Branch-and-Price-and-Cut on the Clique Partitioning Problem with Minimum Clique Size Requirement [1]

Xiaoyun Ji, John E. Mitchell

Department of Mathematical Sciences,
Rensselaer Polytechnic Institute, Troy, NY 12180, USA
jix@rpi.edu, mitchj@rpi.edu

Abstract

Given a complete graph $K_n = (V, E)$ with edge weight c_e on each edge, we consider the problem of partitioning the vertices of graph K_n into subcliques that have at least S vertices, so as to minimize the total weight of the edges that have both endpoints in the same subclique. In this paper, we consider using the branch-and-price method to solve the problem. We demonstrate the necessity of cutting planes for this problem and suggest effective ways of adding cutting planes in the branch-and-price framework. The NP hard pricing problem is solved as an integer programming problem. We present computational results on large randomly generated problems.

Key words: graph partition; branch-and-price; clustering; micro-aggregation;

1 Introduction

In this paper, we solve the Clique Partitioning Problem with Minimum clique size constraints (CPPMIN): given an undirected complete 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 $\Pi = \{P_1, P_2, \ldots, P_k\}$ of V that solves

[1] Research supported in part by NSF grant number DMS-0317323.

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

Each partition P_i is called a *cluster*. A cluster with edges connecting every pair of vertices in the cluster is a clique. Since all the clusters in this complete graph are also cliques, we call this problem *clique partitioning problem* [10]. We consider the complete graph here because problems on a non-complete graph can be easily converted to a problem on a complete graph with some zero edge weights. We also only consider the case when c_e is nonnegative. Notice the total number of clusters k is not fixed in the problem, rather the minimum size for each cluster is given as S. $E(P_i)$ denotes the edges with both endpoints in P_i.

We can generalize CPPMIN by changing the node weight a_v to values other than 1. It is then called Generalized Clique Partition Problem with Minimum Size Requirement (GCPPMIN). In this paper, we restrict ourselves to CPPMIN, but the analysis and algorithm can be readily applied to GCPPMIN.

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 [17], also called clustering problem with knapsack constraints [21] :

$$\begin{aligned} \max \quad & \sum_{i=1}^{k} \sum_{e \in E(P_i)} c_e \\ \text{s.t.} \quad & \sum_{v \in P_i} a_v \leq S \qquad i = 1, \ldots, k \end{aligned} \tag{2}$$

Johnson, Mehrotra and Nemhauser [17] 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 [21] improved upon the results in [17] by using a combinatorial method to solve the column generation subproblem. This model is used for a political redistricting problem by Mehrotra, Johnson and Nemhauser [19]. In this application, the knapsack constraint (2) provides a balance on the population of each district, and the objective function is set 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 the differences in these two problems, i.e., minimization vs maximization in the objective, lower bound vs upper bound in the constraint, lead to almost opposite properties for the two problems. Therefore what proved useful in their method, in particular their efficient combinatorial algorithm for solving the pricing problem, is not applicable in our CPPMIN here.

Another closely related problem is the k-way equipartition problem, which solves

$$\begin{aligned} \min \quad & \sum_{i=1}^{k} \sum_{e \in E(P_i)} c_e \\ \text{s.t.} \sum_{v \in P_i} & a_v = S \qquad i = 1, \ldots, k \end{aligned} \tag{3}$$

Here each cluster contains exactly S vertices. An integer programming problem in the framework of branch-and-cut is set up by Mitchell [22]. There, he discussed the corresponding polyhedral structure, facet-defining constraints in detail. The resulting algorithm is used to solve the NFL team alignment problem in [23]. Ji and Mitchell [16] later considered the same problem but within a branch-and-price framework. In addition to sports team alignment problems, they also applied it on micro-aggregation problems that arises in processing public statistical data. This paper is an extension from [16] in the sense that it is applying the branch-and-price framework on a more general problem.

When the number of vertices n is not a multiple of k, a k-way equipartition is not possible. One way to relax the problem is to solve the k-way partition problem, which requires to divide the graph into no more than k clusters. Chopra and Rao [7] investigated this problem on a general graph. A recent application is on the multiple disposal facilities and multiple inventory locations rollon-rolloff vehicle routing problem by Baldacci, Bodin and Mingozzi[1]. They proposed a very effective exact method based on a bounding procedure that combines different relaxations of the problem.

