup j, N_3, \ldots, N_r,$$
$$\Pi_2 = N_1 \cup j, N_2, N_3, \ldots, N_r,$$

If $\Pi_1, \Pi_2 \in R(G,S)$ and the incidence vector for Π_1 and Π_2 both satisfy the inequality $ax \geq b$ at equality, then we have $a(N_1, j) = a(N_2, j)$, where $a(N,j) = \sum\limits_{v \in N} a(v,j)$

CPPMIN polytope R(G,S) is full dimension implies that each facet-defining inequality is unique up to multiplication by a constant (cf. [51] Theorem 3.5). We summarize in the next lemma another result on the relationship between the constraints of CPPMIN.

Lemma 2.3.2. *Suppose $a^T x \leq b$ and $g^T x \leq h$ are both valid constraints for CPP-MIN polytope $R(G,S)$, s.t. $\{x | a^T x = b, x \in R(G,S)\} \subseteq \{x | g^T x = h, x \in R(G,S)\}$, then $a(i,j) = 0$ implies $g(i,j) = 0$ under the following condition: there exist a feasible partition incidence vector $\bar{x}$ s.t.*

- $a^T \bar{x} = b$
- $\bar{x}_{ij} = 1$;
- *The cluster containing vertex i and j in partition $\bar{x}$, called U, satisfies $|U| \geq 2S+1$, $E(U) \subseteq E^c := \{e \in E : a_e = 0\}$.*

Proof. $\forall v, u, w \in U$, $N_1 \subseteq U$, $N_2 \subseteq U$ s.t. $|N_1| \geq S$,$|N_2| \geq S$, $N_1 \cap N_2 = \emptyset$, $v \in U - N_1 - N_2$, $u \in N_1$, $w \in N_2$. Without loss of generality we can rearrange $\bar{x}$ s.t. $a^T \bar{x} = b$, $\bar{x}(e) = 1, \forall e \in E(N_1 + v)$; $\bar{x}(e) = 1, \forall e \in E(N_2)$; $\bar{x}(e) = 0, \forall e \in \delta(N_1 + v, N_2)$; we can move v from the cluster of N_1 to the cluster of N_2 to construct another solution $\tilde{x}$ s.t. $a^T \tilde{x} = b$, and $\tilde{x}(e) = 1, \forall e \in E(N_1)$; $\tilde{x}(e) = 1, \forall e \in E(N_2 + v)$; $\tilde{x}(e) = 0, \forall e \in \delta(N_1, N_2 + v)$.

Since $E(U) \subseteq E^c$, both $\bar{x}$ and $\tilde{x}$ satisfy $a^T x = b$, therefore both satisfy $g^T x = h$ too.

From lemma 2.3.1, we get $g(v, N_1) = g(v, N_2)$, which can be written equivalently as

$$g(v, N_1 - u) + g(v, u) = g(v, N_2 - w) + g(v, w) \tag{2.8}$$

switch u and w between N_1 and N_2, repeat the above process, we get

$$g(v, N_1 - u) + g(v, w) = g(v, N_2 - w) + g(v, u) \tag{2.9}$$

(2.8)-(2.9) gives us

$$g(v, w) = g(v, u) =: \alpha \qquad \forall u, v, w \in U \tag{2.10}$$

Now consider another incidence vector $\hat{x}$ with $N = N_1 + N_2 + v$ all in one cluster, i.e. $\hat{x}(e) = 1, \forall e \in E(N)$. Comparing with incidence vector $\bar{x}$, we get

$$\begin{aligned} \sum_{e \in \delta(N_1+v, N_2)} g_e = 0 \\ \Rightarrow \quad & (|N_1| + 1)|N_2|\alpha = 0 \\ \Rightarrow \quad & \alpha = 0 \\ \Rightarrow \quad & g(u, v) = 0 \qquad \forall u, v \in U \end{aligned}$$

Especially we have $g(i, j) = 0$. □

Theorem 2.3.3. *$x_{ij} \geq 0$ is a facet for $R(G, S)$ when $n \geq 3S + 1$.*

Proof. This follows easily from Lemma 2.3.2, since for every edge $(k, l) \neq (i, j)$, there exist a feasible solution with $x_{ij} = 0$ and $x_{kl} = 1$, and $U = V - i$ satisfy the other condition in the lemma, thus any constraints $g^T x \geq h$ that implies $x_{ij} \geq 0$ must have $g_{kl} = 0, \forall k, l \neq i, j$, therefore $g^T x \geq h$ became the same as $x_{ij} \geq 0$. □

Note that $x_{ij} \leq 1$ is not a facet because it is implied by summing up 2 triangle constraints: $x_{ij} + x_{ik} - x_{jk} \leq 1$, $x_{ij} + x_{jk} - x_{ik} \leq 1$.

2.3.2 Triangle Inequalities and 2-partition Inequalities

Triangle Inequalities and 2-partition Inequalities are inherited directly from CPP. Since CPPMIN is just a special case of CPP, these two kind of constraints are also valid for CPPMIN. Using lemma 2.3.2, we can show the following result for $R(G, S)$.

Theorem 2.3.4. *2-partition constraint (2.7): $x(U, W) - x(U) - x(W) \leq min\{|U|, |W|\}$ is a facet for $|U| > |W|$, provided $|V| > S|U| + 2S + 1$.*

Proof. Suppose $|U| = p, |W| = q$, we can write $U = \{u_l, l = 1..p\}$, $W = \{w_l, l = 1..q\}$. Consider any constraint $g^T x = h$ such that $\{x | x$ satisfy (2.7) at equality, $x \in R(G, S)\} \subseteq \{x | g^T x = h, x \in R(G, S)\}$,

(i) First, we would show that $g(i, j) = 0, \forall (i, j) \in E - E(U \cup W)$.

Consider the partition $\Pi = \{N_1, N_2, ..., N_q, ..., N_p, N_{p+1}\}$ s.t. $u_l \in N_l$ for $l = 1..p$, $w_l \in N_l$ for $l = 1..q$, $N_{p+1} = V - \sum_{l=1}^{p} N_l$. Partition Π satisfies (2.7) at equality. We use x to denote its corresponding incidence vector.

$\forall i \in U$, $j \in V - U - W$, we can rearrange Π to also satisfy $j \in N_{\bar{l}}, |N_{\bar{l}}| \geq 2S+1$ for $\bar{l}$ s.t. $q < \bar{l} \leq p$ and $i \in N_{\bar{l}}$. Now the three conditions in Lemma 2.3.2 are satisfied by Π, namely, $a^T x = b$, $x_{ij} = 1$, $N_{\bar{l}} \subseteq E^c$, and $|N_{\bar{l}}| > 2S + 1$. So $g(i, j) = 0$.

$\forall i \in V - U - W, j \in V - U - W$, we can rearrange Π to also satisfy $i \in N_{p+1}, j \in N_{p+1}, |N_{p+1}| \geq 2S + 1$. Again Lemma 2.3.2 is satisfied on this partition. So $g(i, j) = 0$.

Now we only need to show $g(i, j) = 0, \forall i \in W, j \in V - U - W$. We can rearrange Π to satisfy $j \in N_{\bar{l}}, |N_{\bar{l}}| \geq S + 1$ for $\bar{l}$ s.t. $i \in N_{\bar{l}}$. We call this partition $\bar{\Pi}$. Since $1 \leq \bar{l} \leq q$, $N_{\bar{l}} \cap U \neq \emptyset$. Let $v = N_{\bar{l}} \cap U$.

We can get another partition $\tilde{\Pi}$ by switching vertex j from $N_{\bar{l}}$ to N_{p+1} in $\bar{\Pi}$. Both $\bar{\Pi}$ and $\tilde{\Pi}$ satisfy constraint (2.7) at equality, so we have

$$\begin{aligned} & g(j, N_{\bar{l}} - j) = g(j, N_{p+1}) \\ \Rightarrow \quad & g(i, j) + g(v, j) + g(j, N_{\bar{l}} - j - i - v) = g(j, N_{p+1}) \end{aligned}$$

Along with

$$\begin{aligned} g(v, j) &= 0 && \text{since } v \in U \\ g(j, N_{\bar{l}} - j - i - v) &= 0 && \text{since } N_{\bar{l}} - j - i - v \subseteq V - U - W \\ g(j, N_{p+1}) &= 0 && \text{since } N_{p+1} \subseteq V - U - W \end{aligned}$$

We get $g(i, j) = 0, \forall i \in W, j \in V - U - W$.

Thus we have showed that $g(i, j) = 0$, $\forall (i, j) \in E - E(U \cup W)$.

(ii) In this part, we are going to show that $g(u, v) = g(w, w') = -g(u, w) = -\alpha$, $\forall u, v \in U, w, w' \in W$.

$\forall u, v \in U, w, w' \in W$, without loss of generality, we can assume $u = u_1, v =$

$u_{q+1}, w = w_1, w' = w_2$. Again we can construct a partition Π in the same way as in (i), $\Pi = \{N_1, N_2, ..., N_q, ..., N_p, N_{p+1}\}$ s.t. $u_l \in N_l$ for $l = 1..p$, $w_l \in N_l$ for $l = 1..q$, $N_{p+1} = V - \sum_{l=1}^{p} N_l$. Partition Π satisfies (2.7) at equality. With x denoting the incidence vector of Π, we have $g^T x = h$.

Now switch $u = u_1$ and $v = u_{q+1}$ between N_1 and N_{q+1}, we get partition $\bar{\Pi}$, represented by incidence vector $\bar{x}$, which also satisfies constraint (2.7), thus satisfying $g^T \bar{x} = h$. Comparing $g^T x = h$ and $g^T \bar{x} = h$, along with the result from (i), we get $g(u, w) = g(v, w)$. Since u, v, w are picked randomly from U and W, we have that for any fixed $w \in W$, $g(u, w) = \alpha_w$ is a constant $\forall u \in U$.

Modify partition Π by combining two clusters N_1 and N_{q+1}, we get partition $\tilde{\Pi}$, represented by incidence vector $\tilde{x}$, which also satisfies (2.7) at equality, so $g^T \tilde{x} = h$. Comparing $g^T x = h$ and $g^T \tilde{x} = h$, we get $g(u, v) = -g(v, w) = -\alpha_w$.

Since w is picked randomly from W, we have $g(u, v) = -\alpha, g(u, w) = \alpha, \forall u, v \in U, w \in W$.

Finally, we need to show $g(w, w') = -\alpha$. Since $w = w_1, w' = w_2$, combining clusters N_1 and N_2 in partition Π, we get partition $\hat{\Pi}$, represented by incidence vector $\hat{x}$. $\hat{x}$ satisfies constraint (2.7), thus satisfying $g^T \hat{x} = h$. Comparing with $g^T x = h$, we have $g(u, v) + g(w, w') + g(u, w') + g(v, w) = 0$. Since $g(u, v) = -\alpha, g(u, w') = g(v, w) = \alpha$, we have $g(w, w') = -\alpha \ \forall w, w' \in W$.

Summarize the above results, we get that $g(i, j) = 0$, $\forall (i, j) \in E - E(U \cup W)$; $g(u, v) = g(w, w') = -g(u, w) = -\alpha$, $\forall u, v \in U, w, w' \in W$. So we conclude that $g^T x = h$ is equivalent to the constraint (2.7). Therefore we have proved that (2.7) is a facet.

$\square$

Since triangle constraint is just a special case of the 2-partition constraints when $min\{|U|, |W|\} = 1$, the above theorem also applies to that triangle constraint.

2.3.3 Pigeon Inequalities

From now on, we only consider the case when $r > 0$, i.e. when n is not a multiple of S. When $r = 0$, the new constraints we are going to introduce are still valid, but they may not be facets under the conditions given.

2.3.3.1 Pigeon Inequalities for k-way Partition $P1(G, k)$

To consider the pigeon constraint for CPPMIN problem, we would like to introduce the pigeon constraint for the k-way partition problem first. Given a connected graph $G = (V, E)$ with edge weights c_e, the k-way partition problem is to partition the node set V into no more than k nonempty subsets so as to minimize the total weight of the edges with end points in two different subsets. Chopra and Rao [10] investigated this problem, and give the following Integer Programming formulation. Let $\Pi = (N_i, i = 1, 2, \dots m)$ denotes an $m - partition$ of the node set V, i.e., a partition that has m clusters. Each subset N_i in Π is assigned an index $t_i, 1 \leq t_i \leq m$, which identifies the subset that nodes in N_i belong to. For partition Π, they defined an incidence vector (x, z) where

$$x_e = \begin{cases} 1 & \text{if edge } e \in E_\Pi \\ 0 & \text{otherwise} \end{cases} \qquad z_{it} = \begin{cases} 1 & \text{if node } i \in N_t \\ 0 & \text{otherwise} \end{cases}$$

Notice that z variables are introduced here to differentiate different cliques. It is introduced to take advantage of the graph structure of G, since G is not assumed to be complete in [10].

The corresponding polytope is defined as

$$P1(G, k) = conv\{(x, z) | (x, z) \text{ is the incidence vector of an } m\text{-partition for } m \leq k\}$$

Theorem 2.3.5. *([10], Theorem 3.1.1) Let $Q = (V(Q), E(Q)) \subseteq G$ be a clique of size $p = tk + q$ where $t \geq 1$ and $0 \leq q < k$. The pigeon inequality*

$$\sum_{e \in E(Q)} X_e \geq \frac{1}{2}t(t-1)(k-q) + \frac{1}{2}t(t+1)q \tag{2.11}$$

is facet defining for P1 if and only if $1 \leq q < k$, i.e. p is not an integer multiple of k.

2.3.3.2 Pigeon Inequality for $R(G, S)$

Now consider CPPMIN: S is the minimum size for each cluster, so the graph can be divided into no more than $k = \lfloor \frac{n}{S} \rfloor$ clusters. Thus any solution to our problem can be represented as a valid solution to Chopra and Rao's k-way partition problem, so the pigeon constraints are valid for our problem too. We will give a necessary and sufficient condition for pigeon constraint (2.11) to be a facet for $R(G, S)$.

As shown in Figure 2.1, a partition Π holding at equality for equation (2.11) would have the following structure: there are totally k components in the partition, $\Pi = (N_1, N_2, \ldots, N_k)$, $k - q$ of them each has exactly t vertices from Q, i.e., $|N_l \cap V(Q)| = t$ for $l = 1, 2, \ldots, k - q$. The other q components each has exactly $t + 1$ vertices from Q, i.e., $|N_l \cap V(Q)| = t + 1$ for $l = k - q + 1, \ldots, k$.

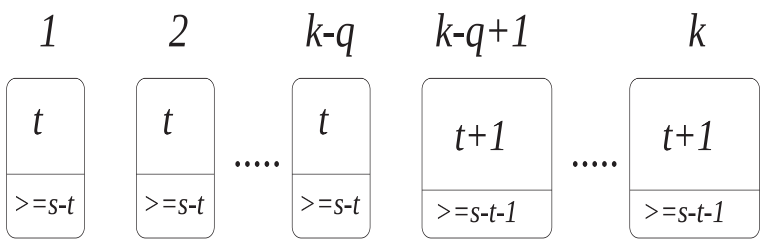

Figure 2.1: Illustration for Pigeon Inequalities

Theorem 2.3.6. *Given a graph $G = (V, E)$ and an integer S, let $k = \lfloor \frac{n}{S} \rfloor$, $r = n$ mod S, thus $n = kS + r$. Let $Q = (V(Q), E(Q)) \subseteq G$ be a clique of size $p = tk + q$ where $t \geq 1$ and $1 \leq q < k$. Consider the pigeon constraint* (2.11),

(i) When $r = 0$, pigeon constraint (2.11) *is valid for $R(G, S)$, but not facet defining.*

(ii) When $r = 1$, pigeon constraint (2.11) *is facet defining for $R(G, S)$ if and only if $Q = V$.*

(iii) When $r > 1$, pigeon constraint (2.11) *is always facet-defining for $R(G, S)$.*

Proof. The proof goes according to the three cases.

(i) when $r = 0$,

A partition Π satisfy Pigeon constraint (2.11) at equality implies that there are exactly k clusters, with $r = 0$, each cluster would have exactly S vertices. Thus the size constraints (2.1) hold at equality. In other words, the pigeon constraints are implied by the size constraints (2.1). So no pigeon constraints are facets.

(ii) When $r = 1$,

Any feasible partition Π that satisfy constraint (2.11) at equality for a $p < n$ must also satisfy at equality the constraint (2.11) for $p = n$, which is written as (2.12),

$$\sum_{e \in E(V)} x_e \geq (k-r)\binom{S}{2} + r\binom{S+1}{2} \tag{2.12}$$

because it has to be of the structure with $k-1$ clusters of size S and 1 cluster of size $S+1$, i.e., $\Pi = (N_1, N_2, \ldots, N_k)$, where $|N_i| = S$ for $i = 1, \ldots, k-1$, and $|N_k| = S + 1$. Thus constraints (2.11) with $p < n$ is not a facet. It will be shown in Part (iii) (b) along with the case when $r > 1$ that Constraint (2.12) is a facet.

(iii) when $r > 1$,

We are going to prove the theorem using the following property for facets: For a full dimensional polytope P, a valid constraint $a^T x \leq \alpha$ is a facet if and only if any valid constraint of P $bx \leq \beta$ satisfying $\{x \in P | a^T x = \alpha\} \subseteq \{x \in P | b^T x = \beta\}$ must also satisfy $\frac{a_i}{b_i} = \frac{\alpha}{\beta}, \forall i$. This comes directly from the definition of a facet.

Consider any valid inequality $b^T x \leq \beta$ s.t. $\{x \in P | \text{x satisfies (2.11) at equality}\} \subseteq \{x | bx = \beta\}$. To make the following proof clearer, we rewrite the biggest pigeon constraint (2.12) as the following,

$$\sum_{e \in E(Q)} x_e + \sum_{e \in \delta(Q)} x_e + \sum_{e \in E(V-Q)} x_e \geq (k-r)\binom{S}{2} + r\binom{S+1}{2} \tag{2.13}$$

$\forall v \in V$, we can construct a partition $\Pi = (N_1, N_2, \ldots, N_k)$, where $|N_l| \geq S$, for $l = 1, 2, \ldots, k$; $|N_l \cap V(Q)| = t$, for $l = 1, 2, \ldots, k-q$; $|N_l \cap V(Q)| = t+1$, for $l = k-q+1, \ldots, k$; and $v \in N_i, |N_i| \geq S+1$, $|N_i \cap V(Q)| = t+1$.

(a) By moving vertex v from V_i to V_1, we obtain partition $\Pi' = (N_1', N_2, N_3, \ldots, N_i', \ldots, N_k)$, where $N_1' = N_1 \cup \{v\}$, $N_i' = N_i - \{v\}$.

No matter whether $v \in V(Q)$ or not, both Π and Π' satisfy pigeon constraint (2.11) at equality. From lemma 2.3.1, we have

$$b(N_1, v) = b(N_i - v, v) \tag{2.14}$$

(b) Now suppose $v \in V(Q)$, we are going to show $b(i_1, v) = b(i_2, v) =: \delta_v$, $\forall i_1, i_2, v \in V(Q)$. This is exactly the same as the proof in [10]. For the completeness of this proof, we put it down here.

We can assume the partition Π was constructed so that $i_1 \in N_1 \cap V(Q), i_2 \in N_i \cap V(Q)$. Switching i_1 and i_2, we get another partition $\bar{\Pi} = (\bar{N}_1, N_2, \ldots, \bar{N}_i, \ldots, N_k)$ where $\bar{N}_1 = N_1 \cup i_2 - i_1$, $\bar{N}_i = N_i \cup i_1 - i_2$. Now repeat the procedure (a) with partition $\bar{\Pi}$, we'll get

$$b(\bar{N}_1, v) = b(\bar{N}_i - v, v) \tag{2.15}$$

In other words, we move v from $\bar{N}_i$ to $\bar{N}_1$ in $\bar{\Pi}$, to obtain partition $\hat{\Pi} = (\bar{N}_1 + v, N_2, \ldots, \bar{N}_i - v, \ldots, N_k)$. Since both $\bar{\Pi}$ and $\hat{\Pi}$ satisfy pigeon constraint at equality, from lemma 2.3.1, we obtain (2.15).

Rewrite (2.14) and (2.15), we have

$$b(N_1 - i_1, v) + b(i_1, v) = b(N_i - v - i_2, v) + b(i_2, v) \tag{2.16}$$

$$b(\bar{N}_1 - i_2, v) + b(i_2, v) = b(\bar{N}_i - v - i_1, v) + b(i_1, v) \tag{2.17}$$

(2.16) – (2.17) give us

$$b(i_1, v) = b(i_2, v) = \delta_v \qquad \forall i_1, i_2 \in Q \tag{2.18}$$

Since i_1, i_2, v are chosen arbitrarily, we have

$$b(i, v) = \delta_v \qquad \forall i, v \in V(Q) \tag{2.19}$$

This actually implies what we have stated in part (ii) of the theorem, i.e., when $Q = V$, the biggest pigeon constraint (2.11) is a facet.

(c) From now on, in this proof, we assume $v \in V - V(Q)$. Let $i_1 \in N_1 \cap V(Q)$, $i_2 \in N_i \cap V(Q)$.

Repeat the procedure as in (b) on partition Π, we get

$$b(i_1, v) = b(i_2, v)$$

Again, since v, i_1, i_2 are chosen arbitrarily, we have

$$b(i, v) = \alpha_v, \, \forall i \in V(Q), v \in V - V(Q) \tag{2.20}$$

(2.20) holds for every $v \in V - V(Q)$.

(d) When $|V - V(Q)| \geq 3$, without loss of generality, let $j_1, j_2 \in V - V(Q)$, $j_1 \in N_1$, $j_2 \in N_i$.

Switching j_1 and j_2, we get $\tilde{\Pi} = (\tilde{N_1}, N_2, \ldots, \tilde{N_i}, \ldots, N_k)$ where $\tilde{N_1} = N_1 - j_1 \cup j_2$, $\tilde{N_i} = N_i - j_2 \cup j_1$.

