EXACT SOLUTION METHODS FOR THE ASSIGNMENT PROBLEM WITH CONFLICT CONSTRAINTS

by

Elif Arslan

B.S., Industrial Engineering, Boğaziçi University, 2019

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in M.S., Industrial Engineering
Boğaziçi University

May, 2022

ACKNOWLEDGEMENTS

The dissertation process was quite a journey with lots of learning, researching and studying for endless nights and weekends. However, I enjoyed every minute. I am not sure if it would be the case if my supervisor was not İ. Kuban Altınel. I am deeply grateful for everything for what he taught, for his patience and for his belief in me. I would also like to thank dear Temel Öncan and dear Z. Caner Taşkın for taking time to read my work and to participate in the thesis jury.

I would like to thank my family. Being far away in different cities and even in different countries did not prevent them to fully support me. They have been right there with me all the time.

I would like to thank Alim Buğra Çınar, for helping out without any hesitation and sharing his experience when I needed the most.

I would also like to thank my friends Hazal Dalak and Özlem Özdemir who are going through dissertation process as well. Being together and supporting each other made everything much easier.

ABSTRACT

EXACT SOLUTION METHODS FOR THE ASSIGNMENT PROBLEM WITH CONFLICT CONSTRAINTS

The Assignment Problem with Conflict Constraints (APC) deals with finding a maximum weight assignment in the presence of incompatibilities. The incompatibilities are constituted by the conflicting edge pairs such that both edges in a conflicting pair cannot be in a feasible solution. Unlike the assignment problem, APC is a $\mathcal{NP}$-hard problem; therefore, it brings about the importance of finding efficient solution procedures for APC. Yet, there are only a few studies on APC that put forward exact solution procedures.

To the best of our knowledge, this is the first study which proposes Branch & Cut algorithms for the solution of APC. Within the computational analysis, we first evaluated the performance of Branch & Bound with different branching rules. Afterwards, we assessed the performance of additional algorithms with the best performing branching rule. Finally, we checked the contribution of the added cuts to the overall problem with the selected branching rule and the default branching rule of the mixed integer linear programming commercial solver. The computational results showed that the Branch & Bound algorithm with the best performing branching rule, the Branch & Cut algorithm with clique cuts and the Branch & Cut algorithm with combination of clique and cycle cuts performed better compared to the state-of-art commercial solver.

ÖZET

ÇATIŞMA KISITLI EN BÜYÜK AĞIRLIKLI ATAMA PROBLEMİ İÇİN KESİN ÇÖZÜM YÖNTEMLERİ

Çatışmalı Atama Problemi (ÇAP), uyumsuzlukların varlığında maksimum ağırlık atamayı bulmakla ilgilenir. Uyumsuzluklar, çatışan kenar çiftleri nedeniyle ortaya çıkar ve çatışan çiftteki her iki kenar olası bir çözümde birlikte bulunamaz. Atama probleminden farklı olarak, ÇAP bir $\mathcal{NP}$-zor problemdir ve ÇAP hakkında kesin çözüm prosedürlerini ortaya koyan sadece birkaç çalışma vardır; bu nedenle, ÇAP'yi çözmek için verimli bir kesin çözüm prosedürü bulmak bu çalışmanın motivasyonu olmuştur.

Bildiğimiz kadarıyla, ÇAP'yi ele alan çalışmalarda Dal-Kesim çözüm yaklaşımı kullanılmamaktadır. Bu tezde, geliştirilen algoritmaların başarımlarını artırmak için tasarlanmış ek algoritmalarla birlikte çeşitli dallanma kuralları ve kesiler önerilmektedir. Önerilen algoritmaları ölçmek için öncelikle dallanma kurallarının başarımları değerlendirildi. Daha sonra seçilen dallanma kuralı ile ek algoritmaların etkinlikleri ölçüldü. Son olarak, seçilen dallanma kuralını ve bir tam sayı programlama ticari çözücüsünün varsayılan dallanma kuralını kullanarak, eklenen kesintilerin genel problemlere katkısı belirlendi. Bilgisayısal sonuçları, seçilen dallanma kuralına sahip Dal-Sınır algoritması ile düğümsel ve maksimal klik kesimli Dal-Kesme algoritmasının çok başarılı olduğunu göstermektedir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

C	Conflict graph
$d_G(v)$	Degree of node v
EC	Even cycle
$E(G)$	Edge set of graph G
G	Simple graph
$G \backslash S$	Subgraph of G by removing node set S
$G[X]$	Induced subgraph of G by removing edge set X
K	Clique
$\mathcal{K}(G)$	Set of all cliques in graph G
$N_C(e)$	Conflicting edge set of graph edge e
OC	Odd cycle
$P(G)$	Confliting edge pair set of graph G
$V(G)$	Node set of graph G
w_e	Weight of edge e
α_G	Stability number of graph G
$\delta_G(v)$	Neighbors of node v in graph G
ρ	Initial solution Size
$\sigma_G(e)$	Neighbors of edge e in graph G
ϕ	Graph density
ω	Conflict density