The other 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 our CPMMIN problem. We have also investigated the branch-and-cut method for solving the CPPMIN problem. The result is summarized in [15]. By applying two different methods on the same problem, we hope to provide a comparison of the two and gain some insight into the pros and cons for each method.

Column generation is a classic method that was originally used to solve Linear

Programming problems with large number of variables. Since its adaptation to integer programming more than a decade ago, it has been reported as a success on many difficult IP problems, such as the generalized assignment problem [28] , crew scheduling [30], bin packing and cutting stock problems [31], edge coloring problems [24], graph coloring problems [20], etc. For a general introduction, one can refer to Barnhart et al. [3]. Wilhelm [32] gave a review with emphasis on formulation issues. Lübbecke and Desrosiers [18] surveyed column generation with emphasis on the dual point of view.

2 An Integer Programming Formulation for Column Generation

Now we give the integer programming formulation of CPPMIN in a form that is suitable for column generation. We 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 $w_P = \sum_{e \in E(P)} w_e$, 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 equality constraint says that every node v must be covered in exactly one of the chosen clusters. In this paper, we 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):

$$\begin{aligned} \min \quad & \sum_P w_P x_P \\ \text{s.t.} \quad & \sum_{P: v \in P} x_P = 1, \quad \forall v \in V & (4) \\ & x_P \geq 0 & (5) \\ & P \in 2^V, |P| \geq S. & (6) \end{aligned}$$

There are $\sum_{i=S}^{n} \binom{n}{i}$ Ps that satisfy (6), thus the same number of variables x_P. But it is not possible, nor necessary to include all these variables in the initial formulation, since most of them take the value zero in the optimal solution anyway. Instead, we can start with a subset of clusters, then add in the needed ones later using a pricing algorithm. This pricing step is also called column generation since it is similar to 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 price in a new variable by a trivial example. A graph of 3 vertices is required to be partitioned into clusters containing no less than two vertices. There is obviously only one feasible solution to this problem, which is to put all three vertices in one cluster. But there are four feasible clusters to consider, $P_1 = \{v_1, v_2\}$, $P_2 = \{v_2, v_3\}$, $P_3 = \{v_1, v_3\}$ and $P_4 = \{v_1, v_2, v_3\}$. Suppose we have considered three clusters P_1, P_2 and P_3, i.e., $T = \{P_1, P_2, P_3\}$. Now we need to decide if we should expand T into $T' = \{P_1, P_2, P_3, P_4\}$. The LP relaxation of (MIP) on T' can be written as the following:

$$
\begin{array}{lrrrrrll}
\min & w_1 x_1 & +w_2 x_2 & +w_3 x_3 & +w_4 x_4 & & & \text{dual variables} \\
\text{s.t.} & x_1 & +x_2 & & +x_4 & = 1 & \rightarrow \pi_1 & \\
& & +x_2 & +x_3 & +x_4 & = 1 & \rightarrow \pi_2 & (7) \\
& x_1 & & +x_3 & +x_4 & = 1 & \rightarrow \pi_3 & \\
& x_i & & & & \geq 0 & &
\end{array}
$$

Let π_v be the dual variables corresponding to the equality constraint. The dual problem is

$$
\begin{array}{lrrrrll}
\max & \pi_1 & +\pi_2 & +\pi_3 & & & \text{primal variables} \\
\text{s.t.} & \pi_1 & & +\pi_3 & \leq & w_1 & \rightarrow x_1 \\
& \pi_1 & +\pi_2 & & \leq & w_2 & \rightarrow x_2 \qquad (8) \\
& & \pi_2 & +\pi_3 & \leq & w_3 & \rightarrow x_3 \\
& \pi_1 & +\pi_2 & +\pi_3 & \leq & w_4 & \rightarrow x_4
\end{array}
$$

Let $(\bar{x}_1, \bar{x}_2, \bar{x}_3)$ and $\bar{\pi}$ 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 (7).

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 that we can decide whether to expand T or not 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 CPPMIN 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 a cutting plane method or branching to force integrality.

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 enforce integrality.

This can be illustrated in the example in Figure 1 - 2. A problem of size $n = 14$, $S = 4$ is shown, the edge weights are the Euclidean distance between the vertices on the two ends of the edge. The optimal solution is shown in Fig 1 with an objective value of 628, but the best LP objective value of the LP relaxation (MLP) is 462, with the corresponding LP solution shown in Fig 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 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 lower objective value.