Repeat the procedure in (a) with $\tilde{\Pi}$ we have

$$b(\tilde{N_1}, v) = b(\tilde{N_i} - v, v) \tag{2.21}$$

Rewrite these two equations (2.14) and (2.21), we have

$$b(N_1 - j_1, v) + b(j_1, v) = b(N_i - r - j_2, v) + b(j_2, v) \tag{2.22}$$

$$b(\tilde{N_1} - j_2, v) + b(j_2, v) = b(\tilde{N_i} - v - j_1, v) + b(j_1, v) \tag{2.23}$$

(2.22) – (2.23) gives us

$$b(j_1, v) = b(j_2, v)$$

Again since j_1, j_2, v are chosen arbitrarily, we have

$$b(j, v) = \beta_v, \forall j \in V - V(Q), v \in V - V(Q) \tag{2.24}$$

When $|V - V(Q)| < 3$, (2.24) is obviously true.

(e) From equation (2.14) in part (a), we have

$$\begin{aligned} & b(N_1, v) = b(N_i - v, v) \\ \Rightarrow \quad & t\alpha_v + (|N_1| - t)\beta_v = (t+1)\alpha_v + (|N_i| - t - 1)\beta_v \\ \Rightarrow \quad & \alpha_v = (|N_1| - |N_i| + 1)\beta_v, \forall v \in V - V(Q) \end{aligned} \tag{2.25}$$

since $r > 1, v \notin Q$, We can construct at least two valid partition Π, one with $|N_1| = |N_i|$ and the other with $|N_1| + 1 = |N_i|$. To have equation (2.25) hold for both of these two cases, we have to have

$$\alpha_v = \beta_v = 0 \tag{2.26}$$

(f) Having (2.19), (2.20), (2.24), (2.26) together, we have proved that any constraint $bx \geq \beta$ dominating pigeon constraint (2.11) has the same coefficient for the edges within Q and 0 as the coefficient for all other edges. i.e. $bx \geq \beta$ is of the form

$$\sum_{e \in E(Q)} x_e \geq \beta. \tag{2.27}$$

Thus we have shown that (2.11) is a facet when $r > 1$.

□

We give two examples to illustrate cases (ii) and (iii) in the theorem.

Example I: A pigeon constraint that's not a facet, as shown in Figure 2.2. Consider a CPPMIN on $G = K_7$, with $S = 3$. Here $n = 7, S = 3, k = 2, r = 1$. Let W be any $q = 3$ vertices from V. The pigeon constraint on W: $x(W) \geq 1$ is not a facet because any solution satisfying this inequality at equality will satisfy the biggest pigeon constraint, ie. the pigeon constraint involving all edges, $X(G) \geq 9$ at equality. The biggest pigeon constraint is a facet. In the figure, $W = \{v_3, v_4, v_7\}$, the blue dash line is the support graph for the pigeon constraint on it. The black lines gives one feasible solution to the CPPMIN problem.

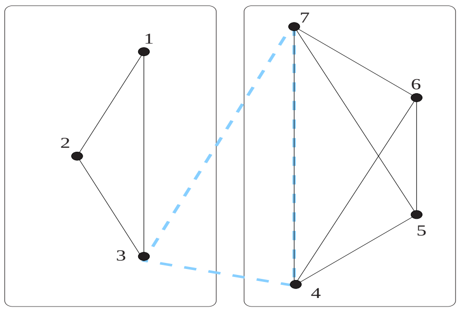

Figure 2.2: Example I - Pigeon Constraint on K_7, with $q = 3$

Example II: A pigeon constraint that is a facet, as shown in Figure 2.3. Consider complete graph $G = K_8$, here $n = 8, S = 3, k = 2, r = 2$. Let W be any $q = 4$ vertices from V. The pigeon constraint on W: $x(W) \geq 2$ is in fact a facet. For a problem of this size, it is possible to verify the result by enumerating all the solutions that satisfy this inequality at equality. In the figure, $W = \{v_2, v_3, v_7, v_8\}$, the blue dash line is the support graph for the pigeon constraint on it. The black lines gives one feasible solution to the CPPMIN problem, which satisfy the pigeon constraint on W: $x(W) \geq 2$ at equality, but does not satisfy the pigeon constraint on V, i.e., equation (2.12) at equality.

2.3.3.3 How to Find Violated Pigeon Constraints

Consider the complete graph G in the CPPMIN problem, change the edge weight into the corresponding x value in the LP solution. The problem of finding violated pigeon constraints of certain size p is the same as the problem of finding a clique of size p on this weighted graph that has the total weight smaller than the

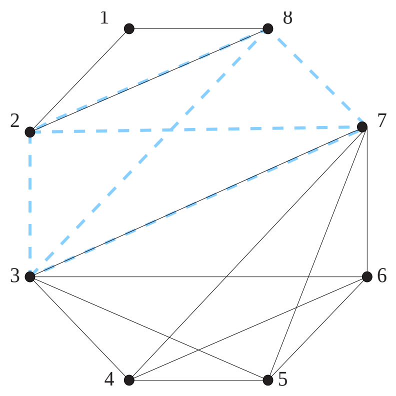

Figure 2.3: Example II - Pigeon Constraint on K_8, with $q = 4$

minimum value required in the pigeon constraint. If we know the minimum weight clique of size p, then we can find the violated constraint or prove that there is no pigeon constraint of size p violated. However, in general, the problem of finding minimum weight clique of size p is a NP-hard problem, except for some specific value of p, such as $p = n$, $p = n - 1$, $p = 1$, $p = 2$. Thus in our experiment, we will look for only 3 kinds of pigeon constraint as follows:

- when $p = n = |V|$, there is only one pigeon constraint. We simply added it into the initial problem.

- when $p = n-1$, there are n such pigeon constraints, and they are easy to check. We added the violated ones in the cutting plane step. But in our experiment, this kind of constraint is seldom violated, consequently seldom added.

- When $p = k + 1$, there are $\binom{n}{k+1}$ such pigeon constraints. They are time-consuming to enumerate, and it is hard to find the violated ones. but these constraints are not as dense as the previous ones, and they are often violated as shown from the experiments.

  We modified a greedy algorithm for maximum dispersion suggested by Hassin [27] to look for the minimum clique of size p. Our modified algorithm is shown in Algorithm 1. Note that one reason that we could adopt this method successfully is based on the assumption that the LP solutions satisfy the triangle inequalities. For this reason and also because pigeon constraints are comparably more dense, we add in this kind of pigeon constraints only when no triangle and 2-way constraints are added in the current iteration. Through the experiments we see that the majority of the added pigeon constraints are of this kind.

Algorithm 1 Find a small weight clique of size $k+1$

```
Input:
  $k$ - number of clusters
  $lpx$ - the linear program solution
Output:
  $support$ - a small weight clique of size $(k+1)$
  $sweight$ - the total weight of $support$
Steps:
  $support = \emptyset$
  for $i = 1$ to $(k+1)/2$ do
    find $w(v_i, v_j) = min\{w(v_i, v_j) | (v_i, v_j) \in E\}$
    $support = support + \{v_i, v_j\}$
    find all the vertices $v$ in $E$ that has $lpx(v, v_i) > 0.5$ or $lpx(v, v_j) > 0.5$
    delete from $E$ the vertices found in the last step.
  end for
  if $k$ is odd, add to $support$ an arbitrary vertex.
```

2.3.4 Flower Constraints

2.3.4.1 Flower Constraints for $R(G, S)$

Lemma 2.3.7. *Given G=(V,E), let $W = (V(W), E(W)) \subseteq V$ be a clique of size $w = S + q$, $q < S$, The flower inequality*

$$\sum_{e \in E(W)} x_e + \sum_{e \in \delta(W)} x_e \geq \binom{S}{2} + \binom{q}{2} + q(S-q), \qquad \forall W \subset V \tag{2.28}$$

is a valid constraint for $R(G, S)$.

Proof. Equation (2.28) is at equality when S vertices from W are clustered together, and the other q vertices are clustered together. Following from the minimum size constraint on cluster size, each vertex in q has to be connected to $S - q$ vertices outside W. This is the minimum value for the number of edges connected with W, because to replace any one edge inside W, we have to add another 2 edges connected to $V\backslash W$ to satisfy the size constraint (2.1); and to shift any vertex from the group of size q to the group of size S would also increase the total number of edges by $(S-q)$. $\square$

Figure 2.4 shows the support graph for an example of constraint (2.28). Here $w=7$, $S=4$, $q=3$.

$$\sum_{e\in E(W)} x_e + \sum_{e\in\delta(W)} x_e \geq 9+3=12 \tag{2.29}$$

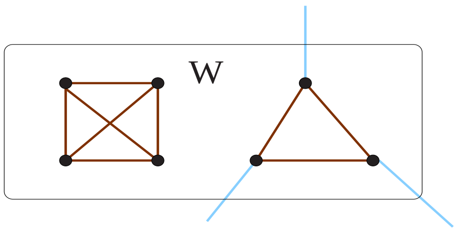

Figure 2.4: Illustration for Flower Inequalities

Figure 2.5 illustrates the two possible ways to tighten up constraint (2.29), which corresponds to the solid line in the figure. These tightenings modify the coefficients of (2.29) so that other configurations of W also satisfy the constraint at equality. We are interested in two particular configurations: one is to have every vertex in W in one big cluster so that $\sum_{e\in E(W)} x_e = 21$ and $\sum_{e\in\delta(W)} x_e = 0$; the other is to have each vertex in W in a separate cluster so that $\sum_{e\in E(W)} x_e = 0$ and $\sum_{e\in\delta(W)} x_e = 21$. These give the constraints corresponding to the dashed line and the dotted line, respectively. The cutting plane generated from the latter case turns out to be already implied by the degree constraints. So we are left with the first case, tightening up the solid line to the dashed line.

As we can see from figure 2.5, to the right of the intersection point (3,9), the solid line is already tighter than the dashed line. Since solid line is already valid, the dashed line has to be valid. This gives the first part of the proof for the following

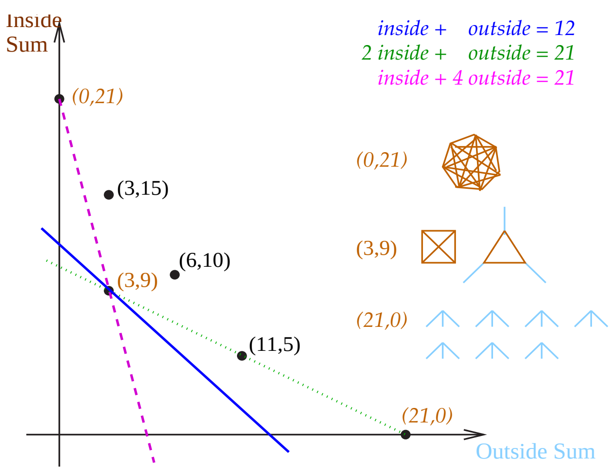

Figure 2.5: Lifting Flower Inequalities

theorem, which summarizes the above observations.

Theorem 2.3.8. *The lifted flower constraint*

$$(S-q)\sum_{e\in E(W)} x_e + S\sum_{e\in\delta(W)} x_e \geq (S-q)\binom{S+q}{2} \tag{2.30}$$

is a valid constraint for $R(V,S)$, $\forall W \subset V, |W| = S+q, 0 \leq q < S$.

Proof. For any partition Π on W let x be the incidence vector of Π, we'll prove the theorem according to two cases, depending on the value of $\sum_{e\in\delta(W)} x_e$ for Π.

(i) When $\sum_{e\in\delta(W)} x_e > (S-q)q$,

$$\begin{aligned}
&(S-q)\sum_{e\in E(W)} x_e + S\sum_{e\in\delta(W)} x_e \\
&= (S-q)(\sum_{e\in E(W)} x_e + \sum_{e\in\delta(W)} x_e) + q\sum_{e\in\delta(W)} x_e \\
&\geq (S-q)(\frac{S(S-1)}{2} + \frac{q(q-1)}{2} + q(S-q)) + q\sum_{e\in\delta(W)} x_e \\
&\qquad \text{from (2.28)} \\
&> (S-q)(\frac{S(S-1)}{2} + \frac{q(q-1)}{2} + q(S-q)) + q(q(S-q)) \\
&\qquad \text{from the assumption } \sum_{e\in\delta(W)} x_e > (S-q)q \\
&= (S-q)(\frac{S(S-1)}{2} + \frac{q(q-1)}{2} + qS) \\
&= (S-q)\binom{S+q}{2}
\end{aligned}$$

(ii) When $\sum_{e\in\delta(W)} x_e \leq (S-q)q$, partition Π on W has to be of the form $\Pi = \{W_1, W_2\}$, and $S' = |W_1| \geq S$, $r' = |W_2| \leq q$, $S'+q' = S+q$. Define another two partitions on W, $\hat{\Pi} = \{W_1 + w, W_2 - w\}$ and $\tilde{\Pi} = \{W\}$. Let

$$\begin{aligned}
\widehat{inside}_+ &= |E(\hat{\Pi})| - |E(\Pi)|, \\
\widehat{outside}_- &= |\delta(\Pi)| - |\delta(\hat{\Pi})|, \\
\widetilde{inside}_+ &= |E(\tilde{\Pi})| - |E(\Pi)|, \\
\widetilde{outside}_- &= |\delta(\Pi)| - |\delta(\tilde{\Pi})|
\end{aligned}$$

We compute them and get

$$\begin{aligned}
\widehat{inside}_+ &= S' - (q'-1) \\
\widehat{outside}_- &= (S-q') - (q'-1) \\
\widetilde{inside}_+ &= S'q' \\
\widetilde{outside}_- &= (S-q')q'
\end{aligned}$$

Which implies

$$\begin{aligned}\frac{\widehat{inside_+}}{\widehat{outside_-}} &= \frac{S'-(q'-1)}{(S-q')-(q'-1)} = 1 + \frac{S'-S+q'}{(S-q')-(q'-1)}\\ \frac{\widetilde{inside_+}}{\widetilde{outside_-}} &= \frac{S'}{S-q'} = 1 + \frac{S'-S+q'}{S-q'}\end{aligned}$$

So we have

$$\frac{\widehat{inside_+}}{\widehat{outside_-}} \geq \frac{\widetilde{inside_+}}{\widetilde{outside_-}} \tag{2.31}$$

This implies that $\hat{\Pi}$ will satisfy equation (2.31) as long as there exist one Π satisfying equation (2.31). Since partition $\{W_1, W_2\}$, $S' = |W_1| = S$, $q' = |W_2| = q$ satisfies (2.31), starting with this partition, and from induction we get that all Π satisfy (2.31).

□

Part (ii) of the proof is also illustrated in figure 2.5. Notice that a line connecting points (3,15) and (3,9) has a bigger slope than the dash line that corresponds to constraint (2.30), so point (3,15) is above this dash line, in other words, constraint (2.30) is valid for the partition corresponding to point (3,15).

From lemma 2.3.2, we know any valid constraint $g^T x \leq h$ implying flower inequality must have $g(i,j) = 0$, $i, j \notin W$. But the flower constraint is not a facet in general. For example, see Figure 2.6, when $|W| = 5, S = 4, q = 1$, any incidence vector satisfying the following flower constraint at equality

$$3 \sum_{e \in E(W)} x_e + 4 \sum_{e \in \delta(W)} x_e \geq 30 \tag{2.32}$$

also satisfies the following constraint at equality:

$$-\sum_{e \in cycle(W)} x_e + 4 \sum_{e \in E(W)-cycle(W)} x_e + 2 \sum_{e \in \delta(W)} x_e \geq 15 \tag{2.33}$$

where $cycle(w)$ is a cycle in W. Figure 2.6 shows the only two configurations satisfying flower constraint (2.32) at equality. They also both satisfy constraint (2.33) on any cycle covering the 5 vertices, for example cycle ABCDEA. The question remains open on how to lift the flower constraint to a facet.

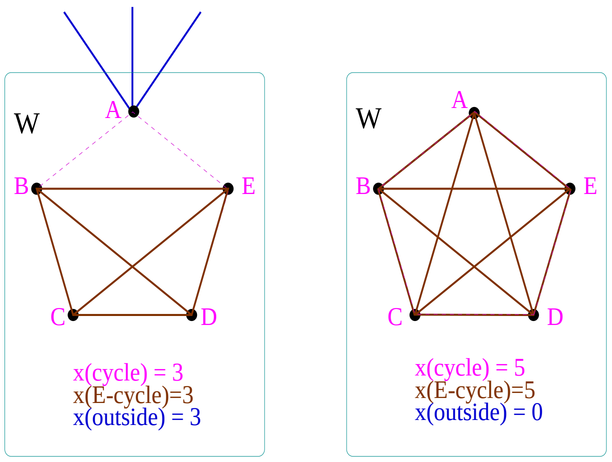

Figure 2.6: Two Configurations Satisfying Inequalities (2.33) at Equality

2.3.4.2 How to Find Violated Flower Constraints

Since Flower constraints are dense constraints, we only look for them when we couldn't find other cutting planes. Again, we are going to use heuristic method to find them. According to the current LP solutions, we divide the vertices into connected components heuristically, then we check if any of these components violate the flower constraint. The intuition behind this strategy is a realization that the flower constraint is more likely to be violated when the x weights are more concentrated on the edges within a component. For example, consider again the example of figure 2.4. Setting $x_e = \frac{1}{2}$ for $e \in E(W)$ and $x_e = 0$ for $e \in \delta(W)$, this satisfies the degree constrains on W, but violates flower constraint (2.30).

2.4 Computational Results

In this part, we are going to show the frame work of our algorithm, and the results of our computational experiments.

2.4.1 Branch and Cut algorithm

Algorithm 2 shows the framework of a branch and cut algorithm that we use here. When we are concentrating on adding in cutting planes at the root node, the LP relaxations are solved by an interior point method, this is because, initially the cutting planes are deep, the LP solution is only used to generate cutting planes, an absolute optimal solution is not necessary. In fact, a close to optimal solution helps to generate deeper cutting planes. After we start branching, we switch to the simplex method to solve the LP relaxation, since at this time the LP solution are much closer to optimality. A more detailed explanation on the combination of interior point method and simplex method in a branch and cut algorithm can be found in Mitchell and Borchers[48].

The cutting planes that are specific to this problem are added in the separation routine shown in algorithm 3.

Algorithm 2 Branch and Cut Framework

```
1: Initialize.
2: Approximately solve the current LP relaxation using an interior point method.
3: If the gap between the value of the LP relaxation and the value of the incumbent
   integer solution is sufficiently small, STOP with optimality.
4: If the duality gap for the current LP is smaller than $10^{-8}$ or if a given limit on the
   number of outer iterations have been reached, call MINTO to do a branch-and-
   cut until an optimal solution is proved or a time limit runs out. To guarantee the
   solution returned by MINTO is a feasible solution, we add in violated triangle
   constraints as cutting planes.
5: Try to improve the incumbent solution locally by switching vertices and moving
   extra vertices around.
6: Use the separation routine Algorithm 3 to find violated cutting planes and return
   to Step 2.
```

2.4.2 A Heuristic Algorithm

Domingo-Ferrer and Mateo-Sanz [14] surveyed heuristic methods for a very similar problem to CPPMIN in the application to micro-aggregation problems. The difference between the problem discussed in [14] and CPPMIN is the objective function. A slightly different objective function is used in [14] to model the homogeneity within a cluster. We implemented one heuristic algorithm for this slightly different

Algorithm 3 Separation Routine

```
1: The algorithm first searches for triangle inequalities using complete enumeration.
   Inequalities are bucket sorted by the size of the violation. Inequalities are added
   starting with those in the most violated subset, ensuring that no two of these
   added inequalities share an edge. The violation of the last constraint added is
   restricted to be no smaller than a multiple of the violation of the first constraint
   added.
2: If not enough constraints are added so far, add 2-partition inequalities.
3: If not enough constraints are added so far, add pigeon constraints of size $n-1$.
4: If not enough constraints are added so far, add pigeon constraints of size $k+1$.
5: If not enough constraints are added so far, add flower constraints.
```

problem in [14] to compare with our cutting plane approach. The reason we choose this particular algorithm is because its one-run performance is among the best of the heuristic algorithms mentioned in [14]. Notice computation time is not an issue here, instead the optimality of the heuristic solution is important. The heuristic algorithm is illustrated in Algorithm 4.

Algorithm 4 A Heuristic Algorithm for CPPMIN

```
1: Let $U = V$.
2: Find the two most distant vertices $x_s, x_t \in U$.
3: Form a cluster around $x_s$ with the $S-1$ closet vertices to $x_s$ in $U$, remove these
   $S$ vertices from $U$.
4: Form a cluster around $x_t$ with the $S-1$ closet vertices to $x_t$ in $U$, remove these
   $S$ vertices from $U$.
5: If $|U| \geq 2S$, go back to step 2.
6: If $S \leq |U| < 2S$, form a new cluster using these vertices.
7: If $|U| < S$ assign each vertex to the same cluster its closest neighbor belongs to.
8: Try to improve the solution locally by switching vertices and moving extra ver-
   tices around.
```

2.4.3 Random Uniform Problems

In this category, we generate 2 types of data.

Type I : Vertices are generated randomly on a unit square following a uniform distribution. The edge weight is the integral part of 100 times the distance between vertices.

Type II : In this case, we don't generate vertices explicitly, instead, edge weights are generated directly as uniformly distributed random variables between 1 and 100. In this case, the edge weights do not satisfy the the triangle inequalities any more, thus the problem turns out to be much harder to solve than Type I problems, because on average, every vertex is equally close to every other vertex.

For each type of data, we consider two choices of S, $S = 4$ or $S = 7$. In the following discussion, we will be concentrating on the case of $S = 4$, but as one can see from the tables, the case of $S = 7$ is very similar, so the analysis for the case of $S = 4$ holds the same as for the case of $S = 7$. In Tables 2.1, 2.2 and 2.4, we consider $S = 4$, we generate 5 groups of problems between 21 and 103 vertices. Each group includes all the 3 cases of remainders, 1, 2 and 3. Our experimental data showed that there is no major difference between these three cases, so we don't show the results separately in the tables. 5 problems of each case were generated, so totally, for each column, 15 instances were generated and solved. Every number reported in the tables is the average performance of 15 instances of the same size.