LIST OF ACRONYMS/ABBREVIATIONS

AP	Assignment Problem
APC	Maximum Weight Assignment Problem with Conflict Constraints
B&B	Branch & Bound
B&C	Branch & Cut
IP	Integer Programming
LB	Lower Bound
LP	Linear Programing
MaxTSPC	Maximum Weight Traveling Salesman Problem with Conflict Constraints
MCAPC	Minimum Cost Assignment Problem with Conflict Constraints
MCMC	Minimum Cost Matching with Conflict Constraints
MWMC	Maximum Weight Matching with Conflict Constraints
MWSSP	Maximum Weight Stable Set Problem
TPC	Transportation Problem with Conflicts
UB	Upper Bound

1. INTRODUCTION

The ordinary Assignment Problem (AP) deals with assigning a certain set of items to another set of items. The problem has received a great deal of interest in studies [1] and it has been addressed in a vast number of real life applications for different areas of practice. AP arises in problems such as the transportation problem, the workforce planning and the task assignment. By definition, AP is a special type of matching problem whose graph consists of two sets of nodes such that within a set no two nodes have an edge between them. Such described graph is called a bipartite graph. Even though the assignment problem is an Integer Programming (IP) problem, as one asks whether an item is assigned to another, due to its bipartite graph structure the extreme points of the polytope that defines the feasible region of the linear programming relaxation is integral [2]. Having such property, the relaxed AP yields the same optimal solution and the problem becomes solvable in polynomial time.

The AP formulation may require additional constraints as the variations of the problem arise, which is not unexpected in complex real life problems. Conflict constraints is a set of such additional constraints and the problem obtained through the inclusion of conflict constraints to the AP and setting the objective as maximization of the total value of the selected edges is called Maximum Weight Assignment Problem with Conflict Constraints (APC). We say that there exists a conflict (or disjunction) if the assignment of *job* i to *machine* j bans the assignment of *job* k to *machine* l and vice versa. Unlike AP, APC is not an easy problem to solve [3]. Hence, finding an efficient solution the APC is in curiosity and constitutes the main motivation of this thesis.

The efforts for efficiently solving the APC becomes worthwhile after its real life applications are taken into consideration. The conflict constraints represent the incompatibilities between a pair of edges, i.e., a pair of assignment decisions.

Therefore, storage of a toxic product in the presence of consumed goods or transportation of products with different levels of room temperature requirements can be modeled as APC. It is possible to see the applications in workforce planning as well. The problem of designation of soldiers or doctors to fulfill obligatory services without setting spouses on duty apart can be modeled as APC.

One should also note that there are a variety of problems such that the simplifications of which is APC. For example in [4] a routing problem with conflict and workload constraints is studied. In this case, APC is obtained after subtour elimination and workload constraints are relaxed, signs of the values in the cost vector are reversed and equality constraints of AP are updated to be inequalities.

Similarly, relaxation of the subtour elimination constraints of the Traveling Salesman Problem with Conflict (MaxTSPC) results in APC. Hence, any efficient solution method developed for APC will also contribute to the solution of MaxTSPC.

2. NOTATIONS AND DEFINITIONS

This chapter aims to introduce the definitions and notations used in the next sections. APC is represented with a simple bipartite graph and the combinatorial structure of the problem is utilized in the proposed solution algorithms. Hence, various notations and terminologies are crucial in conveying the details of the thesis.

We can denote a *bipartite graph* with $G = (V(G), E(G))$ whose edge set is represented with $E(G)$ and node set is denoted with $V(G) = X(G) \cup Y(G)$, where $X(G) \cap Y(G) = \emptyset$. $E(G)$ contains $e = \{i, j\}$ if there is an edge between $v_i \in X(G)$ and $v_j \in Y(G)$. Each edge is assigned to a positive weight which is denoted by w_e. If edge e has a conflict with $f = \{k, l\}$, the edge pair $\{e, f\}$ is included in the conflicting edge pair set $P(G)$ of graph G and in the conflicting edge pair set of edge e and edge f denoted respectively by $N_C(e)$ and $N_C(f)$. In addition, neighbors of a node v is the set of edges whose one end is v, which is represented with $\delta_G(v)$. We call $|\delta_G(v)|$ as the degree of node v and denote it as $d_G(v)$. On the other hand, neighbors of an edge $e = \{i, j\}$ is described as the set of edges whose one end is either $v_i \in X(G)$ or $v_j \in Y(G)$ and it is denoted by $\sigma_G(e)$.The number of neighbors of an edge is the degree of the edge and represented by $|\sigma_G(e)|$.