We would like to mention here that a similar gap between the MIP and its LP relaxation is also present in the case of k-way equipartition, although it may not happen in every instance as it does here. For an example and more details on the case of k-way equipartition, see Ji and Mitchell [16].

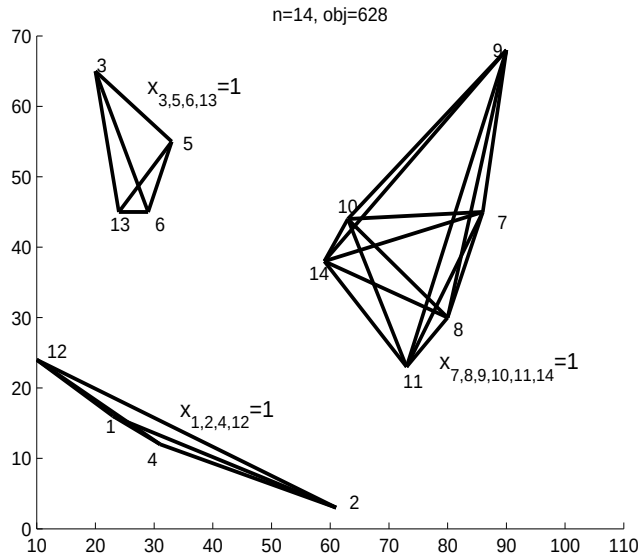

Fig. 1. Optimal Solution for a CPPMIN Problem of Type I with $n = 14, S = 4$

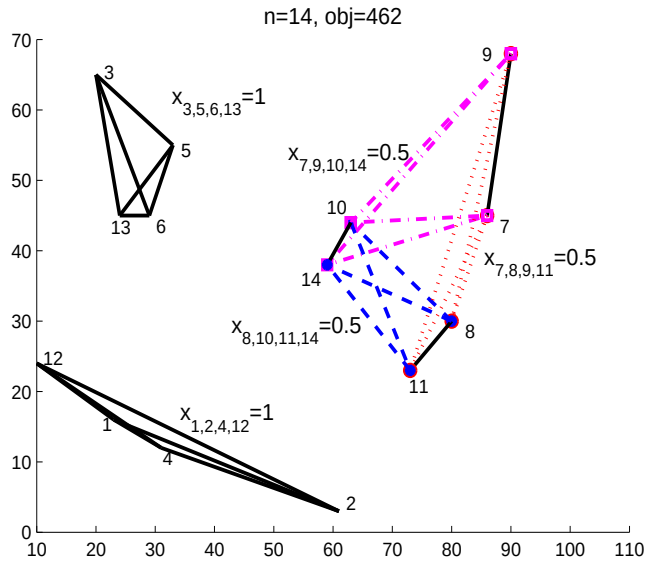

Fig. 2. (MLP) Solution for a Problem of Type I size n=14, S=4

A weak LP relaxation at the root node seldom leads to a successful branch-and-bound process, especially 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.

To tighten up the relaxation, we make use of the following observation. The fractional LP solution in Figure 2 can be easily cut off if we add in the constraint $\sum_{P \subseteq 7,8,9,10,11,14} x_P \leq 1 = \lfloor 6/4 \rfloor$, which is implied by the requirement that

each cluster has to contain at least $S = 4$ vertices. Generalizing this example, we get the following theorem:

Theorem 1 *Given $Q \subseteq V, |Q| < qS$, inequality* (9) *is a valid constraint for (MIP) on Q.*

$$\sum_{P:P\subseteq Q} x_P \leq q - 1 \tag{9}$$

The proof is straight forward since when $|Q| < qS$, we can partition Q into at most $q - 1$ clusters that contain at least S vertices.

This constraint relates to the Pigeon constraint in Ji and Mitchell [15] for the branch-and-cut formulation of the CPPMIN problem.