Every problem is attacked in three ways. First, the heuristic method is used to find a good solution, then the cutting plane approach tries to generate a good linear programming approximation to the problem, and if it can not prove the optimality of the best feasible solution, we finally resort to a branch-and-cut code in MINTO.

Accordingly, each table is divided into 3 blocks, corresponding to these three approaches, labelled as "Heuristic Alg", "Cutting Plane" and "MINTO run" respectively. The first block gives the gap and time for the heuristic approach. The second block corresponds to the cutting plane part of the algorithm. The rows in the table give the number of problems solved to optimality, out of the total of 15 (or 30 for S=7) problems, how many cutting plane solutions improved the heuristic solution, the average final gap, the average running time (in seconds), the average number of total number of cuts, pigeon constraints and flower constraints added, the average number of outer iterations, and the average number of interior point iterations. The third block is the result after branching using MINTO 3.0.2. For problems not solved by the cutting plane scheme, we finish the process off with MINTO to try to obtain an optimal solution. It reads in the cutting plane formulation from an MPS file, then uses a branch and cut algorithm with only triangle constraints added in as cutting planes. The LP relaxations are solved by CPLEX 6.6. The runtime are reported with an upper bound of 500 seconds. The results reported include the number of problems solved to optimality by MINTO (notice this does not include the problems that had already been solved in the cutting plane method), how many MINTO solutions improved the cutting plane solution, the final gap between the best IP solution and LP lower bound, total running time, the number of branch and bound nodes.

The experiments are done on a Sun Ultra 10 Workstation. The heuristic and cutting plane algorithms are coded in FORTRAN. The MINTO branch and bound part is coded in C.

The strength of the pigeon and flower constraints is shown by comparing Table 2.1 and Table 2.2. Both tables used the same set of testing data, but in Table 2.1, we didn't try to add any pigeon or flower constraints. The block labelled "Improvement" compared the difference between the two sets of results after cutting plane code, first by the number of better integer solutions found (out of 15 instances for each column), then by the improvement on LP lower bound. Considering these improvements with the extra time we have to spent to look for pigeon and flower constraints, which is about 1 or 2 minutes more, we can say that pigeon and flower constraints are important constraints for CPPMIN problems. Of course the performance of the branch and bound part of the problem is improved too in Table 2.2 since we started with a tighter relaxation here. Figure 2.7 shows a typical run of the cutting plane algorithm.

We only showed the performance with triangle and 2-way constraints for Type I data here, but similar results are observed in other kind of data too. Notice that feeding all the triangle inequalities to CPLEX directly is impractical; CPLEX was unable to solve a 25 vertex instance in 1 day.

Table 2.3 for the case of $S = 7$ follows the same structure except now each column corresponds to the average of 30 instances.

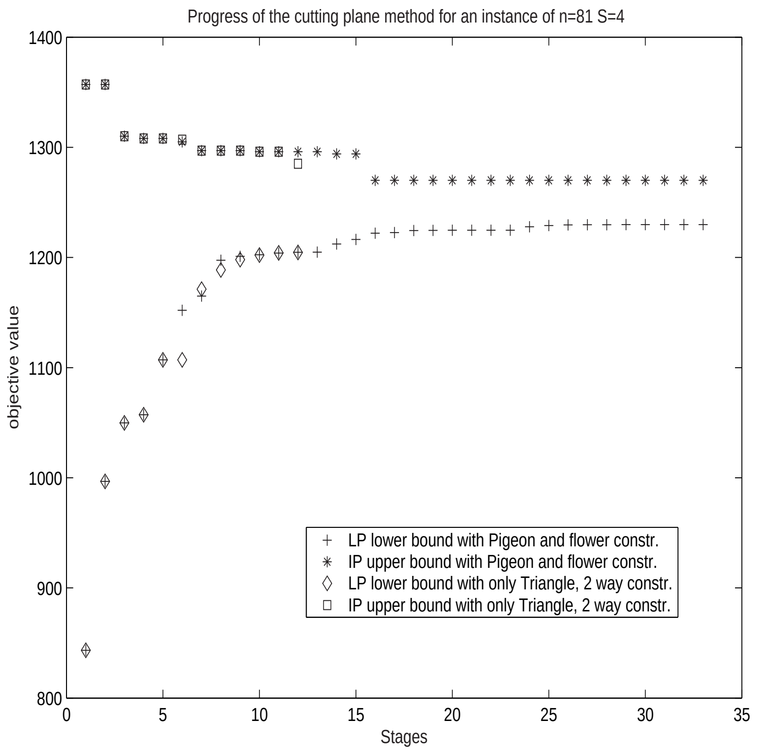

Figure 2.7: Progress of the Cutting Plane Algorithm

Table 2.2 and Table 2.3 show that the cutting plane algorithm was typically able to solve the Type I problems within 4% of optimality in about 3 minutes for a problem of size around 100. The branch and cut code in MINTO is run basically to check the optimality of the solution. So we only checked and added triangle constraints. The results show that the branch and bound method doesn't improve the solution or the gap much, so it is probably not necessary in most of the cases.

Table 2.4 shows the result for Type II problems. These are harder partition problems, because on average, each vertex is equally close to every other vertex. While trying to solve them using our algorithm, a lot more violated triangle inequalities are generated since the edge weights do not satisfy the triangle inequality any more. As we can see from Table 2.4, it takes much longer time to solve them. Due to the difficulty of this type of problems, we only showed the case for $S = 4$.

Our problem is equivalent to maximizing the sum of inter-cluster distances. In this metric, the percentage of error is far smaller. In our formulation (CPPMIN), suppose the best integer solution found is $\bar{x}$, the optimal solution to the lp relaxation is x. We compute the relative gap as

$$gap_{min} = \frac{\sum\limits_{e\in E} c_e\bar{x}_e - \sum\limits_{e\in E} c_e x_e}{\sum\limits_{e\in E} c_e x_e}$$

In the equivalent problem of maximizing the sum of inter-cluster distances, the corresponding relative gap would be

$$\begin{aligned} gap_{max} &= \frac{\sum\limits_{e\in E} c_e(1-x_e) - \sum\limits_{e\in E} c_e(1-\bar{x}_e)}{\sum\limits_{e\in E} c_e(1-x_e)} \\ &= \frac{\sum\limits_{e\in E} c_e\bar{x}_e - \sum\limits_{e\in E} c_e x_e}{\sum\limits_{e\in E} c_e - \sum\limits_{e\in E} c_e x_e} \end{aligned}$$

When the graph G is a complete graph, most of the x_e are zero, therefore, gap_{max} is much smaller than gap_{min}. For example, for a Type I problem of 102 nodes, a gap of $gap_{min} = 3\%$ in our formulation can easily correspond to a gap of around $gap_{max} = 0.02\%$ in this alternative formulation.

2.4.4 Micro-aggregation Problems

One application of CPPMIN that we are particularly interested in is micro-aggregation problem. Micro-aggregation is a technique to process statistical data for releasing to the public, so that the confidentiality of respondents are protected and the informational content of the data are preserved as much as possible. It is used for economic data where respondent identifiability is high. Data are divided into groups with varying size ($\geq$ fixed size), to avoid a large gap within a group. For univariate data, the data can be sorted and the problem can be solved efficiently using a dynamic programming approach, see [26], but for higher dimensional data, sorting is no longer well defined. Sande [57] introduced the problem and several possible methods for higher dimension data. Domingo-Ferrer and Mateo-Sanz [14] gives a good summary on the heuristic methods. Micro-aggregation problems in higher dimension can be described as clustering problems with a variable number

n	21-23	41-43	61-63	81-83	101-103
$k = \lfloor \frac{n}{S} \rfloor$	5	10	15	20	25
Heuristic Alg.					
Gap	9.85%	12.32%	13.12%	14.54%	12.68%
Time	0.0060	0.0141	0.0278	0.0441	0.0672
Cutting Plane					
Solved exactly	1	0	0	0	0
Better Solution	12	14	15	15	15
Gap	4.63%	4.65%	5.28%	5.97%	5.37%
Time	0.89	3.80	7.98	19.43	41.05
Cuts added	64	121	168	227	311
Pigeon	-	-	-	-	-
Flower	-	-	-	-	-
Outer Iter	8	12	12	15	18
Inner Iter	55	120	132	195	253
MINTO run					
Solved exactly	14	15	14	4	0
Better Solution	0	2	9	12	12
Gap	0%	0%	0.55%	4.28%	4.47%
time	3.26	41.43	189.18	417.39	500.60
nodes	39	156	361	457	290

Table 2.1: Branch-and-Cut Results on CPPMIN Type I Problems for $S = 4$ with only Triangle and 2-way constraints

of clusters and a minimum cluster size.

The Type I data in the last section is one kind of simulation for some economic data. But data from micro-aggregation problems usually has the following two important properties

1. Cluster size are small, i.e $S << k$.

2. Economic data tends to be highly skewed.

Accordingly, we generate these following two types of data for our experiments, as suggested by Sande [58].

Type III : Vertices are randomly generated on a plane with the axes being bivariate, and each following an exponential distribution independently. The edge weight is the integral part of 100 times the distance between vertices. This case can simulate economic data that tends to be highly skewed with a long tail. An

n $k = \lfloor \frac{n}{S} \rfloor$	21-23 5	41-43 10	61-63 15	81-83 20	101-103 25
Heuristic Alg.					
Gap	6.97%	10.27%	11.04%	12.40%	11.19%
Time	0.0066	0.0143	0.0273	0.0438	0.0664
Cutting Plane					
Solved exactly	4	1	0	0	0
Better Solution	12	14	15	15	15
Gap	1.87%	2.67%	3.02%	3.65%	3.77%
Time	6.99	23.08	58.42	110.58	175.63
Cuts added	97	178	261	339	433
Pigeon	12	9	9	8	7
Flower	2	4	7	9	8
Outer Iter	16	25	31	36	37
Inner Iter	121	285	415	523	577
Improvement					
Better than Tri	0	1	0	8	7
LPOBJ Improve	2.57%	1.75%	1.82%	1.87%	1.31%
MINTO run					
Solved exactly	11	14	14	5	0
Better Solution	0	1	3	8	6
Gap	0%	0%	0.30%	2.67%	3.47%
time	3.25	24.07	157.06	393.70	510.13
nodes	11	38	182	240	139

Table 2.2: Branch-and-Cut Results on CPPMIN Type I Problems for $S = 4$

example of the solution on this kind of data is shown in Figure 2.8. The performance of the algorithm on this kind of data is shown in Table 2.5 and 2.6.

Type IV : Vertices are randomly generated on a plane. One axis has an exponential distribution, the other axis has a $uniform \times exponential$ distribution. More precisely, let u be exponential, v be uniform between 0 and 1, then $x = u$ and $y = u \times v$. The edge weight is the integral part of 100 times the distance between vertices. In term of economic data, one can think of u as the size of measure of economic data and v as the fraction of the size which is spent on payrolls. Some businesses are very labor intensive and others are capital intensive with less labor. An example of the solution on this kind of data is

n	36-41	71-76
$k = \lfloor \frac{n}{S} \rfloor$	5	10
Heuristic Alg.		
Gap	4.65%	8.03%
Time	0.0100	0.0272
Cutting Plane		
Solved exactly	0	0
Better Solution	25	35
Gap	2.10%	2.32%
Time	67.93	235.65
Cuts added	418	773
Pigeon	19	15
Flower	1	2
Outer Iter	35	40
Inner Iter	336	546
MINTO run		
Gap	0.58%	2.04%
time	150.60	402.68
nodes	147	76

Table 2.3: Branch-and-Cut Results on CPPMIN Type I Problems for $S = 7$

shown in Figure 2.9. The results for this type of problems are shown in Table 2.7 and 2.8.

From the results in Tables 2.5-2.8, we can see that these two types of problem are harder than the Type I problem, but none the less we can solve them to within 4% of optimality in about 3 minutes. Again it takes quite a long time to use a branch and bound algorithm to try to verify that the solutions we get really are optimal solutions.

We can also see that the problems of Type IV are a little bit easier than Type III. This is because the y variable is always less than x, thus making the x axis more important, and the problem closer to the easier one dimensional clustering problem.

2.5 Conclusions, Other Applications and Future Work

In summary, we have shown that the Clique Partition Problem with Minimum size constraints can be formulated as an IP problem and approximated nicely by a

n	21-23	41-43
$k = \lfloor \frac{n}{S} \rfloor$	5	10
Heuristic Alg.		
Gap	13.50%	24.16%
Time	0.0062	0.0124
Cutting Plane		
Solved exactly	0	0
Better Solution	15	15
Gap	4.74%	13.28%
Time	75.69	1657.34
Cuts added	393	1027
Pigeon	11	9
Flower	0	0
Outer Iter	25	34
Inner Iter	192	305
MINTO run		
Solved exactly	15	0
Better Solution	0	6
Gap	0%	8.93%
time	40.83	476.70
nodes	80	169

Table 2.4: Branch-and-Cut Results on CPPMIN Type II problems for $S = 4$

LP relaxation by adding cutting planes. We gave the mathematical formulation of the IP problem, discussed the polyhedral structure of the corresponding polytope, gave two kind of specialized cutting planes, and showed the pigeon constraint is facet defining in most cases. Our computational results showed that with these special constraints, a cutting plane scheme leads consistently to high quality solutions in a reasonable amount of time. This proves the effectiveness of these special constraints.

CPPMIN also arises naturally from other applications such as telecommunication clustering see Lisser and Rendl [37], locating communication centers, see Guttmann-Beck and Rubinstein [25], and alignment of sports teams, see Mitchell [47].

Regarding micro-aggregation problems, different kinds of objective function have also be proposed to characterize the within-group homogeneity. Domingo-Ferrer and Mateo-Sanz [14] introduced a few of them that are all based on the dis-

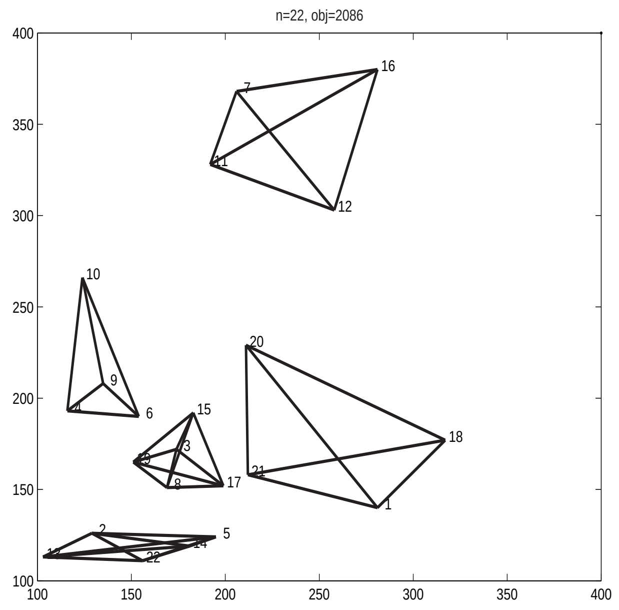

Figure 2.8: Solution for a CPPMIN Problem of Type III: $n = 22, S = 4$

tance between each vertex and its cluster center, which would result in a non-linear objective function. Sande [57] suggested using the total weight of the minimum spanning tree within each clusters. We will address this particular form in the next chapter.

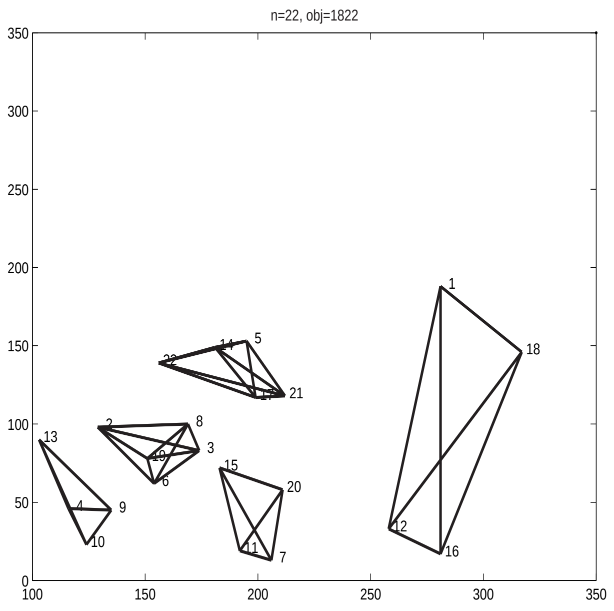

Figure 2.9: Solution for a CPPMIN Problem of Type IV: $n = 22, S = 4$

n	21-23	41-43	61-63	81-83	101-103
$k = \lfloor \frac{n}{S} \rfloor$	5	10	15	20	25
Heuristic Alg.					
Gap	3.12%	6.98%	9.49%	8.54%	9.01%
Time	0.0056	0.0113	0.0207	0.0313	0.0475
Cutting Plane					
Solved exactly	1	0	0	0	0
Better Solution	6	14	15	15	15
Gap	2.14%	2.56%	2.03%	2.60%	3.46%
Time	9.45	20.20	49.94	104.74	181.08
Cuts added	123	184	274	369	438
Pigeon	13	10	8	8	8
Flower	1	4	5	8	11
Outer Iter	24	26	31	36	39
Inner Iter	195	298	434	560	646
MINTO run					
Solved exactly	14	15	13	7	1
Better Solution	0	2	2	7	4
Gap	0%	0%	0.48%	1.76%	3.01%
time	4.39	28.52	155.85	361.73	485.88
nodes	25	81	253	247	125

Table 2.5: Branch-and-Cut Results on CPPMIN Type III Problems for $S = 4$

n	36-41	71-76
$k = \lfloor \frac{n}{S} \rfloor$	5	10
Heuristic Alg.		
Gap	3.87%	6.42%
Time	0.0129	0.0285
Cutting Plane		
Solved exactly	0	0
Better Solution	16	29
Gap	2.05%	2.11%
Time	62.12	229.22
Cuts added	390	733
Pigeon	17	14
Flower	1	3
Outer Iter	33	40
Inner Iter	327	547
MINTO run		
Solved exactly	24	0
Better Solution	0	1
Gap	0.59%	1.56%
time	194.75	465.86
nodes	136	71

Table 2.6: Branch-and-Cut Results on CPPMIN Type III Problems for $S = 7$

n	21-23	41-43	61-63	81-83	101-103
$k = \lfloor \frac{n}{S} \rfloor$	5	10	15	20	25
Heuristic Alg.					
Gap	6.44%	5.60%	7.43%	8.05%	8.60%
Time	0.0057	0.0115	0.0205	0.0318	0.0482
Cutting Plane					
Solved exactly	3	1	0	0	0
Better Solution	10	12	15	15	15
Gap	3.02%	1.42%	2.41%	2.36%	3.30%
Time	5.12	20.27	43.80	97.17	150.30
Cuts added	93	189	263	355	422
Pigeon	11	12	8	7	7
Flower	1	4	5	8	9
Outer Iter	15	26	28	37	36
Inner Iter	118	313	398	563	593
MINTO run					
Solved exactly	12	14	14	8	1
Better Solution	0	1	3	4	2
Gap	0%	0%	0.10%	1.24%	3.01%
time	6	16.97	152.04	347.16	476.36
nodes	66	38	260	257	177

Table 2.7: Branch-and-Cut Results on CPPMIN Type IV Problems for $S = 4$

n	36-41	71-76
$k = \lfloor \frac{n}{S} \rfloor$	5	10
Heuristic Alg.		
Gap	4.05%	5.53%
Time	0.0130	0.0360
Cutting Plane		
Solved exactly	1	1
Better Solution	30	34
Gap	1.79%	2.03%
Time	58.86	290.00
Cuts added	386	769
Pigeon	15	15
Flower	1	2
Outer Iter	29	39
Inner Iter	288	533
MINTO run		
Solved exactly	29	3
Better Solution	0	0
Gap	0.56%	1.79%
time	157.59	478.51
nodes	136	38

Table 2.8: Branch-and-Cut Results on CPPMIN Type IV Problems for $S = 7$

CHAPTER 3
Minimum Weight Constrained Forest Problem

3.1 Introduction

In this chapter, we are going to use the branch-and-cut method to solve the Minimum Weight Constrained Forest (MWCF) problem, where the cost for each cluster is measured as the weight of the minimum spanning tree of the cluster. Given a graph $G = (V, E)$ with edge weights c_e, and an integer $S < n$, a tree is a connected acyclic subgraph of G, and a forest is an acyclic subgraph of G. The weight of a forest is the sum of the edge weights of all the edges in the forest. The MWCF problem looks for a spanning forest of graph G with each tree in the forest spanning at least S vertices. The objective is to minimize the weight of the spanning forest.

Imielińska, Kalantari and Khachiyan [33] showed that this is an NP hard problem when $S \geq 4$, and described a simple greedy heuristic with an approximation ratio of 2. The time complexity for their algorithm is $O(m + nlogn)$ for general graphs, where $m = |E|, n = |V|$. Their algorithm basically computes a minimum spanning tree (MST) for G, then sorts the edges in the MST according to their weight, the heuristic selects in a greedy fashion a forest such that each of its trees spans at least S vertices. Goemans and Williamson [19] extended the above heuristic algorithm to a class of NP-hard minimum-cost graph problems.

Ali and Huang [1] addressed the minimum weight balanced spanning forest problem, in which the number of trees is fixed and the number of vertices in each tree cannot differ by more than 1. They set up the problem as an integer programming problem similar to ours, but our formulation has a tighter LP relaxation. The algorithm used Lagrangian relaxation and heuristics to find lower bound and upper bounds. The Lagrangian relaxation is tightened up by a dual ascent procedure that is similar to column generation. Their computation results showed that on average, a bound of about 3% can be obtained in 700-1400 dual iterations for a 100-vertex graph with 20% edge density. We will compare this result with ours and show that our approach results in better performance.