To find a violated constraint from a fractional solution, we first observe that a possible candidate Q must have the following property: $Q \subseteq \bar{Q} = \{v| \exists P \text{ s.t. } v \in P \text{ and } 0 < x_p < 1\}$. This property significantly reduces the searching space for violated constraints, but it is still inefficient to enumerate and check all possible subsets of $\bar{Q}$. In stead, we use a simply heuristic, Algorithm 1, to check only those subsets generated by combining the corresponding columns of two fractional variables in the current optimal (MLP) solution. Since the clusters corresponding to each column are between size S and $2S - 1$, the combined subsets are between size $S + 1$ and $4S - 2$. Therefore our constraint is relatively small. Our empirical results show that this subset of clusters captures most of the important cutting planes of this type. A much better LP bound in the root node is achieved with these cutting planes added, so we didn't try more sophisticated cutting plane generation methods. When no more cutting planes can be found, if we still have a fractional optimal solution, we resort to branching without cutting planes added in the lower nodes.

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

$$\begin{aligned}
\min \quad & \sum_P w_P x_P & & \\
\text{s.t.} \quad & \sum_{P:v\in P} x_P = 1 & \forall v \in V & \quad (10)\\
& \sum_P x_P \leq k & & \quad (11)\\
& \sum_{P\subseteq Q_i} x_P \leq \lfloor |Q_i|/S \rfloor & i = 1..q & \quad (12)\\
& x_P \geq 0 & & \quad (13)\\
& P \in T \subseteq 2^V, |P| \geq S. & & \quad (14)
\end{aligned}$$

Algorithm 1 Algorithm for finding violated constraints of type (9) for MLP

Input:

xlp - MLP solution

Output:

$QSET$ - subsets of vertices violating constraint (9)

Steps:

$QSET = \emptyset$

for each pair of (i, j) s.t. $0 < xlp_i < 1$ and $0 < xlp_j < 1$ **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 (9) is violated on Q **then**

$QSET = \{QSET\} \cup \{Q\}$

end if

end for

We separate constraint (11) from (12) because the former is included in the initial formulation, while the latter is added in dynamically when Q_i is identified using algorithm 1 during the computation. Let π_ν, σ and σ_i be the dual variables corresponding to equations (10) - (12). We write down the corresponding dual problem:

$$\begin{array}{lrcll} \max \sum\limits_{v=1}^{n} \pi_v & +k\sigma & + \sum\limits_{i=1}^{q} \lfloor \frac{|Q_i|}{S} \rfloor \sigma_i & & \\ \text{s.t.} \sum\limits_{v \in P} \pi_v & +\sigma & + \sum\limits_{i:P \subseteq Q_i} \sigma_i & \leq w_P & \text{for every } P \in T \\ & \sigma & & \leq 0 & \\ & & \sigma_i & \leq 0 & i = 1..q \end{array} \tag{15}$$

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 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 columns. The other is how to generate new columns 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. $\sum_{v\in P}\pi_v+\sigma+\sum_{i:P\subseteq Q_i}\sigma_i > w_P$, to price into the master problem (MLP), or prove that such a cluster doesn't exist, therefore we are at optimality. Complication arises with the last term $\sum_{i:P\subseteq Q_i}\sigma_i$: its value depends on the cluster P we found. If $P \not\subseteq Q_i, \forall i = 1..q$, then $\sum_{i:P\subseteq Q_i}\sigma_i = 0$; otherwise, $\sum_{i:P\subseteq Q_i}\sigma_i$ may be negative.

Accordingly we separate the process into two steps. First we consider the case when $P \subseteq Q_i$, since Q_i is small, we can enumerate all feasible subsets of Q_i, and check if any of them can be priced out into the master problem. If no such clusters can be found, we continue to the second step: 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 $w_P - \sum_{v\in P}\pi_v - \sigma < 0$. 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 q is relatively small, and their sizes are small too, so we can enumerate all the feasible subsets in Q_i, $i = 1, \ldots, q$, fairly quickly.

4 Generate Improving Clusters

In the second step 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 our approach.

4.1 Problem Definition and Quadratic Formulation

The MINNEWCP problem is stated formally as the following: given a graph $G = (V, E)$, a vertex 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 $\sum_{i,j\in P} c_{ij} - \sum_{v\in P}\pi_v$. In our pricing problem we have an additional cardinality constraint $|P| \geq S$.

We can change the objective function in MINNEWCP into maximizing $\sum_{v\in P}\pi_v -$

$\sum_{i,j \in P} c_{ij}$. In this equivalent form, the problem is also called the generalized independent set problem, first defined by Hochbaum and Pathria in [12]. 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 $\sum_{v \in P} \pi_v$, 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, while here we have a finite value for the penalties. 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 [12]. 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.

We start by formulating 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 (10) and (11) respectively. To incorporate the column generation step into a branch-and-bound framework later, we also suppose each vertex v has a cardinality a_v. At the root node, we assume $a_v = 1$. We can formulate our pricing problem as a binary quadratic problem with linear constraints, called MINNEWCPQP,

$$\begin{aligned}
\min \quad & \frac{1}{2} y^T C y - \pi^T y & & \\
\text{s.t.} \quad & a^T y \geq S & & (16) \\
& \sum_{v \notin Q_i} a_v y_v \geq 1 & i = 1, \ldots, q \quad & (17) \\
& y_v \in \{0, 1\} & & (18)
\end{aligned}$$

where each entry c_{ij} in C is the edge weight for edge (i, j), and $c_{ii} = 0$ for $i = 1 \ldots n$. Constraint (16) imposes the size constraint of feasible clusters.

Constraint (17) is from the discussion in the last section, to enforce that the selected cluster can not be a subset of any $Q_i, i = 1 \dots q$. 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 convex, 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 [2], who solve the unconstrained quadratic 0-1 program by converting it to a max-cut problem and solving it using branch-and-cut. Beasley [4] gives a comparison on heuristic algorithms for unconstrained quadratic 0-1 program.

4.2 Related Problems

The pricing problem that Mehrotra and Trick solved in [21] for their knapsack clustering problem has a similar structure as MINNEWCP. The difference is that they did a maximization of the same objective function with a knapsack upper bound constraint while we have a minimization problem with a lower bound constraint on the size. They proposed an effective combinatorial method to solve the pricing problem, which leads to the success of the main branch-and-price scheme. The strength of their method lies in a shifting of weight from edges to vertices to give a close upper bound. This way they can get a good upper bound from a combinatorial analysis rather than solving a LP. Johnson, Mehrotra and Nemhauser [17] worked on the same problem as in [21]: 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.

Just as the max-cut and min-cut problems differs from NP to P, the change of the objective from maximization to minimization in our pricing problem prevented us from using any results from the above literature. In addition, constraints (16) and (17) impose further difficulties in adopting a combinatorial approach.

The pricing problem in [21] and [17] is a special case of Quadratic Knapsack Problem, (QKP), which has attracted great interest recently, see [8], [5], etc.

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

By the change of $z = 1 - y$, We can formulate the MINNEWCPQP as a QKP with an additional constraint. But in the formulation, most of the entries in L would be negative. In the most general form of QKP, the entries in L can be either positive or negative, but the current QKP research mostly focuses on the case when the entries in L are nonnegative. We could not find computational results on QKP with negative entries in the literature.

4.3 IP formulation and Boolean Polytope

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

$$
\begin{aligned}
\min \quad & -\sum_{i \in V} \pi_i y_i + \sum_{(i,j) \in E} c_{ij} z_{ij} & & \\
\text{s.t.} \quad & \sum_{i=1}^{n} a_i y_i \geq S & & (19) \\
& \sum_{i \notin Q_k} a_i y_i \geq 1 & k = 1..q \quad & (20) \\
& z_{ij} \leq y_i & & (21) \\
& z_{ij} \leq y_j & & (22) \\
& z_{ij} \geq y_i + y_j - 1 & & (23) \\
& z_{ij} \geq 0 & & (24) \\
& y_i \in \{0,1\}, z_{ij} \in \{0,1\} & & (25)
\end{aligned}
$$

Equations(21) and (22) ensure that z_{ij} must be zero if either one of y_i or

y_j is zero. Equation (23) ensures that z_{ij} is one if both y_i and y_j are one. The convex hull of (21), (22),(23), (24) (25) is called the Boolean Quadratic Polytope, denoted

$$QP^n = conv\{(y,z) \in R^{n(n+1)/2} | (y,z) \text{ satisfy } (21),(22),(23),(24),(25)\}$$

Padberg [26] discussed this boolean quadratic polytope structure and gave some facet defining constraints. Other variations on this kind of polytope has also been discussed in the literature, for more details see Ji [14].