Guttmann-Beck and Hassin [25] proposed an approximation algorithm on a similar problem: to partition V into p subsets of given size, so as to minimize the total weight of the edges in the spanning trees in each cluster. While assuming the triangle constraints are satisfied by the edge weights, their algorithm runs in $O(p^2 4^p + n^2)$ time and comes within a factor of $2p - 1$ of optimality. When the size of every cluster are the same, i.e., when it is an equi-partition, the complexity can be reduced to $O(n^2)$.

Yamada, Takahashi and Kataoka [65] described a mini-max spanning forest problem where the root of each tree is given. The objective is to minimize the maximum of the tree weights. They proposed a branch and bound method to solve the problem exactly. It branches on edge variables. But instead of solving an LP to find a lower bound, they used combinatorial methods to establish a lower bound on the branch-and-bound node. This problem differs from our MWCF problem in two aspects. First the number of trees in the forest is given by the number of roots. The specification of the exact root vertices can be enforced by changing the cost function to a very large number for the edge connecting every pair of the root vertices, so that they fall into different trees automatically. The second major difference is that they used the maximization criterion in the objective function to achieve balancing in the size of the trees, while we put a size restriction in as a hard constraint.

A number of other variations of the spanning forest problem have been considered before. The minimum weight spanning forest problem with no other constraints can be solved in polynomial time using a greedy algorithm. A degree-constrained spanning tree problem can be also be solved in polynomial time using a greedy method. But other constrained spanning tree problems belong to NP-complete problems when the capacity constraint is not trivial. Cordone and Maffioli [13] discussed the complexity of a general family of spanning forest problems, including root constraints, inclusion constraints and weight constraints. They showed that most of them are strongly NP-hard, except for a few special cases.

The MWCF problem is a partition problem with the weight of the minimum spanning tree as the cost for each partition. The weight of a minimum spanning tree is a natural analog of the range of each partition. It is also called single link or nearest neighbor clustering in the computer science community. It arises in many applications, such as telecommunication [25], political districting [65, 39]. It is also used in statistical micro-aggregation, see [57].

Most of the research in the literature for spanning forest problems used heuristic methods. Because the similarity between MWCF problem and CPPMIN, and the success of branch-and-cut method on CPPMIN, we are going to apply it on MWCF problem in this chapter. The MWCF problem is related with the CPPMIN problem in the sense that both have a lower bound on the size of each partition, but representing each partition as a spanning tree rather than a clique changes the whole feasible region. In the following sections, we will give our IP formulation of the problem, discuss the corresponding polyhedral structure, give several facets, and talk about how to find them. We show the computational results in Section 5 and draw our conclusion in Section 6.

3.2 Integer Programming Formulation

We define a binary variable x_e, which takes the value 1 if edge e is within the forest and 0 otherwise. Our IP formulation for MWCF is

$$\begin{aligned}
\min \quad & \sum_{e \in E} c_e x_e \\
\text{s.t.} \quad & \sum_{e \in E(Q)} x_e \leq |Q| - 1, && \forall Q \subseteq V && (3.1) \\
& \sum_{e \in E(W)} x_e + \sum_{e \in \delta(W)} x_e \geq |W| - \lfloor \frac{|W|}{S} \rfloor && \forall W \subseteq V && (3.2) \\
& x_e \in \{0, 1\}
\end{aligned}$$

We will use $n = |V|$ to denote the number of vertices. Throughout this paper, we assume G is the complete graph on V. A problem on a non-complete graph can be easily converted to a problem on a complete graph by adding the missing edges in with a large edge weight c_e. As showed by Imielińska, Kalantari and Khachiyan [33], this problem is NP complete when $S \geq 4$. We will assume $S \geq 4$ and $n \geq 2S$

from now on.

Constraint (3.1) is used to prevent cycles. We will refer to it as **cycle constraint**. Constraint (3.2) is a lower bound on the number of edges in a forest of size $|W|$ with at most $\lfloor \frac{|W|}{S} \rfloor$ components. We will refer to it as **flower constraint** since it is similar to the flower constraint for CPPMIN in Chapter 2.

We would like to note here that it is natural to think of enforcing the minimum size requirement S as:

$$\sum_{e \in \delta(U)} x_e \geq 1, \quad \forall U \subseteq V, |U| < S \tag{3.3}$$

But it actually can be implied by the combination of cycle and flower constraints on set U

$$\sum_{e \in E(U)} x_e \leq |U| - 1$$
$$\sum_{e \in E(U)} x_e + \sum_{e \in \delta(U)} x_e \geq |U|$$

Both (3.1) and (3.2) have exponential number of constraints, but most of them are not binding at the optimal solution. We will solve the MWCF problem using a branch-and-cut method starting with the following initial LP relaxation

$$\begin{array}{lll}
\min & \sum_{e \in E} c_e x_e & \\
\text{s.t.} & \sum_{e \in E(V)} x_e \leq |V| - 1 & (3.4) \\
& \sum_{e \in \delta(v)} x_e \geq 1, \qquad \forall v \in V & (3.5) \\
& \sum_{e \in E(V)} x_e \geq |V| - \lfloor \frac{|V|}{S} \rfloor & (3.6) \\
& 0 \leq x_e \leq 1 & (3.7)
\end{array}$$

Constraint (3.4) is of type (3.1). Constraints (3.5) and (3.6) are two special case of type (3.2). Additional constraints will be added in as cutting planes.

3.3 Polyhedral Theories

We define the polytopes $T(G)$ and $F(G,S)$ for the spanning tree and spanning forest as follows:

$$\begin{aligned} T(G) &:= conv\{x \in \{0,1\}^n : \text{x is the incidence vector of a spanning tree } T \text{ on } G\} \\ F(G,S) &:= conv\{x \in \{0,1\}^n : \text{x is the incidence vector of a spanning forest } F \text{ on } G, \\ &\quad\ \text{each tree in the forest has at least S nodes}\} \end{aligned}$$

Obviously $T(G) \subseteq F(G,S)$.

Edmonds [15] has given a complete linear inequality description of $T(G)$.

$$\begin{aligned} T(G) \ = \{x| \quad & x(E) = |V| - 1, \\ & x(E(Q)) \leq |Q| - s(Q), \qquad \forall Q \subseteq V, 2 \leq |S| \leq |V| - 1 \\ & x_e \geq 0, \qquad \forall e \in E\} \end{aligned}$$

Here $s(Q)$ = number of connected components in subgraph Q. Using matroid theory, he showed that the incidence vectors of spanning trees of a connected graph G are exactly the vertices of the above polytope. It was also shown that all these inequalities are facets for the tree polytope $T(G)$ on a complete graph G.

We would like to mention that this is not the only linear inequality description of $T(G)$. Fulkerson [17] proposed another one, which was also proved later by Chopra [9] using a different method.

Now we give our results for the dimension of the polytope $T(G)$ and $F(G,S)$. The dimension of $T(G)$ can be shown using matroid theory [15]. Here we give an alternative proof. The same method will used in all the proofs in this chapter.

Theorem 3.3.1. *Let $G = (V,E)$ be a connected graph with no vertices of degree 1. The dimension of its spanning tree polytope $T(G)$ is $|E| - 1$.*

Proof. We are going to show this by showing that the only equality that is satisfied by all the spanning tree incidence vectors is the equality $e^T x = |V| - 1$, where e is the vector of ones.

Let $g^T x = \beta$ be an equality satisfied by all the incidence vectors x of the spanning trees of G. Let u be a vertex. Pick a spanning tree on $G - \{u\}$. Adding any one of the edges from u to one of its neighbors gives a spanning tree for G. Since all these trees have $g^T x = b$, we must have the coefficient in g corresponding to edge (u, v) be a constant α for for all edges (u, v) incident to u. This same coefficient must also be the coefficient for all edges incident to all the neighbors of u. Spreading outwards, we have $g = \alpha e$, so $g^T x = \beta$ is equivalent to $e^T x = \beta/\alpha$. □

Theorem 3.3.2. *Let $G = (V, E)$ be a connected graph with no vertices of degree 1, $|V| \geq 2S$. The spanning forest polytope $F(G, S)$ is full dimensional.*

Proof. When $|V| \geq 2S$, we have at least one solution $x \notin T(G)$. Since $T(G) \subseteq F(G, S)$ and $dim(T(G)) = |E| - 1$, $dim(F(G, S)) = |E|$, i.e., $F(G, S)$ is full dimensional. □

3.4 Constraints

3.4.1 Bound Constraints

Theorem 3.4.1. *$x_e \geq 0$ is a facet for $F(G, S)$.*

Proof. As mentioned earlier, it has been showed that $x_e \geq 0$ is already a facet for $T(G)$ [15]. Since $|V| \geq 2S$, we can construct a forest of two trees each has at least S vertices, satisfying $x_e = 0$. This is an instance $\notin T(G)$. So $x_e \geq 0$ is also a facet for $F(G, S)$. □

The constraint $x_e \leq 1$ is also a facet for $F(G, S)$, but we'll include it as a special case of the cycle constraint.

3.4.2 Cycle Constraints

3.4.2.1 The Tightness of Cycle Constraints

Theorem 3.4.2. *Given a graph $G = (V, E)$, where $|V| \geq 2S$, cycle constraint* (3.1) *is valid for $F(G, S)$, and it is a facet if and only if $Q = V$ or $|Q| \leq n - S$.*

Proof. (3.1) is valid for $F(G, S)$, according to the definition of a forest. We'll prove when it is a facet according to the three cases.

(i) When $Q = V$, every instance in $T(G)$ satisfies this cycle constraint at equality. Since $dim(T(G)) = |E| - 1$, the cycle constraint on V is a facet for $F(G, S)$.

(ii) When $n - S < |Q| < n$, the cycle constraint is not a facet, because to guarantee every tree has at least S vertices, all the instances satisfying this constraint at equality must be a spanning tree on G, so the cycle constraint on Q is implied by the cycle constraint on V. Or in other words, it is implied by the cycle constraint on V, and the flower constraint on $W = V - Q$:

$$\begin{aligned} \sum_{e \in E(V)} x_e &\leq |V| - 1 \\ \sum_{e \in E(W)} x_e + \sum_{e \in \delta(W)} x_e &\geq |W| \qquad \text{since } |W| < S \end{aligned}$$

(iii) When $|Q| \leq n - S$, it's already a facet for the $T(G)$. We can construct a spanning tree instance x s.t. $x \in F(G, S), x \notin T(G)$: let Q be one cluster and $W = V - Q$ be another cluster. So the cycle constraint on Q is a facet.

□

3.4.2.2 How to Find Cycle Constraints

It is well known that MST problem can be solved by Kruskal's greedy algorithm in $O(|V|^2)$, so it is not surprising that the cycle constraint can be identified in polynomial time. It is showed in Padberg and Wolsey [53], that constraint (3.1) can be identified in $O(|V|^4)$ time by computing a minimum cut, i.e. by carrying out at most $n - 2$ maximum flow calculations. In fact, this is rather slow considering MST can be solved in $O(|V|^2)$ time. In our MWCF problem with positive edge weights, the cycle constraints are also not violated very often. Because of these two reasons, we only use a heuristic algorithm to find the violated cycle constraints. From the fractional LP relaxation solution, we round up the x's that are close to one and use it to divide the graph into several partitions, then check if the cycle constraints are violated on these partitions.

3.4.3 Flower Constraints

3.4.3.1 The Strength of Flower Constraints

Theorem 3.4.3. *Consider flower inequality* (3.2) *on graph $G = (V, E)$, where $n = |V| = kS + r$ and $|W| = pS + q$, it is valid for $F(G, S)$, and defines a facet for $F(G, S)$ if and only if $W = V$ or $q \geq 1$ and $p \leq k - 2$.*

Proof. First we will show (3.2) is a valid constraint for $F(G, S)$. Let x be the incidence vector of a feasible solution. Let $\bar{G}$ be the support graph of x on W, i.e., $\bar{G} = (W, \{e \in E(W) : x_e = 1\})$. Let $p+t$ be the number of disconnected components in $\bar{G}$, then $x(E(W)) = |W| - p - t$. If $t \leq 0$, $x(E(W)) = |W| - p - t \geq |W| - \lfloor |W|/S \rfloor$. If $t > 0$, there are at least t components containing less than S vertices each. Each of these components must be connected to vertices in $V - W$, thus $x(\delta(W)) \geq t$. Thus in both cases, $x(E(W)) + x(\delta(W)) \geq |W| - \lfloor |W|/S \rfloor$.

To prove the necessary and sufficient conditions for constraint (3.2) to be a facet for $F(G, S)$, we consider three cases:

(i) When $W = V$, constraint (3.2) can be written as

$$\sum_{e \in E(V)} x_e \geq |V| - \lfloor \frac{|V|}{S} \rfloor \tag{3.8}$$

Let $g^T x = \beta$ be any equality such that $\{x | x \text{ satisfy (3.2) at equality}, x \in F(G, S)\} \subseteq \{x | g^T x = \beta, x \in F(G, S)\}$. We just need to show that the coefficient in $g(e)$ for any edge $e \in E(V)$ is a constant. This can be done by the same technique in the proof for Theorem 3.3.1.

Pick any pair of edges (u, v) and (u, w) sharing an endpoint. We can partition the vertices into clusters of the right size for satisfying the constraint (3.2) at equality and so that all the endpoints u, v, w of the two edges are in the same cluster. Then there are two feasible solutions that satisfy $g^T x = \beta$ at equality, where the only difference is one includes (u, v) and the other includes (u, w). Hence, all the coefficients in the constraint $g^T x = \beta$ must be the same.

(ii) When $q = 0$, $|W| = pS$, constraint (3.2) is not facet defining because to satisfy it at equality, the pS vertices must be in exactly p trees of size S and they can not be connected to any vertices in $V - W$. So they all satisfy

$$\begin{aligned} \sum_{e \in E(W)} x_e &= |W| - \lfloor \frac{|W|}{S} \rfloor \\ x_e &= 0 \qquad \forall e \in \delta(W) \end{aligned}$$

Thus the dimension for these incidence vectors is less than $|E| - |W||V - W| < |E|$.

(iii) When $q \geq 1$, the cycle constraint on $V - W$

$$\sum_{e \in E(V-W)} x_e \leq |V - W| - 1 \tag{3.9}$$

and the flower constraint on V, (3.8), together imply

$$\sum_{e \in E(W)} x_e + \sum_{e \in \delta(W)} x_e \geq |W| - (\lfloor \frac{|V|}{S} \rfloor - 1) \tag{3.10}$$

Comparing inequality (3.10) and (3.2), we get the following necessary condition for (3.2) to be facet defining:

$$\lfloor \frac{|W|}{S} \rfloor < \lfloor \frac{|V|}{S} \rfloor - 1 \tag{3.11}$$

which can be written as $p < k - 1$.

Now we are going to show this is also a sufficient condition by showing any valid constraint of $F(G, S)$ $g^T x = \beta$, satisfying $\{x \in P | x \text{ satisfy } (3.2) \text{ at equality}\} \subseteq \{x \in P | g^T x = \beta\}$ must have the same coefficient for the edges $e \in E(W)$ and $e \in \delta(W)$, and 0 as the coefficients for the edges $e \in E(V - W)$. We refer to the coefficient in $g^T x = \beta$ for edge e as $g(e)$.

When $p < k - 1$,

$$\begin{aligned} |V - W| &= (k - p)S + r - q \\ &= ((k - p - 1)S + r) + (S - q) \end{aligned}$$

Pick any vertices $a \in W$, $c, d \in V - W$, see Figure 3.1. Divide the vertices in W into 2 parts, A and B, such that $a \in A$, $|A| = q$, $|B| = pS$. Divide the vertices in $V - W$ into 2 parts, C and D, such that $c \in C$, $d \in D$, $|C| = S - q$, $|D| = (k - p - 1)S + r$.

Now construct a spanning forest as the following. A and C form one tree of size S with edge (a, c) in the tree, and no other edges in $\delta(W)$ in the tree. B is spanned by p trees of size S. D is spanned by one spanning tree. Let F denote the edges in this spanning forest of G. F satisfies (3.2) at equality, so it satisfies $g^T x = \beta$ at equality.

We first show the coefficient $g(e)$ for edges $e \in E(V - W)$ are zero. The coefficient for the edge (c, d) is zero, because because both F and $F' = F + (c, d)$ satisfy (3.2) at equality, thus both satisfy $g^T x = \beta$ at equality. Since c and d are picked randomly from $V - W$, the result can spread out to all the edges in $E(V - W)$.

We can use the same proof in (i) to show the coefficient for the edges in $E(W)$ are the same.

Now, we only need to show to show the coefficient for the edges in $\delta(W)$ are the same as the coefficient of the edges in $E(W)$.

- When $p > 0$, pick any $b \in B$, let $F'' = F' + (a, b) - (a, c)$, then F', F'' both satisfy constraint (3.2) at equality, so $g(a, b) = g(a, c)$.
- When $p = 0$ and $q > 1$, pick any $u \in A$. If $(a, u) \notin F'$, we can get partition F'' from F' by rearranging the edges in A, such that $(a, u) \in F''$, otherwise, let $F'' = F'$. $F''' = F'' - (a, u) + (u, c)$ satisfies constraint (3.2) at equality. Comparing F''' with F', we get $g(a, u) = g(u, c)$.
- When $p = 0$ and $q = 1$, in other words $|W| = 1$, we can pick any $v \in C + D, v \neq c$, $F'' = F' - (a, c) + (a, v)$ satisfy constraint (3.2) at equality, so $g(a, c) = g(a, v)$.

Since a, b, c, u, v are all arbitrarily chosen in the three cases, we have $g(u, v)$ is a constant for any $u \in W$, $v \in V$. Thus we have showed the flower constraint is a facet when $q \geq 1$ and $p \leq k - 2\ \delta(W)$

$\square$

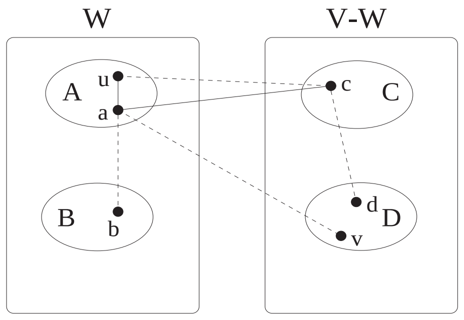

Figure 3.1: Flower Constraints for $F(G, S)$

3.4.3.2 An Example of Flower Constraints

Our numerical results show that the flower constraint is an important type of constraint. Here we give one example. Figures 3.2 to 3.6 are the solutions corresponding to a problem of size $n = 50$, $S = 4$. The vertices are generated randomly on a square of 400×400, the edge weights are taken as the Euclidean distance between each pair of vertices. Different values of the x_e are represented by different line style. Figures 3.2 and 3.3 are the LP and IP solutions from the cutting plane algorithm when we use a less aggressive method to look for violated flower constraints, so two of them were not identified. It results in a gap of 3.09%, and takes 1241 nodes and 40 seconds to find and prove the optimal solution (Fig 3.6) in the branch and cut stage. Fig 3.4 and 3.5 are the corresponding LP, IP solutions after we spent more effort to look for violated flower constraints and found the two missing ones. We have a better LP bound, a better primal solution, giving a gap of 1.8% at the end of the cutting plane procedure, and more significantly, in the branch and cut process, it takes only 3 nodes and 6 seconds to find and prove the optimal solution.

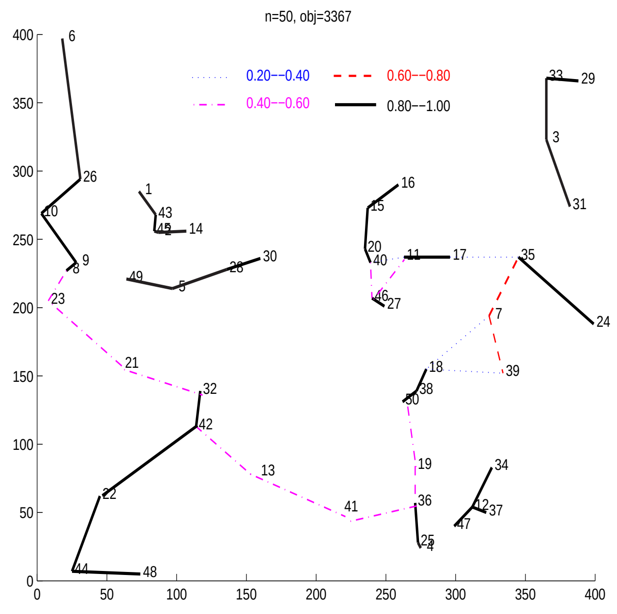

Figure 3.2: Missing Two Flower Constraints: LP Solution

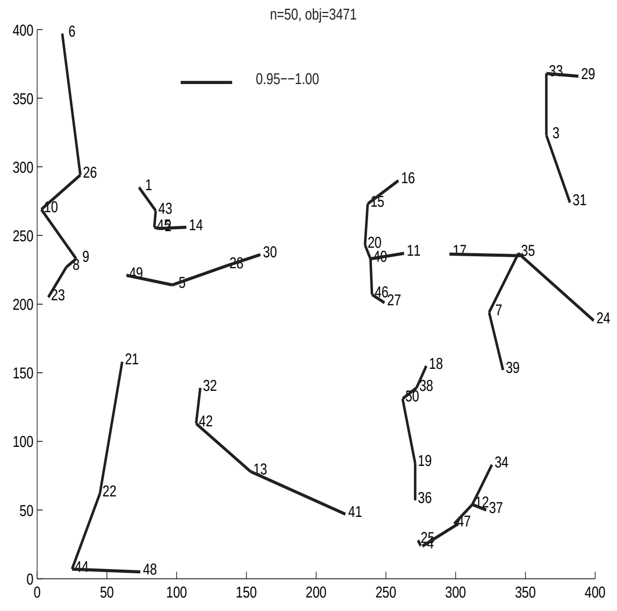

Figure 3.3: Missing Two Flower Constraints: IP Solution

3.4.3.3 How to Find Violated Flower Constraints

Since flower constraints are strong cutting planes, and they are often violated by the optimal solution of the LP relaxation, we spend some effort in identifying

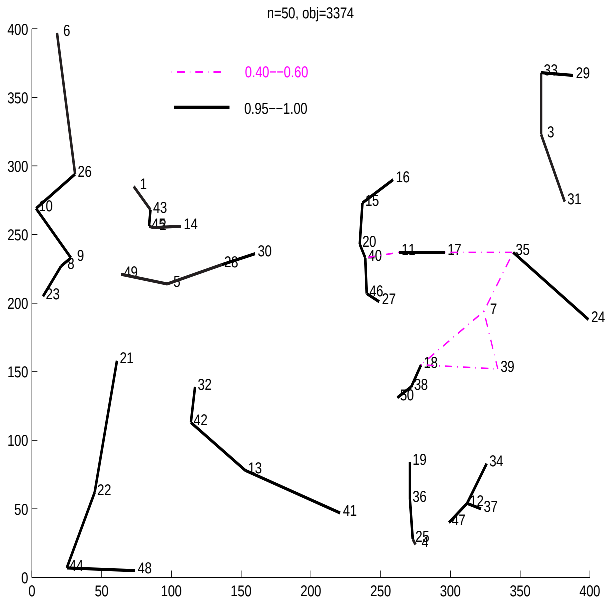

Figure 3.4: With the Two Flower Constraints Added: LP Solution

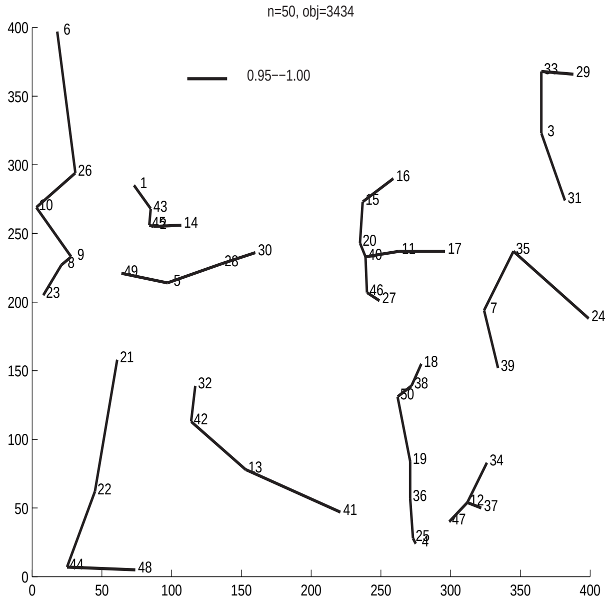

Figure 3.5: With the Two Flower Constraints Added: IP Solution

them. First we check if any of the components found while looking for cycle constraints violates the flower constraints. If no violating flower constraints is found, we try to identify two types of vertices. First, the vertices that are only connected

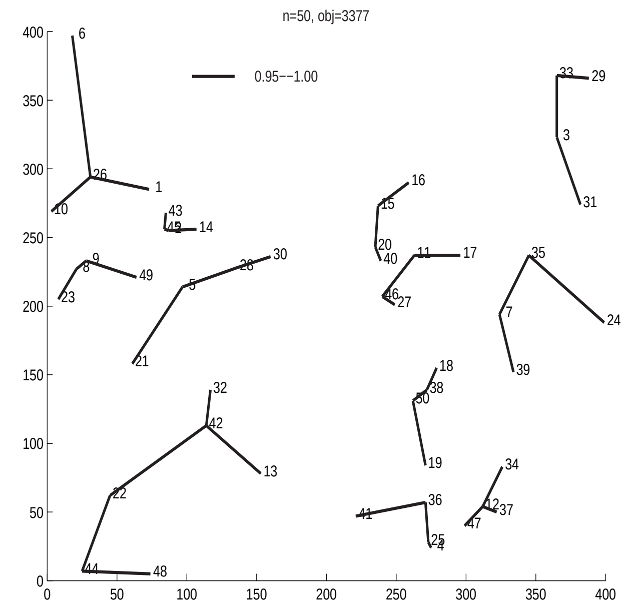

Figure 3.6: Optimal Solution

with two fractional edges and they sum up to 1. Vertices 13, 41, 21, 23, 39, 19 in Figure 3.2 are all examples of this kind of vertices. Second, the vertices that are connected with only one edge of value one and nothing else, the other end of the edge is connected to only fractional edges. Vertices 24 and 27 in Figure 3.2 are examples of this kind of vertices. Then we connect every pair of the found vertices by the shortest path to see if any violation can be found on the shortest path. This part of the algorithm is detailed in Algorithm 5.

3.5 Computational Results

In this part, we are going to show the framework of our algorithm, and the results of our computational experiments. The branch-and-cut code is implemented using MINTO 3.0.2, so the algorithm follows the frame work in MINTO. The separation routine for adding in cutting planes in shown in Algorithm 6.

3.5.1 A Heuristic Algorithm

Algorithm 7 gives an heuristic algorithm for MWCF. It is based on the heuristic algorithm of complexity $O(m + nlogn)$ from Imielinska, Kalantari and Khachiyan

Algorithm 5 Heuristic Algorithm for finding violated flower constraints

```
Input:
  $x$ - LP solution
Output:
  $QSET$ - subsets of vertices violating flower constraints
Steps:
  $QSET = \emptyset$
  Find $Q1 = \{i \in V | x(i, j_1) + x(i, j_2) = 1, x(i, j_1) > 0, x(i, j_2) > 0$ and $x(i, j) = 0, \forall j \neq j_1, j_2\}$
  Find $Q2 = \{i \in V | x(i, j) = 1, x(i, k) = 0, \forall k \neq j, 0 \leq x(j, l) < 1, \forall l \neq i\}$
  Let $Q3 = Q1 + Q2$
  for each pair of vertices $(i, j)$ in $Q3$ do
    Let Q = the vertices on the shortest path in G connecting $i$ and $j$.
    if (3.2) is violated on Q then
      $QSET = \{QSET\} + \{Q\}$
    end if
  end for
```

Algorithm 6 Separation Routine

```
1: Divide the vertices into components according to the $xlp$ value.
2: Check and add violated cycle constraints on these components.
3: Check and add violated flower constraints on these components.
4: If no violated flower constraints are found, call Algorithm 5 to try to find some
   flower constraints.
```

[33]. Because in our experiments, we have $c_e \geq 0$, we can improve upon their algorithm by deleting the edges that connect two subtrees of at least S vertices. Notice in MWCF problem with positive edge weights, a tree in an optimal solution may span more than $2S$ vertices, for example when there is one vertex that is close to every other vertex, while the distances between these other vertices are very large. In other words, we may not be able to divide a tree of $2S$ or more vertices into two subtrees each of at least S vertices.

This post-process improved the performance of the heuristic method from a gap of 10% to around 4%. We implemented it to compare with our branch and cut algorithm, and used the heuristic solution as an initial upper bound.

Algorithm 7 Heuristic Algorithm for Minimum Weight forest

```
Input:
  $W$ - Minimum Spanning Tree
Output:
  $F$ - forest of trees, each spanning at least $S$ vertices
Steps:
  $F_S = \emptyset$
  while $F_S$ does not span V do
    if $W \neq$ empty set, and the shortest edge in $W$ does not connect 2 trees of size
    at least $S$ in $F_S$ then
      $F_S$=$F_S$+{e}
      $W = W - \{e\}$
    end if
  end while
  while $F_S \neq \emptyset$ do
    pick the longest edge $e \in F_S$
    if $e$ connect two tree of at least size $S$ then
      $F_S$=$F_S$-{e}
    else
      $F$=$F$+{e}
      $F_S$=$F_S$-{e}
    end if
  end while
```

3.5.2 Random Uniform Problems

The data used here are generated in the same way as the data used in chapter 2 for CPPMIN problems. Type I is from uniformly distributed points on a square. The edge weights are simply the Euclidean distance. In Type II, the edge weight are generated directly as a random number between 1 and 100. Tables 3.1 to 3.2 are for Type I and Type II data,

Each table is divided into three parts to represent the performances of the heuristic algorithm, the cutting plane algorithm, and the branch-and-cut algorithm. They are labeled as “Heuristic Alg”, “Cutting Plane” and “B&C” respectively.

The first block gives the gap and time for the heuristic approach. The second block corresponds to the result at the end of root node in the branch and cut problem. This is gives us a sense of how good the cutting plane method is. The rows in the table give the total number of instances, the number of instances solved to optimality in the root node, the average gap at the end of cutting plane method, the average running time (in seconds), the average number of cycle and flower constraints added, the average number of LPs solved.

For problems not solved to optimality, we start branching on fractional variables. The same cutting plane method is used as in the root nodes. The performance is recorded in the third block, including the total number of instances required branching. The number of instances solved to optimality before reaching a time upper bound of 500 seconds, (notice this does not include the problems that had already been solved in the cutting plane method), how many "B&C" solutions improved the cutting plane solution, the final gap between the best IP solution and LP lower bound, total running time and the number of branch and bound nodes.

The experiments are done on a Sun Ultra 10 Workstation. The heuristic algorithm is coded in FORTRAN. The branch and cut algorithm is implemented using MINTO 3.0.2 in C and C++.

Table 3.1 shows the result for Type I uniformly distributed vertices. For problems smaller than 200 vertices, the branch-and-cut algorithm can solve each problem to optimality in less than 3 minutes. For problems of 300 vertices the cutting plane algorithm was typically able to solve the Type I problems within 4% of optimality in about 3 minutes. The branch and cut process can improve the optimality to 2% in 500 seconds. We should also notice here that the heuristic algorithm has a pretty good performance in terms of optimality and speed.

Table 3.2 shows the result for the graphs with uniformly distributed edge weights. It has better performance than Type I data. We can solve the problems smaller than 400 vertices to optimality in about 2 minutes. This is different from the performance on CPPMIN in Chapter 2, but is consistent with the observation in Yamada, Takahashi and Kataoka [65]. They offered an explanation, pointing out that the edge costs in Type II instances are distributed in a much wider interval than the edge cost in Type I instances, which results in the existence of a much smaller number of near optimal solutions.

Table 3.3 shows the result for graphs with 121-123 nodes, S range from 7 to 60, so there number of clusters range from 17 to 3. The heuristic solution gets better as S gets bigger. The performance of the cutting plane code also improved slightly as

S gets bigger with smaller gap, less number of iterations and shorter time. But the trend on the performance for branch-and-bound codes is not very clear. It seems that the problem is hardest when $S \approx \sqrt{N}$.

These results are better than the computational results on balance spanning forests problem from Ali and Huang [1], who used a Lagrangian relaxation to compute the lower bounds. For example, they obtained a bound of 3.98% after 1108 iterations for a problem of size 100 node, dividing into 2 cluster of size 50, on a non-complete graph which has an edge density of 15%. It takes about 100 seconds. In comparison, we can solve a problem of around 120 nodes dividing into trees no less than 40 vertices in 13 seconds with a gap of 0.87%. The result is shown in Table 3.3. In the solution process, 83 cycle constraints and 95 flower constraints are added in 28 iterations.

n	81-83	121-123	161-163	201-203	241-243	301-303	421-423
$k = \lfloor \frac{n}{S} \rfloor$	20	30	40	50	60	75	105
Heuristic Alg.							
Gap	3.84%	3.46%	4.72%	3.02%	4.21%	5.46%	5.63%
Time	0.0216	0.0429	0.0729	0.1107	0.1537	0.2376	0.4592
Cutting Plane							
Total Instances	15	15	15	15	15	15	15
Solved exactly	1	1	0	0	1	0	0
Gap	0.61%	1.15%	1.32%	1.90%	1.69%	3.64%	4.81%
Time	9.92	28.10	64.37	99.05	142.50	183.68	403.43
Cycle	35	56	84	103	125	156	187
Flower	311	515	819	1015	1183	1122	1149
LPs solved	54	60	76	81	97	64	77
B&C							
Total Instances	14	14	15	15	14	15	15
Solved exactly	14	14	15	5	7	2	0
Better Solution	9	12	11	14	13	15	6
Gap	0%	0%	0%	0.51%	0.56%	1.88%	4.06
total time	17.85	53.67	171.20	401.86	403.44	492.98	513.18
nodes	15	20	36	46	29	15	3

Table 3.1: Branch-and-Cut Results on MWCF Type I Problems for $S = 4$

n $k = \lfloor \frac{n}{S} \rfloor$	81-83 20	121-123 30	161-163 40	201-203 50	241-243 60	301-303 75
Heuristic Alg.						
Gap	2.67%	3.19%	2.67%	2.94%	2.84%	3.17%
Time	0.0214	0.0424	0.0718	0.1084	0.1551	0.2384
Cutting Plane						
Total Instances	15	15	15	15	15	15
Solved exactly	9	9	7	5	5	1
Gap	0.46%	0.25%	0.24%	0.40%	0.44%	2.08%
Time	5.61	11.52	18.41	26.95	48.28	58.60
Cycle	1	1	1	1	1	1
Flower	134	235	381	477	751	640
LPs solved	24	27	34	32	52	37
B&C						
Total Instances	6	6	8	10	10	14
Solved exactly	6	6	8	10	10	14
Better Solution	4	6	5	9	10	12
Gap	0%	0%	0%	0%	0%	0%
total time	5.84	12.54	24.28	34.71	60.20	104.38
nodes	2	3	7	6	6	11

Table 3.2: Branch-and-Cut Results on MWCF Type II Problems for $S = 4$

3.5.3 Micro-aggregation Problems

Table 3.4 and 3.5 are for micro-aggregation problem data: Type III and IV. Again, the cutting plane method provides a very tight lower bound, and the heuristic gives good feasible solutions. The basic trends and analysis are exactly the same as the those for Type I and Type II data.

3.6 Conclusions

In summary, we have shown that the Spanning Forest Problem with Minimum tree size constraints can be formulated as an IP problem and approximated nicely by a LP relaxation by adding cutting planes. We gave the mathematical formulation of the IP problem, discussed the polyhedral structure of the corresponding polytope, gave the sufficient and necessary conditions for the proposed constraints to be facets. Our computational results showed that with these strong cutting planes, a branch-and-cut algorithm leads consistently to high quality solutions in a reasonable amount

n	121-123	121-123	121-123	121-123	121-123
S	7	10	20	30	40
$k = \lfloor \frac{n}{S} \rfloor$	17	12	6	4	3
Heuristic Alg.					
Gap	2.70%	2.13%	1.20%	1.10%	0.98%
Time	0.0424	0.0424	0.0428	0.0429	0.0428
Cutting Plane					
Total Instances	15	15	15	15	15
Solved exactly	1	1	1	0	0
Gap	1.56%	1.55%	1.07%	0.87%	0.77%
Time	17.46	14.96	12.62	12.93	11.44
Cycle	56	63	70	85	83
Flower	243	1837	119	105	95
LPs solved	38	30	29	32	28
B&C					
Total Instances	14	14	14	15	15
Solved exactly	14	14	13	13	15
Better Solution	14	14	10	14	13
Gap	0%	0%	0.08%	0.18%	0.0%
total time	121.94	222.14	177.15	201.92	61.05
nodes	94	196	250	293	85

Table 3.3: Branch-and-Cut Results on MWCF Type I Problems for $n = 121 - 123$ **and** $S = 7 - 40$

of time.

Comparing with the CPPMIN problem in Chapter 2, we can solve larger spanning tree problems faster. This is partly due to the fact the solution for a spanning tree problem is less dense. Also we have a pretty tight linear relaxation here.

The problem can be extended to a capacitated spanning forest problem: Given a graph $G = (V, E)$ with edge weights c_e, vertex weight w_v, and minimum weight requirement S, The Minimum Weight Capacitated Forest problem looks for a spanning forest of graph G with the sum of the vertex weight in each tree weighting at least S. The objective is to minimize the sum of the edge weight of the spanning forest.

n $k = \lfloor \frac{n}{S} \rfloor$	81-83 20	121-123 30	161-163 40	201-203 50	241-243 60	301-303 75
Heuristic Alg.						
Gap	3.25%	3.57%	4.12%	3.40%	3.33%	5.05%
Time	0.0206	0.0413	0.0703	0.1067	0.1529	0.2357
Cutting Plane						
Total Instances	15	15	15	15	15	15
Solved exactly	4	4	0	0	0	0
Gap	0.98%	1.03%	1.76%	1.24%	1.25%	3.94%
Time	10.75	32.44	75.99	106.53	170.31	221.39
Cycle	36	65	100	136	170	177
Flower	284	505	952	1132	1464	1153
LPs solved	39	49	84	80	108	75
B&C						
Total Instances	11	11	15	15	15	15
Solved exactly	11	11	14	14	12	3
Better Solution	9	9	14	14	14	14
Gap	0%	0%	0.02%	0.01%	0.16%	0.68%
total time	12.84	102.90	338.29	449.86	474.85	918.58
nodes	5	38	68	44	19	20

Table 3.4: Branch-and-Cut Results on MWCF Type III Problems for $S = 4$

We can set it up as the following integer programming problem:

$$\begin{aligned}
\min \quad & \sum_{e\in E} c_e x_e \\
\text{s.t.} \quad & \sum_{e\in E(Q)} x_e \leq |Q| - 1, && \forall Q \subseteq V && (3.12)\\
& \sum_{e\in E(W)} x_e + \sum_{e\in\delta(W)} x_e \geq |W| - \lfloor \frac{\sum_{v\in W} w_v}{S} \rfloor && \forall W \subseteq V && (3.13)\\
& x_e \in \{0,1\}
\end{aligned}$$

This we would like to investigate in the future.

n	81-83	121-123	161-163	201-203	241-243	301-303
$k = \lfloor \frac{n}{S} \rfloor$	20	30	40	50	60	75
Heuristic Alg.						
Gap	3.03%	4.22%	3.47%	3.75%	3.01%	4.70%
Time	0.0210	0.0420	0.0709	0.1077	0.1540	0.2369
Cutting Plane						
Total Instances	15	15	15	15	15	15
Improved	13	15	15	15	13	12
Solved exactly	1	2	0	0	3	0
Gap	0.86%	0.79%	1.38%	1.34%	0.91%	3.60%
Time	10.75	26.69	48.56	87.48	113.29	113.01
Cycle	26	51	55	94	103	85
Flower	259	471	646	953	1014	574
LPs solved	41	40	61	76	82	34
B&C						
Total Instances	14	13	15	15	12	15
Solved exactly	14	13	13	14	5	3
Better Solution	10	7	14	14	12	15
Gap	0%	0%	0.04%	0.19%	0.07%	0.97%
total time	14.07	51.38	141.78	312.57	269.16	471.29
nodes	9	21	22	28	18	13

Table 3.5: Branch-and-Cut Results on MWCF Type IV Problems for $S = 4$

CHAPTER 4
CPPMIN using Branch-and-Price-and-Cut

4.1 Introduction

We restate our CPPMIN problem as the following: given an undirected graph $G(V, E)$ with node weight $a_v = 1, v \in V$, edge cost $c_e, e \in E$, and an integer S, the CPPMIN problem is to find a partition $\Pi = \{P_1, P_2, ..., P_k\}$ of V such that it solves

$$\begin{aligned} \min \quad & \sum_{i=1}^{k} \sum_{e \in E(P_i)} c_e \\ \text{s.t.} \quad & \sum_{v \in P_i} a_v \geq S \qquad i = 1..k \end{aligned} \tag{4.1}$$

Notice the total number of clusters k is not fixed. When the node weight a_v is not fixed at 1, we call it Clique Partition Problem with Generalized Minimum Size Requirement (CPPGMIN). In this chapter, we are going to use branch-and-price to solve just CPPMIN, but the analysis and algorithm can be readily applied to CPPGMIN.

If we change the minimization into maximization and the lower bound on the size constraint into an upper bound, we get the opposite problem, min-cut clustering problem [34], also called clustering problem with knapsack constraints [41] :

$$\begin{aligned} \max \quad & \sum_{i=1}^{k} \sum_{e \in E(P_i)} c_e \\ \text{s.t.} \quad & \sum_{v \in P_i} a_v \leq S \qquad i = 1..k \end{aligned} \tag{4.2}$$

Johnson, Mehrotra and Nemhauser [34] considered this problem with k given, and solved it using branch-and-price, with the column generation subproblem solved as an integer programming problem on the boolean quadratic polytope. They discussed some strong valid inequalities for their column generation subproblem and described their solution strategy with computational results. Mehrotra and Trick [41] improved upon the results in [34] by using a combinatorial method to solve the column generation subproblem. Mehrotra, Johnson and Nemhauser [39] used the min-cut clustering problem to model the political redistricting problem. The knapsack constraint (4.2) is used to balance the population of each district; the objective function is set up to enforce compactness of each district.

Their successful application of column generation on this problem inspired us to try branch-and-price on CPPMIN. But as we shall show later in this chapter, the two changes in the problem changed the property of the problem dramatically, so what was useful in their algorithms in not applicable in our CPPMIN any more.

Column generation has been applied on a series of integer programming problems and reported as a success. Barnhart et al. [5] gave an introduction on formulations of integer programs with a huge number of variables and their solutions by column generation methods. Savelesbergh [59] discussed the branch-and-price algorithm on the generalized assignment problem. Other applications include crew scheduling [62], bin packing and cutting stock problems [63], edge coloring problems [49], graph coloring problems [40]. Wilhelm [64] gave a review on applying column generation methods to solve IP problems with emphasis on formulation issues. Lübbecke and Desrosiers [38] surveyed column generation with emphasis on the dual point of view.