Since c_{ij} is always nonnegative, we don't need constraints (21)-(22). 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 (20). So we can reformulate the problem as the following (PRICEIP),

$$\begin{aligned}
\min \quad & -\sum_{i\in V} \pi_i y_i + \sum_{(i,j)\in E} c_{ij} z_{ij} & & \\
\text{s.t.} \quad & z_{ij} \geq y_i + y_j - 1 & & (26)\\
& \sum_{i=1}^{n} a_i y_i \geq S & & (27)\\
& \sum_{i\notin Q_k} y_i \geq 1 & k = 1,\ldots,q \quad & (28)\\
& z_{ij} \geq 0 & (i,j) \in E \quad & (29)\\
& y_v \in \{0,1\} & v \in V \quad & (30)
\end{aligned}$$

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 [29] . They are similar to the constraints generated for the linear relaxation of Quadratic Knapsack Problem in Caprara, Pisinger and Toth[6].

For $i = 1..n$, we multiply the size constraint (27) by y_i and replace y_i^2 by y_i, $y_i y_j$ by z_{ij}, getting the following valid constraint:

$$\sum_{j\neq i} a_j z_{ij} \geq (S - a_i) y_i \qquad i = 1,\ldots,n \tag{31}$$

This is also called the star inequality in Hunting, Faigle and Kern [13].

We can also multiply (27) by $1 - y_i$, and get

$$Sy_i + \sum_{j \neq i} a_j y_j - \sum_{j \neq i} a_j z_{ij} \geq S \qquad i = 1, \ldots, n \tag{32}$$

However, our computational results show that these additional constraints, as well as the general constraints for QKP mentioned in Padberg [26], 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. Since it is not necessary to generate the column with the most negative reduced cost each time, we stop solving the IP as soon as a column with a negative reduced cost is found. But if such a column does not exist, we have to solve the pricing problem to optimality to make sure.

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 calling the IP solver. Algorithms 2-4 give the three heuristics that we employed. They are applied in the order of 2 to 4. Only when the previous algorithms do not generate any columns to be priced in, would we try the next algorithm. These heuristic algorithms are similar to those in Ji and Mitchell [16] for k-way equipartition problems with minor changes of the size of the cliques we pick. For completeness, we list them below.

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

for $i = 1$ to n **do**
  Find the closest $2S - 1$ 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 3 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 no more than 10 columns on the same vertex added.

Algorithm 4 Heuristic Pricing Algorithm III: Greedily find a small node-edge-weighted clique of size at least S

Input:

S - minimum cluster size
c_{ij} - edge weight
π_i - node weight
a_i - node capacity

Steps:

for $i = 1$ to N **do**
  $CLIQ = \{i\}$
  Find vertex v s.t. $w(CLIQ, v) = \pi_v - \sum_{e \in \delta(v, CLIQ)} c_e$ is minimum for $v \notin CLIQ$.
  if $|CLIQ| < S$ or $w(CLIQ, v) > 0$ **then**
    $CLIQ = CLIQ + v$
    GOTO 7
  else
    GOTO 14
  end if
  Check if $CLIQ$ can be put into the violating column pool.
  Do local search near $CLIQ$, see if any columns can be put into violating column pool.
end for
Add the 10 most violating columns from the column pool, with no more than 5 columns on the same vertex added.

5 Branching

It is well known that the simple 0-1 branching rule in a branch-and-cut framework does not work in a branch-and-price algorithm. Therefore we have adopted the commonly used Ryan-Foster branching rule [27] for set partition models. In a fractional solution for MLP, we can identify two fractional variable x_{P_1} and x_{P_2}, and two vertices i and j, such that both vertices are in P_1 but only one of them is in P_2. 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.

This branching strategy actually corresponds to the branching strategy on x_{ij} in the compact branch-and-cut formulation of the problem in Ji and Mitchell [15]. This conforms with the observation in Lübbecke and Desrosiers [18], "to branch on meaningful variable sets". Our most valuable source of informa-

tion 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, correspond to the pigeon constraint for the compact formulation in Ji and Mitchell [15].

After branching, our CPPMIN problem changed into a GCPPMIN problem, for each combined vertex v now has a vertex weight a_v bigger than one, representing the number of original vertices it corresponds to. Consequently, a_v in the pricing subproblem for some vertices are not one any more. So we are in fact solving a GCPPMIN 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:

$$\sum_{i:P_i \subseteq Q} x_i \leq k - 1 \tag{33}$$

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

6 Stabilization

Another complication arises from the degeneracy of the primal problem. Notice the primal problem (7) 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 [9]. We are bound to have alternative dual solutions. We need to pick one out to construct the pricing subproblem. One would like to pick one to reduce useless oscillations in the dual space. Various methods have been proposed on this issue, see Lübbecke and Desrosiers [18].

Instead of using those more complicated methods in [18], we simply try to avoid oscillations by expanding our initial problem to include not only the columns of a good heuristic solution but also the clusters consisting of the closest $S - 1$ and S vertices to each vertex. It also helps to avoid generating useless columns by checking heuristic algorithms first to search for violated columns also.

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 Ji and Mitchell [15] 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 [25] with minor changes. It is illustrated in Algorithm 5. One major difference from the MINTO framework 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.

Algorithm 5 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.

7.1 Type I Problems

The data used here are the same types of data used in Ji and Mitchell [15]. Type I data is constructed from uniformly distributed points on a square. The Euclidean distances between each pair of vertices are taken as the edge weights.

Table 1 and 3 list the computational results for Type I data with $S = 4$ and $S = 7$ respectively. As one can see from the two tables, although the problem

is harder in the case of $S = 7$, and it requires longer time to solve, the trend and analysis of $S = 7$ is similar to the case of $S = 4$. Therefore in the following discussion, we concentrate on the case of $S = 4$.

In Table 1, Graph size n ranges from 21 to 103. $k = \lfloor n/S \rfloor$ represents the maximum number of clusters possible. The 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 labeled 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 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, with CPU speed at 440 MHz, system clock 110 MHz and Memory 128 MB. The heuristic algorithm is coded in FORTRAN. The branch and price algorithm is implemented using MINTO 3.0.2 in C and C++. The LP solver is CPLEX 6.1.

Table 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 rather quickly. For problems smaller than 43 vertices, they are solved to optimality 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 considerable amount of time.

To compare the branch-and-cut method in Ji and Mitchell [15] with the branch-and-price-and-cut method presented in this paper, we put together Table 2 with the data from Table 2 in [15] and Table 1. For comparison convenience, we have adjusted the reported time for "B&C run" to represent the total computation time, i.e. it includes root node time. The "B&C run" time in Table 2 in [15] does not include the root node time.

By comparing the performance in terms of both gap and time, one can see that branch-and-price-and-cut outperforms branch-and-cut, on instances with no more than 43 vertices. As the size of the problems get bigger, branch-and-cut's ability to stop at any time with a guaranteed bound duality gap became more useful, especially when it takes a long time for the branch-and-price to finish the root node to provide such a gap bound, even though it almost always returns a gap much smaller than the one provided by branch-and-cut.

Table 3 and 4 shows similar results for the case of $S = 7$. Compared to the case of $S = 4$, these instances are more difficult and takes much longer time to solve. We only show the result of the root node here. A comparison with the branch-and-cut gives us table 4, confirming our previous observation regarding the advantages and disadvantages of branch-and-price vs branch-and-cut on this problem.

7.2 Type II Problems

In Type II problems, the edge weights are generated directly as uniformly distributed random numbers between 1 and 100. Table 5 shows the results for Type II problems. As observed in Ji and Mitchell [15], these are harder partitioning problems. It's not surprising to see that these problems take a much longer time to solve than Type I problems in table 1. We also put together a comparison Table to evaluate against the branch-and-cut results from [15]. From this table, we can see Branch-and-price-and-cut performs much better than the branch-and-cut method. We think this is because the edge weights here do not satisfy triangle inequalities, resulting in a large number of triangle constraints being added in the branch-and-cut method. The branch-and-price algorithm does not use triangle constraints to enforce feasibility of the solutions, so it does not run into such problems. However this type of problem is still harder than Type I problems, because the heuristics for generating columns do not work as well as in Type I problem. A lot more columns are

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
Columns 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 1
Branch-and-Price Results on CPPMIN Type I Problems for $S = 4$

generated using IP solver, which is much more time-consuming. Despite this, we can still conclude that branch-and-price method fits better than branch-and-cut method for problems that violate a lot of triangle constraints.