4.2 An Integer Programming Formulation for Column Generation

Now we give the integer programming formulation of CPPMIN in a form that's suitable for column generation. Consider CPPMIN on graph $G = (V, E)$ with minimum size S, a cluster $P \subseteq V$ is feasible if there are at least S vertices in this cluster, i.e. $|P| \geq S$. For each feasible cluster P, we define a binary variable x_P

$$x_P = \begin{cases} 1 & \text{if cluster } P \text{ is used in the solution of CPPMIN} \\ 0 & \text{otherwise} \end{cases}$$

Let $w_P = \sum\limits_{e \in E(P)} w_e$, then CPPMIN can be formulated as the following IP problem, denoted as (MIP),

$$\begin{aligned} \min \quad & \sum_P w_P x_P \\ \text{s.t.} \quad & \sum_{P: v \in P} x_P = 1 \quad \forall v \in V \\ & x_P \in \{0, 1\} \\ & P \in 2^{|V|}, |P| \geq S \end{aligned}$$

Here $2^{|V|} = \{P : P \subseteq V\}$ represents the set of all subsets of V. The first constraint says that every node v must be covered in exactly one of the chosen clusters. In this thesis, we are going to consider only nonnegative edge weights, so we automatically have $x_P \leq 1$. We then relax the integrality constraint, to get the linear relaxation (MLP):

$$\begin{aligned} \min \quad & \sum_P w_P x_P \\ \text{s.t.} \quad & \sum_{P: v \in P} x_P = 1 \quad \forall v \in V && (4.3) \\ & x_P \geq 0 && (4.4) \\ & P \in 2^{|V|}, |P| \geq S && (4.5) \end{aligned}$$

Obviously, there are many feasible clusters P that satisfy (4.5), thus a large number of variables x_P. But since most of them will be 0 in the optimal solution, we do not need to include all of them in the initial formulation. Instead, we can start with a subset of clusters, then add in the necessary ones later using a pricing algorithm. This step is also called column generation since it is similar to the the column generation method for solving LPs with large number of variables. Starting with a subset of feasible clusters $T \subseteq 2^{|V|}$, we solve a restricted (MLP), called (RMLP), where $P \in T$. The optimal solution of (RMLP) is a feasible solution to MLP. The dual values π_v for each constraint in (RMLP) are used to decide whether we need to expand T.

We demonstrate how to make this decision by the following example. Suppose a CPPMIN requires dividing a graph of 3 nodes into clusters bigger than size 2. There is obviously only one feasible solution to this problem, which is to put all 3 vertices in 1 cluster. We use this trivial example here only to demonstrate how we are going to generate columns for CPPMIN. Suppose we have considered three clusters, $P_1 = \{v_1, v_2\}$, $P_2 = \{v_2, v_3\}$, $P_3 = \{v_1, v_3\}$, i.e., $T = \{P_1, P_2, P_3\}$. Now we consider if we need to expand T into $T' = \{P_1, P_2, P_3, P_4\}$, where $P_4 = \{v_1, v_2, v_3\}$. The LP relaxation of (MIP) on T' can be written as the following:

$$\begin{array}{llllllll} \min & w_1x_1 & +w_2x_2 & +w_3x_3 & +w_4x_4 & & \\ \text{s.t.} & x_1 & +x_2 & & +x_4 & = & 1 \\ & & +x_2 & +x_3 & +x_4 & = & 1 \\ & x_1 & & +x_3 & +x_4 & = & 1 \\ & x_i & & & & \geq & 0 \end{array} \tag{4.6}$$

Take π_v, α_v to be the dual variables corresponding to the equality and inequality constraints respectively. The dual problem is

$$\begin{array}{llllllllll} \max & \pi_1 & +\pi_2 & +\pi_3 & & & & & & \\ \text{s.t.} & \pi_1 & & +\pi_3 & +\alpha_1 & & & & = & w_1 \\ & \pi_1 & +\pi_2 & & & +\alpha_2 & & & = & w_2 \\ & & \pi_2 & +\pi_3 & & & +\alpha_3 & & = & w_3 \\ & \pi_1 & +\pi_2 & +\pi_3 & & & & +\alpha_4 & = & w_4 \\ & & & & \alpha_i & & & & \geq & 0 \end{array} \tag{4.7}$$

Let $(\bar{x}_1, \bar{x}_2, \bar{x}_3)$ and $(\bar{\pi}, \bar{\alpha})$ be the optimal solution to the primal and dual pair, when x_4 was not introduced. $(\bar{x}_1, \bar{x}_2, \bar{x}_3, 0)$ is also a feasible solution to (4.6). Only when the reduced cost of x_4, $w_4 - (\bar{\pi}_1 + \bar{\pi}_2 + \bar{\pi}_3)$, is negative, do we need to add in cluster P_4.

The above example demonstrates whether we need to expand T can be determined by solving a minimum node-edge-weighted cluster problem (MINNEWCP) with minimum size constraints. Given a graph $G = (V, E)$ with node weight $-\pi_v$ and edge weight w_{ij}, find a feasible cluster whose total weight (edge weights and node weights all together) is minimized. If the optimal value is non-negative, then there exist no improving clusters. Otherwise, any feasible cluster with a negative objective value provides an improving cluster. A feasible cluster in the case of CPP-MIN needs to satisfy the minimum size constraint, i.e., it has to have at least S vertices.

This process should be repeated until there are no improving clusters. If the optimal solution to the final linear relaxation, (MLP), is an integer solution, then we are done, otherwise we need to use either cutting plane method or branching to force integrality.

4.3 Cutting Planes

The LP relaxation (MLP), however, is not a very good approximation of the original IP problem (MIP). In fact, whenever the number of vertices n is not a multiple of S, there exists a fractional solution to (MLP) that has a better objective value than the optimal integer solution. And the difference is usually very big, resulting in a bad approximation, consequently a large branch-and-bound tree, if we only use branching to force integrality.

This can be illustrated in the example in Figure 4.1 - 4.2. For a problem of size $n = 14$, $S = 4$, the optimal solution is shown in Fig 4.1 with an objective value of 628, but the best LP objective value of its LP relaxation (MLP) is 462, with the corresponding LP solution shown in Fig 4.2. The huge difference between these two solutions comes from the fractional value of the clusters in vertices $\{7, 8, 9, 10, 11, 14\}$, namely from the factional value on clusters: $x\{7, 8, 9, 11\} = 0.5$, $x\{7, 9, 10, 14\} = 0.5$, $x\{8, 10, 11, 14\} = 0.5$. This phenomenon happens in every cluster with more than S vertices, since it can always be replaced by a fractional combination of clusters of size S with a better objective value.

A weak LP relaxation is bad news in integer programming problems in general, especially so in the context of branch-and-price, because one of the major motivations for considering a formulation with a huge number of variables lies in the hope that this new formulation gives a tight LP relaxation. A weak relaxation at the root node seldom leads to successful branch-and-bound process, so we would like to tighten it up.

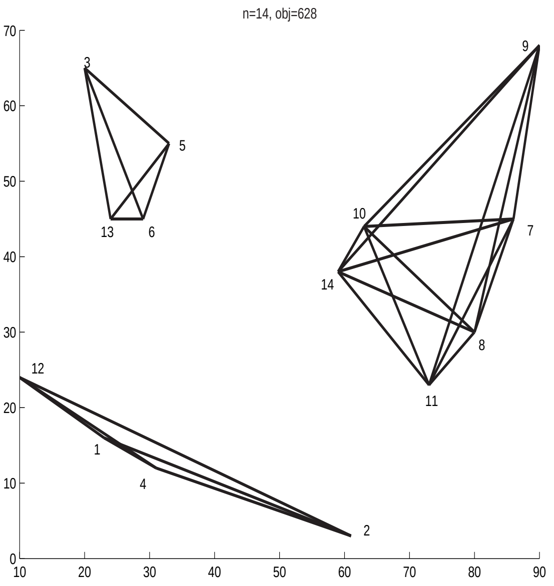

Figure 4.1: Optimal Solution for a CPPMIN Problem of Type I with $n = 14, S = 4$

In this example, further observation shows us the fractional LP solution in Figure 4.2 can be easily cut off if we add in the following constraint $\sum_P x_P \leq 3 = \lfloor 14/4 \rfloor$, which is implied by the requirement that each clusters has to be bigger than $S = 4$ in this example. This is in essence the same as the biggest pigeon constraint in Chapter 2. We added it into the initial formulation of the problem to tighten up the approximation. According the the following theorem, we can further generalize this constraint and add in similar constraints as cutting planes.

Theorem 4.3.1. *Given* $Q \subset V, |Q| < kS$, *inequality* (4.8) *is a valid constraint for (MIP) on* Q.

$$\sum_{P:P\subseteq Q} x_P \leq k - 1 \tag{4.8}$$

The proof is straight forward since when $|Q| < kS$, we can partition Q into at at most $k - 1$ clusters of size bigger or equal to S.

Algorithm 8 gives a heuristic method to look for violated constraints. In this algorithm, we only check the the subsets of vertices generated by combining existing

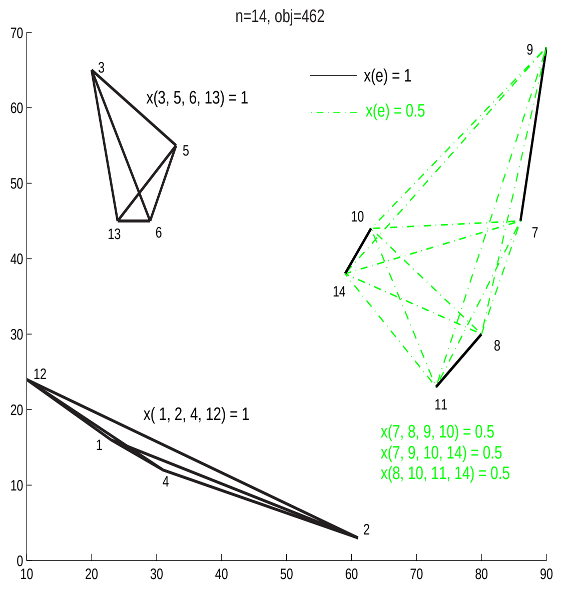

Figure 4.2: (MLP) Solution for a Problem of Type I size n=14, S=4

columns in the master problem, since the clusters corresponding to the columns are between size S and $2S-1$, so the combined subsets are of size between S and $4S-2$. Our empirical results show that checking this subset of clusters already improves the result significantly. Most of the problems get a good LP bound in the root node after these cutting planes are added, so we didn't investigate more sophisticated cutting plane generation methods. When no cutting planes can be found any more, if we still have a fractional optimal solution, we resort to branching.

Algorithm 8 Algorithm for finding violated constraints of type (4.8) for MLP

```
Input:
  $xlp$ - MLP solution
Output:
  $QSET$ - subsets of vertices violating constraint (4.8)
Steps:
  $QSET = \emptyset$
  for each pair of $(i, j)$ s.t. $xlp_i \neq 0$, $xlp_j \neq 0$ do
    Let $P^i$ be the cluster corresponding to $xlp_i$
    Let $P^j$ be the cluster corresponding to $xlp_j$
    Let Q = $P^i \cup P^j$.
    if (4.8) is violated on Q then
      $QSET = \{QSET\} + \{Q\}$
    end if
  end for
```

With these new cutting planes, we write out our restricted (MLP) as (RMLP),

$$\min \quad \sum_{P} w_P x_P$$

$$\text{s.t.} \quad \sum_{P:v\in P} x_P = 1 \qquad \forall v \in V \tag{4.9}$$

$$\sum x_P \leq k \tag{4.10}$$

$$\sum_{P\subseteq Q_i} x_P \leq \lfloor |Q_i|/S \rfloor \qquad i = 1..q \tag{4.11}$$

$$x_P \geq 0 \tag{4.12}$$

$$P \in T \subseteq 2^{|V|}, |P| \geq S \tag{4.13}$$

We separate constraint (4.10) from (4.11) because the former is included in the initial formulation, while the latter is added in dynamically when Q_i is identified using algorithm 8 during the computation. Let π, σ, σ_i, and α_i be the dual variables corresponding to equations (4.9) - (4.12). We can write down the corresponding dual problem:

$$\begin{array}{llllllll}
\max & \sum_{v=1}^{n}\pi_v & +k\sigma & +\sum_{i=1}^{q}\lfloor\frac{|Q_i|}{S}\rfloor\sigma_i & & & & \\
\text{s.t.} & \sum_{v\in P}\pi_v & +\sigma & +\sum_{i:P\in Q_i}\sigma_i & +\alpha_P & = & w_P & \text{for every } P\in T \\
 & & \sigma & & & \le 0 & & \\
 & & & \sigma_i & & \le 0 & & i=1..q \\
 & & & & \alpha_P & \ge 0 & & \text{for every } P\in T
\end{array} \tag{4.14}$$

In general cutting planes are difficult to fit into a column generation framework without complicating the pricing subproblem too much. But it is possible in our situation because the empirical results show that we don't need to add in a lot of cutting planes of this kind to achieve a good approximation of the IP. We have two issues to consider here. One is how to update affected cutting planes after adding in new clusters. The other is how to generate new clusters with these new cutting planes.

The first can be achieved in the following way. We associate a flag with every cutting plane added, and mark it on the vertices affected by this cutting plane. For example, if a subset $Q_q \subseteq V$ is associated with the q^{th} cutting plane, then every vertex in Q_q has a mark q. When new columns are added, we take an intersection of the marks on all the vertices affected by this new column, the result is the constraints that should include this new column.

The second issue is solved in the following way. In the column generation subproblem, we need to either find a cluster P s.t. $\alpha_P = w_P - \sum_{v\in P}\pi_v - \sigma - \sum_{i:P\subseteq Q_i}\sigma_i < 0$, to price into the master problem (MLP), or prove that such a cluster doesn't exist, thus we are at optimality. This is in general not easy, because the value of the last term $\sum_{i:P\subseteq Q_i}\sigma_i$, depends on the the cluster P we found. If $P \not\subseteq Q_i, \forall i = 1..q$, $\sum_{i:P\subseteq Q_i}\sigma_i = 0$, otherwise, $\sum_{i:P\subseteq Q_i}\sigma_i$ may be negative.

In our case, we achieve it in two steps. First we enumerate all the feasible clusters in $Q_i, i = 1..q$, and check if any of them can be price out into the master problem. If no such clusters can be found, we can then ignore the cutting planes in the column generation process, and only look at the clusters P that are not subsets of any Q_i and check if $w_P - \sum\limits_{v \in P} \pi_v - \sigma < 0$. Checking this is the subject of the next section. If no such clusters exist, no columns can be generated, thus we have proved the optimality of the current best solution. Otherwise, we have found a cluster to price in. We should emphasize that this strategy is applicable because in our case, the number of added constraints is relatively small, and their sizes are small too, so we can enumerate all the feasible subsets in $Q_i, i = 1..q$, fairly quickly.

4.4 Generate Improving Clusters

In the second case of the previous discussion, we need to solve the following pricing subproblem, the Minimum Node-Edge-Weighted Cluster Problem (MINNEWCP) with minimum size constraints. Finding a fast and good algorithm to solve this problem is an essential part of the branch-and-price scheme. In this section, we will explain how our approach.

4.4.1 Problem Definition and Quadratic Formulation

We state the problem formally as the following. Given a graph $G = (V, E)$, a weight π_v associated with each vertex $v \in V$, and an edge cost c_{ij} associated with each edge $(i, j) \in E$. The MINNEWCP problem is to select a subset of the vertices $P \subseteq V$ that minimize the difference between the cost of the edges in $E(P)$ and the weight of the vertices in P. In other words, the objective is to minimize the quantity $\sum\limits_{i,j \in P} c_{ij} - \sum\limits_{v \in P} \pi_v$. In our pricing problem we have an additional cardinality constraint on P, $|P| \geq S$.

We can change the objective function in MINNEWCP into maximize $\sum\limits_{v \in P} \pi_v - \sum\limits_{i,j \in P} c_{ij}$. In this equivalent form, this problem is also called the generalized independent set problem, first defined by Hochbaum and Pathria in [31]. Given a graph $G = (V, E)$ and a node weight π_v associated with each vertex $v \in V$, recall the independent set problem is to find a subset of the vertices $P \subseteq V$ of maximum weight $\sum\limits_{v \in P} \pi_v$, such that no edges $e \in E$ has both of its endpoints in P. In the generalized independent set problem, we can regard the cost on the edges as a penalty for two adjacent vertices to be included in P. The standard independent set problem has an edge penalty of infinity for every edge in E, and 0 for edges not in E. When the underlying graph is bipartite, the generalized independent set problem can be solved efficiently by reducing to a minimum $s-t$ cut in a network. Hochbaum and Pathria used this property to solve a forest harvesting optimization problem efficiently in [31]. But in general, the generalized independent set problem is NP hard, since the independent set problem is NP-hard and a polynomial algorithm of the generalized independent set problem is a polynomial algorithm for the independent set problem. It is easy to see that the generalized independent set problem with cardinality constraint $|P| \geq S$ is also NP hard when S value is nontrivial comparing with n. So our pricing subproblem, MINNEWCP with cardinality constraints, is an NP-hard problem.

Now, we formulate the problem as a quadratic programming problem. Define variable y_v for each vertex $v \in V$,

$$y_v = \begin{cases} 1 & \text{if } v \in P \\ 0 & \text{otherwise} \end{cases}$$

Suppose the current RMLP has $\pi_i, i = 1, 2, ...n$, and σ as the dual variables to (4.9) and (4.10) respectively. To incorporate the column generation step into the branch-and-bound framework later, we also suppose each vertex v has a cardinality a_v. For the moment, we assume $a_v = 1$. We can formulate our pricing problem as a binary quadratic problem with linear constraints, called MINNEWCPQP,

$$\begin{aligned} \min \quad & \frac{1}{2} y^T C y - \pi^T y & & \\ \text{s.t.} \quad & a^T y \geq S & & (4.15) \\ & \sum_{v \notin Q_i} a_v y_v \geq 1 & i = 1..q \quad & (4.16) \\ & y_v \in \{0, 1\} & & (4.17) \end{aligned}$$

where each entry c_{ij} in C is the edge weight for edge (i, j), and $c_{ii} = 0$ for $i = 1, \ddot{,} n$. Constraint (4.15) imposes the size constraint of feasible clusters. Constraint (4.16) is from the discussion in the last section, to enforce that the selected cluster must not be included in any $Q_i, i = 1..q$. As discussed in the previous section, if the optimal value of this pricing problem is smaller than σ, then we can generate new columns, otherwise, we have achieved optimality in the master problem.

Even if we relax the binary constraint $y \in \{0,1\}$ into a linear constraint $y \in [0,1]$, the resulting optimization problem with quadratic objective and linear constraints is still not a convex optimization problem to solve, since the edge cost matrix C in the quadratic term is not positive semidefinite. Extensive research has been done on constrained or unconstrained binary quadratic programming. See for example, Barahona, Jünger and Reinelt [3] solve the unconstrained quadratic 0-1 program by converting it to a max-cut problem and solving it using branch-and-cut. Beasley [6] gives a comparison on heuristic algorithms for unconstrained quadratic 0-1 program.

4.4.2 Related Problems

Mehrotra and Trick [41] solved a very similar pricing problem for their knapsack clustering problem. The only difference is that they did a maximization of the same objective function with a knapsack upperbound constraint rather than a lower bound constraint on the size. They proposed an effective combinatorial method to solve it, which leads to the success of the main price-and-branch scheme. The strength of this method lies in a shifting of weight from edges to node to give a close upper bound. This way they can get a good upper bound from a combinatorial analysis rather than solving a LP. But for we can not make use of this weight shifting scheme to get an useful bound for our minimization problem. and the additional constraints (4.16) and (4.15) impose further difficulties in applying a combinatorial method. In an earlier paper, Johnson, Mehrotra and Nemhauser [34] worked on the the same problem as in [41]: min-cut clustering with capacity lower bound. They looked at the subproblem as a integer programming problem and gave some strong valid inequalities for the subproblem. Again because the difference on the objective, these valid inequalities are usually not violated in our pricing problem.

The pricing problem in [41] and [34] is a special case of Quadratic Knapsack

Problem, (QKP), which has attracted great interest recently,

$$\begin{array}{ll} \max & y^T L y \\ \text{s.t.} & a^T y \leq b \\ & y \in \{0,1\}^n \end{array}$$

If we let $L = diag(\pi) - \frac{1}{2}C$, where $diag(\pi)$ is a diagonal matrix with π as the diagonal entries, the objective function in MINNEWCPQP can be written in the same form as the objective function in QKP, But the size constraint remains a lower bound instead of an upper bound. We can get around this by letting $z = e - y$. Then the MINNEWCPQP can be formulated as a QKP with an addtional constraint:

$$\begin{array}{lll} \min & \frac{1}{2}(e-z)^T C(e-z) - \pi^T(e-z) & \\ \text{s.t.} & a^T z \leq n - S & \\ & \sum_{v \in Q_i} a_v z_v \leq |Q_i| - 1 & i = 1..q \\ & z_v \in \{0,1\} & \end{array}$$

We can write the objective function as $\frac{1}{2}e^T Ce - \pi^T e + \max\ z^T L z$, where $L = -\frac{1}{2}C + diag(Ce - \pi)$. In this QKP formulation, most of the entries in L is negative. In the most general form of QKP, the entries in L can be either positive or negative, but most research focuses on the case when the entries in L are nonnegative.

QKP was first studied by Gallo, Hammer and Simeone [18]. They proposed an exact solution where upper bounds are computed using upper planes, which are linear functions of the binary variables which are not smaller than the QKP objective function over the set of feasible QKP solutions. Billionnet and Calmels [7] used a standard branch-and-cut approach to solve the classical equivalent IP formulation of the problem. The problem sizes they solved were up to 40 variables, and they get to optimality within 1%. Caprara, Pisinger and Toth solved it in [8] using an exact branch-and-bound algorithm where upper bounds are computed by considering a Lagrangian relaxation which is solvable through a number of continuous knapsack problems. They reported solutions of instances with up to $n = 400$. Helmberg, Rendl and Weismantel [30] proposed a combined approach which uses cutting planes and semidefinite programming, and allows for the computation of very tight upper bounds.

4.4.3 IP formulation and Boolean Polytope

We follow the standard "linearization" method in quadratic programming literature to convert the quadratic programming problem MINNEWCPQP into an IP problem.