We also conducted experiments using data for Micro-aggregation problems. Again the root node gives a very tight LP approximation for the problem, but

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	10.24	47.15	215.48	514.28	685.76
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 2
Comparison of Branch-and-Cut and Branch-and-Price Results on CPPMIN Type I Problems for $S = 4$

it takes a long time to compute. The basic trends and analysis are exactly the same as those for Type I data. For detailed results, see [14].

8 Conclusions

In this paper, we discussed solving CPPMIN using a 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 slows down for larger 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 the branch-and-cut method on the same problem in Ji and Mitchell [15], we see a better performance from branch-and-cut-and-price on all types of data, especially for Type II, where a large number of edge weights do not satisfy triangle constraints. However, on instances of large sizes, if an error bound is needed, branch-and-cut would be preferred. If the user is only interested in looking for a good solution without an error

n	36-41	71-76
$k = \lfloor \frac{n}{S} \rfloor$	5	10
Heuristic Alg.		
Gap	1.90%	6.82%
Time	0.003	0.007
Root Node		
Total Instances	26	20
Solved exactly	25	16
Better Solution	16	19
Gap	0.03%	0.28%
Time	43.05	842.92
LPs solved	21	36
Total Cuts	0	3.7
Total Columns	375	726
Initial Columns	208	430
Columns by IP	3	3
Total IP Time	38.88	828.70
Avg IP Time	11.54	247.37

Table 3
Branch-and-Price Results on CPPMIN Type I Problems for $S = 7$

bound guarantee, branch-and-price might give a good solution more quickly.

Future work includes more sophisticated heuristics for generating columns. This is particularly important for larger values of S in order to control the time spent in the exact IP column generation subproblem. Preprocessing the PRICEIP to exploit our knowledge of the structure of the problem may also be helpful.

We are confident that this method can be extend to GCPPMIN, i.e., partitioning problems 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.

n	36-41	71-76
$k = \lfloor \frac{n}{S} \rfloor$	5	10
B&C root		
Gap	2.10%	2.32%
Time	67.93	235.65
B&C run		
Gap	0.58%	2.04%
Time	150.60	402.68
B&P root		
Gap	0.03%	0.28%
Time	43.05	842.92

Table 4
Comparison of Branch-and-Cut and Branch-and-Price Results on CPPMIN Type I Problems for $S = 7$

References

[1] Roberto Baldacci, Lawrence Bodin, and Aristide Mingozzi. The multiple disposal facilities and multiple inventory locations rollon-rolloff vehicle routing problem. *Computers and Operation Research*, 33:2667–2702, 2006.

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

[3] 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.

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

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

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

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

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

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

[10] M. Gröschel and Y. Wakabayashi. Facets of the clique partitioning poly-

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 5
Branch-and-Price Results on CPPMIN Type II Problems for $S = 4$

tope. *Mathematical Programming*, 47:367–387, 1990.

[11] 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*

n	21-23	41-43	61-63
$k = \lfloor \frac{n}{S} \rfloor$	5	10	15
B&C root			
Gap	4.74%	13.28%	-
Time	75.69	1657.34	-
B&C run			
Gap	0%	8.93%	-
Time	116.52	2134.04	-
B&P root			
Gap	3.23%	11.84%	20.85%
Time	27.94	322.75	2936.08
B&P run			
Gap	0%	2.09%	13.07%
Time	31.85	617.74	5504.16

Table 6
Comparison of Branch-and-Cut and Branch-and-Price Results on CPPMIN Type II Problems for $S = 4$

Science, volume 1084, pages 175–189. Springer, 1996.

[12] 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.

[13] 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.

[14] Xiaoyun Ji. *Graph partition problems with minimum size constraints.* PhD thesis, Rensselaer Polytechnic Institute, 2004.

[15] Xiaoyun Ji and John E. Mitchell. The clique partition problem with minimum clique size requirement. Technical report, Rensselaer Polytechnic Institute, 2005.

[16] Xiaoyun Ji and John E. Mitchell. Finding optimal realignments in sports leagues using a branch-and-cut-and-price approach. *International Journal of Operational Research*, 1(1-2):101–122, 2005.

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

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

[19] Anuj Mehrotra, Ellis L. Johnson, and George L. Nemhauser. An optimization based heuristic for political districting. *Management Science*,

44(8):1100–1114, August 1998.

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

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

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

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

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

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

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

[27] 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.

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

[29] 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.

[30] 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.

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

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