Introduce a new variable $z_{ij} = y_i y_j$ for every edge $(i, j) \in E$.

$$z_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E(P) \\ 0 & \text{otherwise} \end{cases}$$

We can reformulate MINNEWCPQP as the following IP problem, referred to as MINNEWCPIP,

$$\begin{aligned}
\min \quad & -\sum_{i \in V} \pi_i y_i + \sum_{(i,j) \in E} c_{ij} z_{ij} && \\
\text{s.t.} \quad & z_{ij} \le y_i && (4.18)\\
& z_{ij} \le y_j && (4.19)\\
& z_{ij} \ge y_i + y_j - 1 && (4.20)\\
& \sum_{i=1}^{n} a_i y_i \ge S && (4.21)\\
& \sum_{i \notin Q_k} a_i y_i \ge 1 \qquad k = 1..q && (4.22)\\
& z_{ij} \ge 0 && (4.23)\\
& y_i \in \{0,1\}, z_{ij} \in \{0,1\} && (4.24)
\end{aligned}$$

Equations(4.18) and (4.19) ensure that z_{ij} must be zero if either one of y_i or y_j is zero. Equation (4.20) ensures that z_{ij} is one if both y_i and y_j are one. The convex hull of (4.18), (4.19),(4.20), (4.23) (4.24) is called the Boolean Quadratic Polytope, denoted

$$QP^n = conv\{(y,z) \in R^{n(n+1)/2} | (y,z) \text{ satisfy } (4.18),(4.19),(4.20),(4.23),(4.24)\}$$

and its linear relaxation is denoted

$$QPLP^n = \{(y,z) \in R^{n(n+1)/2} | (y,z) \text{ satisfy } (4.18),(4.19),(4.20),(4.23)\}$$

Padberg [52] discussed this boolean quadratic polytope structure and pointed out that the linear relaxation is not tight at all. In fact, $y_i = 0.5, i = 1...n$ and $z_{ij} = 0$ always gives a feasible solution to the linear relaxation. When π_i and c_{ij} are both positive, which is often the case in our problem, this fractional solution usually has a much lower objective value than the IP optimal solution. We summarize some of the results from [52] in the following theorem.

Theorem 4.4.1. *([52], Theorem 2-5)*

(i) *QP^n and QP^n_{LP} are full-dimensional, i.e. $dim(QP^n) = dim(QP^n_{LP}) = n(n+1)/2$.*

(ii) *The inequalities $z_{ij} \geq 0$ define facets of QP^n and QP^n_{LP} for all $1 \leq i < j \leq n$.*

(iii) *The inequalities $z_{ij} \leq x_i$ and $z_{ij} \leq x_j$ define facets of QP^n and QP^n_{LP} for all $1 \leq i < j \leq n$.*

(iv) *The inequalities $z_{ij} \geq y_i + y_j - 1$ define facets of QP^n and QP^n_{LP} for all $1 \leq i < j \leq n$.*

(v) *For any $U \subseteq V$ with $|U| \geq 3$ and integer α, $1 \leq \alpha \leq |U| - 2$, the clique-inequality $\alpha y(U) - z(E(U)) \leq \alpha(\alpha+1)/2$ defines a facet of QP^n.*

(vi) *For any $U \subseteq V$ with $|U| \geq 1$ and $T \subseteq V - U$ with $|T| \geq 2$ the cut-inequality $-y(U) - z(E(U)) + z(\delta(U,T)) - z(E(T)) \leq 0$ defines a facet of QP^n.*

Now we define S-clique polytope for our problem,

$$LCP_S^n = conv\{(y,z) \in R^{n(n+1)/2} | \sum y_i \geq S, (y,z) \in QP^n\}$$
$$LCPLP_S^n = conv\{(y,z) \in R^{n(n+1)/2} | \sum y_i \geq S, (y,z) \in QPLP^n\}$$

Notice inequality (4.21) became $\sum y_i \geq S$, when $a_i = 1$, which is the case in the root node of our branch-and-bound tree.

A closely related problem is the b-clique problem. It is also called the weighted maximal b-clique problem (WCP_b), or edge-weighted clique problem. It is a special case of QKP. The problem is to find, among all complete subgraphs with at most b nodes, a subgraph (clique) for which the sum of the weights of all the nodes and edges in the subgraph is maximal, see Hunting, Faigle and Kern [32]. The corresponding b-clique polytope is given as

$$UCP_b^n = conv\{(y,z) \in R^{n(n+1)/2} | \sum y_i \leq b, (y,z) \in QP^n\}$$
$$UCPLP_b^n = conv\{(y,z) \in R^{n(n+1)/2} | \sum y_i \leq b, (y,z) \in QPLP^n\}$$

Hunting, Faigle and Kern [32] discussed polyhedral structure and facets for the b-clique polytope UCP_b^n. It is easy to see that $dim(UCP_b^n) = n(n+1)/2$ when $b \geq 2$. It follows that our S-polytope LQP_S^n is full dimensional if $S \leq n-1$, i.e. $dim(LCP_S^n) = n(n+1)/2$. So the facet-defining inequalities to LCP_S^n is unique (to within scalar multiplication).

Since $LCP_S^n \subseteq LCP_{S-1}^n \subseteq LCP_1^n = QP^n$, the facets for QP^n are obviously valid for our problem, but they are not necessarily facets any more, similar to the results on UCP_b^n.

Notice that the inequalities $y_i \geq 0$ and $y_i \leq 1$ are not facet-defining for LCP_S^n since they are implied by (4.18) - (4.20) and (4.23). When $a_i = 1, i = 1, \ldots, n$, constraint (4.21) changes to $\sum\limits_{i=1}^{n} y_i \geq S$. This constraint is also not facet defining since the vertices of LCP_S^n satisfying it at equality also satisfy the independent equation $\sum\limits_{(i,j)\in E} z_{ij} = S(S-1)/2$.

Since c_{ij} is always nonnegative, we don't need constraints (4.18)-(4.19). Also we don't need to require z_{ij} to be a binary variable explicitly. Because $a_i \geq 1, i = 1, \ldots, n$, we can drop the a_i in constraint (4.22). So we can reformulate the problem as the following (PRICEIP),

$$\begin{aligned}
\min \quad & -\sum_{i\in V} \pi_i y_i + \sum_{(i,j)\in E} c_{ij} z_{ij} & & \\
\text{s.t.} \quad & z_{ij} \geq y_i + y_j - 1 & & (4.25)\\
& \sum_{i=1}^{n} a_i y_i \geq S & & (4.26)\\
& \sum_{i \notin Q_k} y_i \geq 1 & k = 1..q \quad & (4.27)\\
& z_{ij} \geq 0 & (i,j) \in E \quad & (4.28)\\
& y_v \in \{0,1\} & v \in V \quad & (4.29)
\end{aligned}$$

4.4.4 Other Constraints

We can tighten up the linear relaxation of PRICEIP by generating constraints using the Reformulation-Linearization Technique (RLT) introduced by Sherali and Adams [60] . They are similar to the constraints generated for the linear relaxation of Quadratic Knapsack Problem in Caprara, Pisinger and Toth[8].

For $i = 1..n$, we multiply the size constraint (4.26) by y_i and replace y_i^2 by y_i, $y_i y_j$ by z_{ij}, getting the following valid constraint:

$$\sum_{j \neq i} a_j z_{ij} \geq (S - a_i) y_i \qquad i = 1..n \tag{4.30}$$

This is also called the star inequality in Hunting, Faigle and Kern [32].

We can also multiply (4.26) by $1 - y_i$, and get

$$S y_i + \sum_{j \neq i} a_j y_j - \sum_{j \neq i} a_j z_{ij} \geq S \qquad i = 1..n \tag{4.31}$$

However, our computational results show that these additional constraints, as well as the constraints mentioned in Theorem 4.4.1, do not improve the speed of the branch-and-bound process in solving the pricing problems. They do reduce the number of nodes in the branch-and-bound tree, but the total computation time often increases when these constraints are included either in the initial formulation or as cutting planes. So we just use a pure branch-and-bound procedure to solve PRICEIP directly. It is not necessary to generate the column with the most negative reduced cost each time. Any column with a negative reduced cost would do, so we stop the IP solver as soon as we find a column to price in. But if eligible columns to be priced in do not exist, we have to solve the pricing problem to optimality to prove it.

4.4.5 Heuristic Algorithms

Since solving the pricing problem as an IP problem is an expensive operation, we first use some heuristic algorithms to try to generate columns before solving the problem as an IP problem. Algorithms 9-11 give the three heuristics that we employed. They are applied in the order of 9 to 11. Only when the previous algorithms do not generate any columns to be priced in, would we try the next algorithm.

Algorithm 9 Heuristic Pricing Algorithm I: Enumerate from 2S closest vertices

```
for $i = 1$ to $n$ do
    Find the closest $2S$ vertices to vertex $i$.
    Enumerate all clusters of size $S$ to $2S - 1$ from these $2S$ vertices.
    Put the violating clusters into a column pool.
end for
Add the 10 most violating columns from the column pool, with no more than 10
columns on the same vertex added.
```

Algorithm 10 Heuristic Pricing Algorithm II: Enumerate from constraint clusters

```
for $i = 1$ to $q$ do
    Enumerate all subsets of $Q_i$ between size $S$ and $2S - 1$.
    Put the violating clusters into a column pool.
end for
Add the 10 most violating columns from the column pool, with with no more
than 10 columns on the same vertex added.
```

Algorithm 11 Heuristic Pricing Algorithm III: Greedily find a small node-edge-weighted clique of size at least S

```
Input:
 1: $S$ - minimum cluster size
 2: $c_{ij}$ - edge weight
 3: $\pi_i$ - node weight
 4: $a_i$ - node capacity
Steps:
 5: for $i = 1$ to $N$ do
 6:    $CLIQ = \{i\}$
 7:    Find vertex $v$ s.t. $w(CLIQ, v) = \pi_v - \sum_{e \in \delta(v, CLIQ)} c_e$ is minimum for $v \notin CLIQ$.
 8:    if $|CLIQ| < S$ or $w(CLIQ, v) > 0$ then
 9:       $CLIQ = CLIQ + v$
10:       GOTO 7
11:    else
12:       GOTO 14
13:    end if
14:    Check if $CLIQ$ can be put into the violating column pool.
15:    Do local search near $CLIQ$, see if any columns can be put into violating
       column pool.
16: end for
17: Add the 10 most violating columns from the column pool, with with no more
    than 5 columns on the same vertex added.
```

4.5 Branching

Branching rules in the column generation context are quite different from those in a branch-and-cut scheme. The simple 0-1 branching rule in a branch-and-cut framework would not work here. Suppose a certain column is fixed at zero at a particular branch, we have to prevent this cluster from being regenerated when generating columns in the branch-and-bound nodes in this branch. This, in general, would lead to finding the k^{th} best subproblem solution rather than the optimal solution, which is a much more expensive operation, see Savelsbergh [59], or Ribeiro, Minoux and Penna [55]. So we need a branching rule that can be easily incorporated in the column generation process.

In the set partition model, the Ryan-Foster branching rule [56] is commonly used. In a fractional solution for MLP, we can identify two vertices i and j, such that $i, j \in P_1$ and only one of i, j is in P_2, but both x_{P_1} and x_{P_2} are positive. So we can divide the problem into two branches. In one, vertex i and vertex j must be covered by the same cluster, which can be enforced by just collapsing i and j into a single node in the graph. In the other, vertex i and j must be in different clusters, which can be enforced by changing the weight on edge (i, j) to a very large value. This way, we can impose the branching choices on the pricing subproblems directly rather than adding in additional constraints on the master problem.

Notice this branching strategy actually corresponds to the branching strategy on x_{ij} in the compact branch-and-cut formulation of the problem in Chapter 2. This conforms with the observation in Lübbecke and Desrosiers [38], "to branch on meaningful variable sets". Our most valuable source of information are the original edge variables of the compact formulation; they must be integer, so these are what we branch on. Similarly, the cutting planes that we introduced earlier, corresponds to the pigeon constraint for the compact formulation in Chapter 2.

After branching, our CPPMIN problem changed into CPPGMIN problem. Each combined vertex v has a vertex weight a_v bigger than one, while each original vertex still has a vertex weight of one. In the pricing subproblem, a_v for some vertices may not be one any more. So we are in fact solving a CPPGMIN problem at every branch-and-bound node except the root node.

Even though we didn't add cutting planes in the sub-nodes in our computational results, the constraints can be easily modified to accommodate this change on the cardinality on the vertices, by changing to the following form:

$$\sum_{i:P_i \subseteq Q} x_i \leq k - 1 \tag{4.32}$$

for Q with $\sum\limits_{j:j \in Q} a_j < kS$.

4.6 Stabilization

Recall that the objective function of the pricing subproblem depends on the dual variable of the solution of (4.7). The primal problem (4.6) is very degenerate, since there are $|V|$ rows, but in a feasible integer solution, only $|V|/S = k$ number of $x's$ would be nonzero. Primal degeneracy is well known to cause slow convergence, see Gilmore and Gomory [20]. We are bound to have alternative dual solutions. How to pick the best dual solution, out of all the dual optimal solutions, is a question we have to answer here to reduce oscillations that results in useless moves in the dual space. Various methods has been proposed on this issue, see Lübbecke and Desrosiers [38].

Instead of using those more complicated methods in [38], we simply tries to avoid oscillations by starting with more columns than necessary in our initial problem. To guarantee our initial problem is feasible, we include the columns of a good heuristic solution into the initial columns. To help stabilizing the dual variables, we also include the clusters consisting of the closest $S-1$ and S vertices to each vertex. Our choice of using heuristic Algorithm 9 first in searching for violated columns also helps to avoid generating useless columns.

4.7 Computational Results

In this part, we are going to give the framework of our algorithm and the results of our computational experiments. We first use the heuristic algorithm (Algorithm 4) in Chapter 2 to find a good feasible solution. An initial problem including the clusters in this solution would be a feasible problem. To improve stability of the dual variable value, we also include the clusters composed of the closest S and $S+1$ vertices to each vertex in the initial formulation.

The branch-and-price-and-cut code is implemented using MINTO 3.0.2. The algorithm follows the framework as in MINTO [50] with minor changes. It is illustrated in Algorithm 12. One major difference is in step 4 and 5. Since step 5, solving the pricing problem as an IP, is time consuming, we try to generated cutting planes before step 5. It is only when we can neither generate new columns using other methods, nor generate cutting planes, would we start solving the pricing problem as an IP.

4.7.1 Random Uniform Problems

The data used here are the same data used in Chapter 2 for CPPMIN problems. Again, Type I data is from uniformly distributed points on a square, with the

Algorithm 12 Branch and Price and Cut Framework

```
1: Initialize.
2: Approximately solve the current LP relaxation using CPLEX.
3: Generate columns using heuristic algorithms, if new columns are found goto 2.
4: Check if any cutting planes can be generated, if yes, generate cutting planes and
   goto 2.
5: Generate columns using an IP solver, if new columns are found goto 2.
6: If the gap between the value of the LP relaxation and the value of the incumbent
   integer solution is sufficiently small, STOP with optimality.
7: Try to improve the incumbent solution locally by switching vertices and move
   extra vertices around.
8: Check if any cutting planes can be generated, if yes, generate cutting planes and
   goto 2.
9: Branching.
```

Euclidean distance between each pair of vertices as the edge weights. In Type II, the edge weights are generated directly as a uniformly distributed random numbers between 1 and 100. Table 4.1 and 4.2 list the computational results for Type I and Type II data with $S = 4$, graph size n ranging from 21 to 103.

Each table is divided into three parts to represent the performance of the heuristic algorithm, the root node of the branch-and-price-and-cut algorithm, and the branch-and-price tree. They are labelled as “Heuristic Alg”, “Root Node” and “B&P” respectively.

The first block gives the gap and time for the heuristic approach. The second block corresponds to the result at the end of the root node before branching. The first three rows in the this block give the total number of instances, and out of these many instances, how many are solved to optimality in the root node, how many get better solutions than the heuristic solution. The rest of the data in this block are all average performance, including, the gap at the end of root node, the running time (in seconds), the number of LP's solved, the number of cuts added, the total number of columns at the end of the root node, the number of columns in the initial formulation, the total number of columns found by solving the pricing problem as an IP, the total time spent on solving the pricing problem as an IP and finally the time for solving one pricing problem as an IP.

For problems not solved to optimality, we start branching. No cutting planes are added any more. The performance is recorded in the third block, including the total number of instances requiring branching, the number of instances solved to optimality before reaching an upper bound of 10 for the total number of nodes. (notice this number does not include the problems that are already solved at the root node), how many root node solutions get improved during the branching stage, the final gap between the best IP solution and the LP lower bound, total running time (including the root node time), the number of branch and bound nodes, and the overall number of columns generated in the whole tree.

The experiments are done on a Sun Ultra 10 Workstation. The heuristic algorithm is coded in FORTRAN. The branch and cut algorithm is implemented using MINTO 3.0.2 in C and C++.

Table 4.1 shows the result for Type I data. The small gap at the end of root node indicates a tight relaxation from the combination of column generation and row generation. However, note that the time it takes to solve the root node scales up pretty quickly. For problems smaller than 43 vertices, they are solved to optimality pretty quickly (within 20 seconds). But for problems bigger than 60 vertices, the computation time increases dramatically. This is due to the increasing solution time for the pricing problem, and the increasing number of pricing problems we need to solve. Even though the computation time for solving one pricing problem as an IP is not very big, considering that these pricing problems are NP hard problems, solving them repeatedly sums up to a pretty long time.

Putting the results of branch-and-cut and branch-and-price together in Table 4.3 by combining Table 2.2 in Chapter 2 and Table 4.1, we have a comparison of the two method on Type I problems. we notice that the performances of the two methods, in terms of gap and time, are very similar, branch-and-price-and-cut being slightly better for instances smaller than 43 vertices. When the instances get bigger, the advantage of branch-and-cut over branch-and-price is that the former can stop at any time with a guaranteed value for the gap bound, while the latter has to run a significantly longer time to provide a gap bound, even though it usually returns a smaller gap than branch-and-cut.

Table 4.1 shows the results for Type II problems. As observed earlier in Chapter 2, these are harder partition problems. It's not surprising to see that these problems take a much longer time to solve than Type I data with branch-and-price-and-cut method. But comparing with the results in Table 2.4 from Chapter 2, branch-and-price-and-cut performs better than the branch-and-cut method. We think this is because the edge weights here doesn't satisfy triangle inequalities, resulting in a large number of triangle constraints being added in the branch-and-cut method. But the branch-and-price algorithm doesn't use triangle constraints to guarantee feasibility of the solutions, so it doesn't run into such problems. However this type of problem is still harder than Type I problems, because we don't have good heuristics to generate columns. A lot more columns are generated using IP solver, which is much more time-consuming. Despite this, our computational results still shows that branch-and-price method fits better than branch-and-cut method for problems that violate a lot of triangle constraints.

4.7.2 Micro-aggregation Problems

Tables 4.4 and 4.5 are for Type III and Type IV data, which are constructed to simulate micro-aggregation instances. Again the root node gives a very tight LP approximation for the problem, but it takes a long time to compute. The basic trends and analysis are exactly the same as those for Type I data.

4.7.3 k-way Equipartition Problems

We also ran experiments on k-way equipartition problems, where each cluster has to be of the same size S. Here the total number of vertices n is a multiple of S. The only changes we need to make is on constraint (4.10) in the initial formulation and constraint (4.26) in the pricing subproblem, where the inequalities should be changed to equalities. The same kind of cutting planes are still added in the root node.

Table 4.6 shows the results for Type III exponentially distributed vertices. Comparing with CPPMIN problems of Type III in 4.4, we see that k-way equipartition problems are easier to solve. The computation time scales up much slower than CPPMIN problems. We are able to solve problems of size $n = 100$ in 2 minutes with

n	21-23	41-43	61-63	81-83	101-103
$k = \lfloor \frac{n}{S} \rfloor$	5	10	15	20	25
Heuristic Alg.					
Gap	5.30%	7.57%	7.46%	9.88%	8.21%
Time	0.0056	0.0114	0.0213	0.0337	0.0490
Root Node					
Total Instances	15	15	13	14	4
Solved exactly	11	14	8	5	0
Better Solution	11	14	13	14	4
Gap	0.27%	0.01%	0.98%	1.12%	1.22%
Time	2.41	16.47	188.48	826.04	1791.87
LPs solved	8	15	38	61	72
Total Cuts	2	6	19	35	42
Total Columns	107	200	320	440	528
Initial Columns	65	132	195	260	334
Colunmns by IP	1	2	8	12	18
Total IP Time	2.27	15.80	184.08	798.43	1718.75
Avg IP Time	1.09	5.14	20.28	62.15	92.91
B &P run					
total instances	4	1	5	9	4
Solved exactly	4	1	4	3	0
Better Solution	1	0	3	7	1
Gap	0%	0%	0.18%	0.36%	0.58%
Time	2.90	17.15	273.50	1529.98	4012.68
Nodes	2	1	3	6	10
Final columns	108	200	327	462	560

Table 4.1: Branch-and-Price Results on CPPMIN Type I Problems for $S = 4$

a gap of 1.38%. For the problems up to size $n = 100$, our performance is slightly better than the results in Mitchell [44], which used a branch-and-cut method.

4.8 Conclusions

In this chapter, we discussed solving CPPMIN using branch-and-price scheme. We demonstrated the necessity of cutting planes in this problem, and suggested an effective way of adding cutting planes in the branch-and-price framework. We solved the pricing subproblem as an integer programming problem.

Our computational results showed that branch-and-price performed well on small-size instances (within around 40 vertices), but ran into difficulty for larger

n	21-23	41-43	61-63
$k = \lfloor \frac{n}{S} \rfloor$	5	10	15
Heuristic Alg.			
Gap	9.05%	16.80%	35.23%
Time	0.0059	0.0128	0.0222
Root Node			
Total Instances	15	15	6
Solved exactly	6	0	0
Better Solution	9	9	5
Gap	3.23%	11.84%	20.85%
Time	27.94	322.75	2936.08
LPs solved	39	121	207
Total Cuts	1	3	6
Total Columns	195	426	638
Initial Columns	111	218	316
by IP	27	92	161
Total IP Time	27.33	309.00	2801.17
Avg IP Time	0.96	3.33	17.33
B &P run			
total instances	9	15	6
Solved exactly	9	5	0
Better Solution	5	15	4
Gap	0.00%	2.09%	13.07%
time	31.85	617.74	5504.16
nodes	3	9	10
Final columns	199	460	679

Table 4.2: Branch-and-Price Results on CPPMIN Type II Problems for $S = 4$

problems due to the lack of efficient methods for the pricing subproblem. However, the root algorithm gave a good feasible solution most of the time. After comparing with the results of branch-and-cut method in Chapter 2, we see a similar performance of both methods on all types of data, except for Type II, where branch-and-cut-and-price performs better. In conclusion, we believe branch-and-cut-and-price fits better for instances with a lot of edge weights that do not satisfy triangle constraints. On instances of large sizes, if a strict error bound is needed, branch-and-cut is preferred. If the user is only interested in looking for a good solution without an error bound guarantee, branch-and-price might give a good solution more quickly.

As mentioned earlier the whole algorithm can be extended to partition prob-

n	21-23	41-43	61-63	81-83	101-103
$k = \lfloor \frac{n}{S} \rfloor$	5	10	15	20	25
B&C root					
Gap	1.87%	2.67%	3.02%	3.65%	3.77%
Time	6.99	23.08	58.42	110.58	175.63
B&C run					
Gap	0%	0%	0.30%	2.67%	3.47%
Time	3.25	24.07	157.06	393.70	510.13
B&P root					
Gap	0.27%	0.01%	0.98%	1.12%	1.22%
Time	2.41	16.47	188.48	826.04	1791.87
B&P run					
Gap	0%	0%	0.18%	0.36%	0.58%
Time	2.90	17.15	273.50	1529.98	4012.68

Table 4.3: Comparison of Branch-and-Cut and Branch-and-Price Results on CPPMIN Type I Problems for $S = 4$

lems with knapsack lower bound constraints, where each vertex is given a weight, and each cluster in the solution must be bigger than a certain weight. We would like to extend our work on this in the future.

n	21-23	41-43	61-63	81-83	101-103
$k = \lfloor \frac{n}{S} \rfloor$	5	10	15	20	25
Heuristic Alg.					
Gap	1.24%	5.38%	7.81%	6.44%	6.25%
Time	0.0055	0.0113	0.0215	0.0335	0.0501
Root Node					
Total Instances	15	12	15	13	11
Solved exactly	11	7	6	1	2
Better Solution	6	10	15	13	10
Gap	0.27%	0.59%	0.99%	1.51%	1.99%
Time	4.23	29.32	138.73	419.22	1206.69
LPs solved	14	27	45	54	69
Total Cuts	4	8	22	29	40
Total Columns	120	224	335	435	541
Initial Columns	63	127	199	268	330
by IP	3	7	10	11	15
Total IP Time	3.93	28.00	131.73	376.85	1131.00
Avg IP Time	0.97	3.36	11.56	31.01	70.69
B&P run					
total instances	4	5	9	12	9
Solved exactly	4	4	3	6	4
Better Solution	0	4	5	9	4
Gap	0%	0.13%	0.25%	0.57%	0.57%
time	4.52	39.03	303.52	948.34	2582.47
nodes	2	2	5	7	7
Final columns	120	232	349	460	566

Table 4.4: Branch-and-Price Results on CPPMIN Type III Problems for $S = 4$

n	21-23	41-43	61-63	81-83	101-103
$k = \lfloor \frac{n}{S} \rfloor$	5	10	15	20	25
Heuristic Alg.					
Gap	4.32%	4.57%	5.42%	6.36%	7.41%
Time					
Root Node					
Total Instances	15	15	15	15	3
Solved exactly	11	13	8	5	0
Better Solution	9	13	14	15	3
Gap	1.05%	0.43%	0.73%	1.07%	3.12%
Time	3.29	18.42	127.36	332.23	546.65
LPs solved	11	21	36	49	40
Total Cuts	2	7	16	30	22
Total Columns	113	216	319	424	487
Initial Columns	67	129	199	268	325
by IP	3	4	8	8	6
Total IP Time	3.07	17.50	127.36	332.23	525.33
Avg IP Time	0.73	3.27	14.15	34.23	78.80
B &P run					
total instances	3	2	7	10	3
Solved exactly	3	2	3	2	1
Better Solution	2	2	6	7	3
Gap	0%	0%	0.16%	0.37%	2.15%
time	4.80	31.85	226.91	907.48	2518.76
nodes	2	2	5	6	8
Final columns	117	219	329	449	521

Table 4.5: Branch-and-Price Results on CPPMIN Type IV Problems for $S = 4$

n	40	60	80	100
$k = \lfloor \frac{n}{S} \rfloor$	10	15	20	25
Heuristic Alg.				
Gap	6.05%	3.27%	4.66%	5.85%
Time	0.0106	0.0199	0.0329	0.0458
Root Node				
Total Instances	5	5	5	5
Solved exactly	5	1	3	1
Better Solution	5	4	4	5
Gap	0.00%	0.31%	0.25%	1.38%
Time	19.79	150.33	103.18	118.50
LPs solved	14	57	30	51
Total Cuts	7	45	23	50
Total Columns	180	315	356	466
Initial Columns	127	179	247	319
by IP	1	5	2	0
Total IP Time	19.40	116.00	98.60	118.50
Avg IP Time	9.70	20.17	32.87	69.50
B &P run				
total instances	0	4	2	4
Solved exactly	0	3	1	3
Better Solution	0	1	1	2
Gap	0.00%	0.09%	0.24%	0.31%
time	19.79	233.81	446.60	1182.62
nodes	1	4	3	8
Final columns	180	325	364	508

Table 4.6: Branch-and-Price Results on k-way Equipartition Type III Problems for $S = 4$

CHAPTER 5
Concluding Remarks

5.1 Conclusions

The contribution of this thesis falls in two aspects. First, we studied a class of graph partition problems that has wide applications but has been neglected in the mathematical programming field before, i.e. graph partition problems with minimum size constraints. In particular, we studied the Clique Partition Problems with Minimum Size Constraints (CPPMIN) and the Minimum Weight Constrained Forest (MWCF) problem. Second, we compared the two major solution strategies in solving IP problems on the CPPMIN problem: branch-and-cut vs. branch-and-bound.

For the CPPMIN problem, we first solved it using branch-and-cut. For this method, we investigated the facial structure of the corresponding polytope. We introduced new classes of inequalities for the corresponding polytope. We were able to solve problems on a complete graph of up to $n = 100$ vertices, with a gap of 4% within 500 seconds. Then we solved the same problem using branch-and-price-and-cut. We successfully combined row generation and column generation to give a tight LP relaxation of the problem. We used integer programming to solve the pricing problem, the minimum node-edge-weighted cluster problem, which is a problem of interest by itself. Our branch-and-price algorithm performs well on small-size instances (within around 40 vertices). But more efficient methods for the pricing subproblem need to be designed for branch-and-price to perform successfully on larger problems.

On instances of large sizes, if a strict error bound is needed, branch-and-cut is preferred. If the user is only interested in looking for a good solution, without an error bound guarantee, branch-and-price can be a good alternative. A comparison between the two methods shows that branch-and-price-and-cut is more suitable for instances with edge weights violating triangle constraints.

MWCF was solved successfully using a branch-and-cut algorithm. We investigated the facial structure of the corresponding polytope and found facet-defining inequalities. Our computational results showed that with these strong cutting planes, we can solve problems on complete graphs of up to $n = 300$ vertices, with a gap of 2% within 500 seconds.

5.2 Future Work

Future work can be conducted in two directions: one is to extend the problem into a more general setting, the other is to investigate other methods, such as SDP, in solving these problems.

In Chapter 4, we have already touched on the problem of extending the minimum size constraints into a knapsack constraint, where each node is given a weight, and the final clusters in the partition have to be bigger than a certain weight. The work in Chapters 2 and 3 can also be extended to this problem.

5.2.1 Generalized Problems

We can consider other extensions of the minimum size constraints. For example, a more general problem could be the metric maximum clustering problem with given cluster sizes discussed by Hassin and Rubinstein [28]. Given a complete graph $G = (V, E)$ with edge weights and integers $c_1, ..., c_k$ that sums up to $|V|$, the problem is to find a partition of V into disjoint clusters of size $c_1, ..., c_k$, so as to minimize(or maximize) the sum of weights of edges whose two ends belong to the same cluster. In the uniform case, $c_1 = c_2 = ... = c_k$, this is a k-way equipartition problem. For the general non-uniform case, Hassin and Rubinstein [28] gave an approximation algorithm with error ratio bounded by $\frac{1}{2\sqrt{2}}$, on the maximization case with edge weights satisfying triangle inequalities. This was an improvement on their earlier results in [29]. It would be interesting to see how the IP approach can be used to attack this more general problem.

5.2.2 SDP Formulation

CPPMIN on a graph $G = (V, E)$ with $|V| = n$ can be formulated into a SDP problem in the following way, define the $n \times n$ symmetric matrix X and C as

$$X_{ij} = \begin{cases} 1 & \text{if } i = j \text{ or } i \text{ and } j \text{ are in the same cluster} \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

$$C_{ij} = \begin{cases} 0 & \text{if } i = j \\ c_{ij} & \text{otherwise} \end{cases} \tag{5.2}$$

Our problem can be formulated into a SDP as the following:

$$\begin{aligned} \min \quad & \tfrac{1}{2} C \bullet X \\ \text{s.t.} \quad & diag(X) = 1 \\ & Xe \geq Se \\ & 0 \leq X \leq J \\ & X \succeq 0 \\ & X \in TRI \\ & rank(X) \leq \lfloor \tfrac{n}{s} \rfloor \end{aligned} \tag{5.3}$$

where $TRI = \{X = x_{ij} : x_{ij} + x_{ik} - x_{jk} \leq 1, \forall i, j, k\}$ is the set of matrices satisfying triangle constraints. $J = ee^T$ is the matrix of all ones, $X \succeq 0$ indicates that X is positive semidefinite. If we relax the last constraint, we get the SDP relaxation. The non-negativity $x_{ij} \geq 0$ and the triangle inequalities $X \bullet T_i \leq 1$ do not need to be included in the initial formulation, instead they can be added in as cutting planes, along with Pigeon Constraints $X \bullet P_i \geq b_i$. The advantage of using the SDP relaxation is that the SDP relaxation is usually tighter than an LP relaxation, but the drawback is that the size of the SDP problems that can be solved is fairly limited.

Mitchell [44] compared the SDP cutting plane approach with the LP cutting plane approach on k-way equipartition problem . The computational results showed only a marginal improvement in the lower bound, with a considerable investment of computation time.

It would be interesting to see how this approach compares with the LP relaxation approaches that we discussed in this thesis.

CHAPTER 6
BIBLIOGRAPHY

[1] Chung-Hsing Huang Agha Iqbal Ali. Balanced spanning forests and trees. *Networks*, 21:667–687, 1991.

[2] F. Barahona, M. Grötschel, and A.R. Mahjoub. Facets of the bipartite subgraph polytope. *Mathematics of Operations Research*, 10:340–358, 1985.

[3] F. Barahona, M. Jünger, and G. Reinelt. Experiments in quadratic 0-1 programming. *Mathematical Programming*, 44:127–137, 1989.

[4] Francisco Barahona and Ali Ridha Mahjoub. On the cut polytope. *Mathematical Programming*, 36:157–173, 1986.

[5] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance. Branch-and-price: column generation for solving huge integer programs. *Operations Research*, 46:316–329, 1998.

[6] J. Beasley. Heuristic algorithms for the unconstrained binary quadratic programming problems. Technical report, Management School, Imperial College, London, UK, 1998.

[7] A. Billionnet and F. Camels. Linear programming for the 0-1 quadratic knapsack problem. *European Journal of Operational Research*, 92:310–325, 1996.

[8] A. Caprara, D. Pisinger, and P. Toth. Exact solution of the quadratic knapsack problem. *INFORMS Journal on Computing*, 11:125–137, 1999.

[9] Sunil Chopra. On the spanning tree polyhedron. *Operation Research Letters*, 8:25–29, 1989.

[10] Sunil Chopra and M.R. Rao. The partition problem. *Mathematical Programming*, 59:87–115, 1993.

[11] V. Chvátal. Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Mathematics*, 4:305–337, 1973.

[12] Michele Conforti and M.R.Rao. The equipartition polytope *i* and *ii*. *Mathematical Programming*, 49:49–70, 1990.

[13] Roberto Cordone and Francesco Maffioli. On the complexity of graph tree partition problems. *Discrete Applied Mathematics*, 134, 2004.

[14] J. Domingo-Ferrer and J. M. Mateo-Sanz. Practical data-oriented microaggregation for statistical disclosure control. *IEEE Transactions on Knowledge and Data Engineering*, 14(1):189–201, 2002.

[15] J. Edmonds. Matroids and the greedy algorithm. *Mathematical Programming*, 1:127–136, 1971.

[16] Jack Edmonds and Ellis L. Johnson. Matching: A well-solved class of integer linear programs. In R. K. Guy, H. Hanani, N. Sauer, and J. Schonheim, editors, *Proceedings of the Calgary International Conference on Combinatorial Structures and their Applications*, pages 89–92. Gordon and Breach, New York, London, Paris, 1970.

[17] D.R. Fulkerson. Blocking and anti-blocking pairs of polyhedra. *Mathematical Programming*, 1:168–194, 1971.

[18] G. Gallo, P.L. Hammer, and B. Simeone. Quadratic knapsack problem. *Mathematical Programming*, 12:132–149, 1980.

[19] Michel X. Geomans and David P. Williamson. Approximating minimum-cost graph problems with spanning tree edges. *Operations Research Letters*, 16:183–189, 1994.

[20] P.C Gilmore and R.E. Gomory. A linear programming approach to the cutting stock problem – part *ii*. *Oper. Res.*, 11:863–888, 1963.

[21] R.E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64:275–278, 1958.

[22] R.E. Gomory. An algorithm for integer solutions to linear programs. In *Recent Advances in Mathematical Programming*, pages 269–302. McGraw-Hill, 1963.

[23] M. Gröschel and Y. Wakabayashi. A cutting plane algorithm for a clustering problem. *Mathematical Programming*, 45:59–96, 1989.

[24] M. Gröschel and Y. Wakabayashi. Facets of the clique partitioning polytope. *Mathematical Programming*, 47:367–387, 1990.

[25] Nili Guttmann-Beck and Shlomi Rubinstein. Approximation algorithms for minimum tree partition. *Discrete Applied Mathematics*, 87(1-3):117–137, 1998.

[26] Stephen Lee Hansen and Sumitra Mukherjee. A polynomial algorithm for optimal microaggregation. *IEEE transactions on Knowledge and Data Engineering*, 15(1):1043–1044, January 2003.

[27] Refael Hassin and Shlomi Rubinstein. Approximation algorithms for maximum dispersion. *Operations Research Letters*, 21:133–137, 1997.

[28] Refael Hassin and Shlomi Rubinstein. Robust matchings. *SIAM Journal on Discrete Mathematics*, 15:530–537, 2002.

[29] Refael Hassin and Shlomi Rubinstein. Approximation algorithms for the metric maximum clustering problem with given cluster size. *Operations Research Letters*, 31(3):179–184, May 2003.

[30] C. Helmberg, F. Rendl, and R. Weismantel. Quadratic knapsack relaxations using cutting planes and semidefinite programming. In W. H. Cunningham, S. T. McCormick, and M. Queyranne, editors, *Integer Programming and Combinatorial Optimization, Lecture Notes in Computer Science*, volume 1084, pages 175–189. Springer, 1996.

[31] Dorit S. Hochbaum and Anu Pathria. Forest harvesting and minimum cuts: A new approach to handling spatial constraints. *Forest Science*, 43(4):544–554, November 1997.

[32] M. Hunting, U. Faigle, and W. Kern. A lagrangian relaxation approach to the edge-weighted clique problem. *European Journal of Operational Research*, 131:119–131, May 2001.

[33] Celina Imielińska, Bahman Kalantari, and Leonid Khachiyan. A greedy heuristic for a minimum-weight forest problem. *Operation Research Letters*, 14:65–71, September 1993.

[34] Ellis L. Johnson, Anuj Mehrotra, and George L. Nemhauser. Min-cut clustering. *Mathematical Programming*, 62:133–151, 1993.

[35] Michael Jünger, Gerhard Reinelt, and Stefan Thienel. Practical problem solving with cutting plane algorithms in combinatorial optimization. *Combinatorial Optimization: DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 111–152, 1995.

[36] Eva K. Lee and John. E. Mitchell. Branch-and-bound methods for integer programming. In Christodoulos A. Floudas and Panos M. Pardalos, editors, *Encyclopedia of Optimization*. Kluwer Academic Publisher, August 2001.

[37] A. Lisser and F. Rendl. Graph partitioning using linear and semidefinite programming. *Mathematical Programming*, 95(1):91–101, 2003.

[38] Marco E. Lübbecke and Jacques Desrosiers. Selected topics in column generation. Technical Report 008-2004, Braunschweig University of Technology, December 2002.

[39] Anuj Mehrotra, Ellis L. Johnson, and George L. Nemhauser. An optimization based heuristic for political districting. *Management Science*, 44(8):1100–1114, August 1998.

[40] Anuj Mehrotra and Michael A. Trick. A column generation approach for graph coloring. *INFORMS J. Comput.*, 8:344–354, 1996.

[41] Anuj. Mehrotra and Michael. A. Trick. Cliques and clustering: a combinatorial approach. *Operations Research Letters*, 22:1–12, 1998.

[42] J. E. Mitchell. Computational experience with an interior point cutting plane algorithm. *SIAM Journal on Optimization*, 10(4):1212–1227, 2000.

[43] J. E. Mitchell. Branch-and-cut algorithms for integer programming. In Christodoulos A. Floudas and Panos M. Pardalos, editors, *Encyclopedia of Optimization*. Kluwer Academic Publisher, August 2001.

[44] J. E. Mitchell. Branch-and-cut for the k-way equipartition problem. Technical report, Rensselaer Polytechnic Institute, 2001.

[45] J. E. Mitchell. Cutting plane algorithms for integer programming. In Christodoulos A. Floudas and Panos M. Pardalos, editors, *Encyclopedia of Optimization*. Kluwer Academic Publisher, August 2001.

[46] J. E. Mitchell. Cutting plane algorithms for combinatorial optimization problems. In Panos M. Pardalos and Mauricio G. C. Resende, editors, *Handbook of Applied Optimization*. Oxford University Press, 2002.

[47] J. E. Mitchell. Realignment in national football league: Did they do it right. *Naval Research Logistics*, 50(7):683–701, 2003.

[48] J. E. Mitchell and B. Borchers. Solving linear ordering problems with a combined interior point/simplex cutting plane algorithm. In H. L. Frenk *et al.*, editor, *High Performance Optimization*, pages 349–366. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2000.

[49] George L. Nemhauser and Sungsoo Park. A polyhedral approach to edge coloring. *Operations Research Letters*, 10:315–322, 1991.

[50] George L. Nemhauser, Martin W. P. Savelsbergh, and Gabriele C. Sigismondi. MINTO, a mixed INTeger optimizer. *Operations Research Letters*, 15:47–58, 1994.

[51] George L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, 1999.

[52] Manfred Padberg. The boolean quadric polytope: Some characteristics, facets and relatives. *Mathematical Programming*, 45:139–172, 1989.

[53] Manfred W. Padberg and Laurence A. Wolsey. Trees and cuts. *Annals of Discrete Mathematics*, 17:511–517, 1983.

[54] Svatopluk Poljak and Zsolt Tuza. Maximum cuts and largest bipartite subgraphs. In William Cook, László Lovász, and Paul Seymour, editors, *Combinatorial Optimization: Papers from the DIMACS Special Year*, volume 20 of *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science*. AMS, 1995.

[55] C.C. Ribeiro, M. Minoux, and M.C. Penna. An optimal column-generation-with-ranking algorithm for very large scale set partitioning problems in traffic assignment. *European J. Oper. Res.*, 41:232–239, 1989.

[56] D. M. Ryan and B. A. Foster. An integer programming approach to scheduling. In A. Wren, editor, *Computer Scheduling of Public Transport Urban Passenger Vehicle and Crew Scheduling*, pages 269–280. Amsterdam, North-Holland, 1981.

[57] Gordon Sande. Exact and approximate methods for data directed microaggregation in one or more dimensions. *International Journal of Uncertainty,Fuzziness and Knowledge-Base Systems*, 10(5), 2002.

[58] Gordon Sande. Personal communication, 2003.

[59] Martin Savelsbergh. A branch-and-price algorithm for the generalized assignment problem. *Operations Research*, 45:831–841, 1997.

[60] H.D. Sherali and W.P. Adams. A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. *SIAM Journal on Discrete Mathematics*, 3(3), August 1990.

[61] Michael Sipser. *Introduction to Theory of Computation.* PWS publishing Company, 1996.

[62] P. H. Vance, C. Barnhart, E. L. Johnson, and G. L. Nemhauser. Airline crew scheduling: A new formulation and decomposition algorithm. *Operations Research*, 45:188–200, 1997.

[63] F. Vanderbeck. Computational study of a column generation algorithm for bin packing and cutting stock problems. *Mathematical Programming*, 86(3):565–594, 1999.

[64] Wilbert E. Wilhelm. A technical review of column generation in integer programming. *Optimization and Engineering*, 2:159–200, 2001.

[65] Takeo Yamada, Hideo Takahashi, and Seiji Kataoka. A branch-and-bound algorithm for the mini-max spanning forest problem. *European Journal of Operational Research*, 101:93–103, 1997.