A *subgraph* $G \backslash S$ of a graph G is the graph obtained after removing a set of nodes denoted by S and their neighbors from G whereas an *induced subgraph* $G[X]$ is the graph obtained after removing the set of edges, X, from graph G. Therefore, if the edge set, X is removed from the subgraph $G \backslash S$, the resulting graph is denoted by $G \backslash S[X]$.

A *conflict graph* $C = (V(C), E(C))$ is used to represent the conflicts between edges in G. The conflict graph is constructed by defining a node for each edge e in $E(G)$ and assigning a weight to the node in the conflict graph, which is equal to the weight of edge $e \in E(G)$. Hence a conflict graph C constructed from G satisfies $|E(G)| = |V(C)|$.

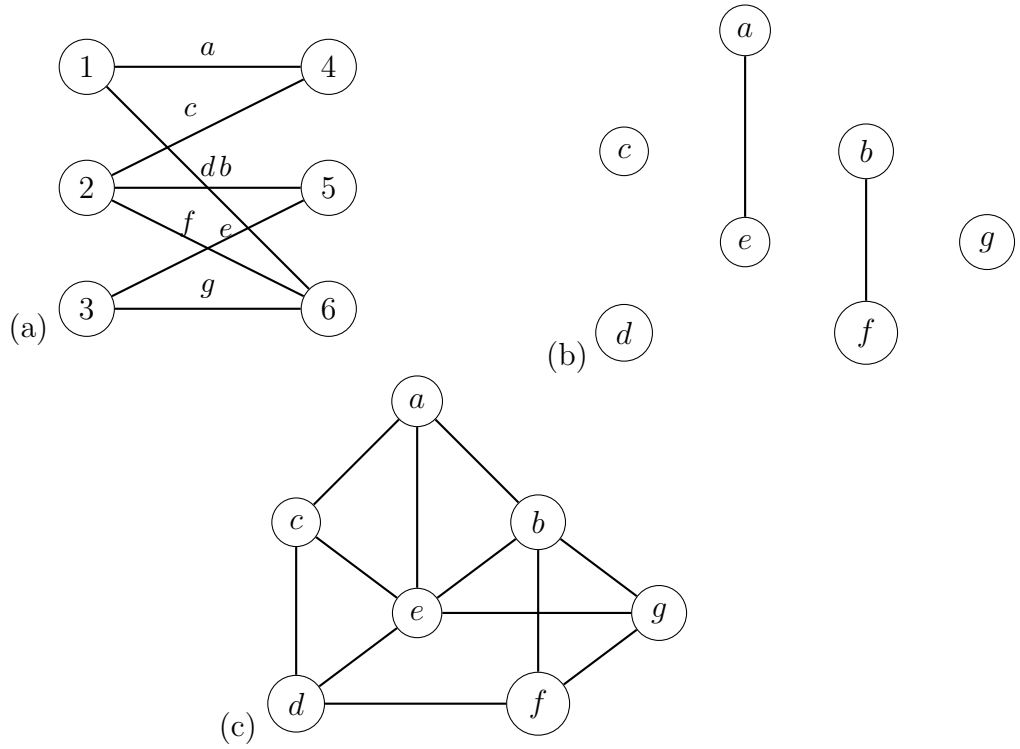


Figure 2.1. (a) Original graph G s.t. $\{a, e\}, \{b, f\} \in P(G)$ (b) Conflict graph (c) Extended conflict graph.

For the sake of simplicity, we denote the node in conflict graph that is created for the edge $e \in E(G)$ with the same naming, e. In addition, there is an edge $\{e, f\}$ in C if and only if $e \in E(G)$, $f \in E(G)$ and these edges are in conflict in G, i.e., $\{e, f\} \in P(G)$. One can also extend the Conflict Graph by taking into account the structure of the test instances. For example, given an assignment problem if an edge $e \in E(G)$ is selected to be in the assignment, remaining edges within $\sigma_G(e)$ cannot be selected. Therefore, each pair $\{e, f\}$ such that $f \in \sigma_G(e)$ can be considered to have a conflict as well and one can define an edge between the corresponding nodes $\{e, f\} \in V(C)$ in C. An example is provided in Figure 2.1 in which conflict graph and extended conflict graph are constructed from a bipartite graph.

A *clique* is a set of nodes that are pairwise adjacent and $\mathcal{K}(G)$ denotes the set of all possible cliques within graph G. The size of a clique K is the number of nodes within K and it is represented with $|K|$. A *maximal clique* is a clique such that no remaining node in the graph can be added without disrupting the clique property.

The set of nodes that are pairwise non adjacent is a *stable set* of G. The maximum size of a stable set in a graph G is called the *stability number* of G and it is denoted by α_G. Finding a stable set with maximum weight is called *maximum weight stable set problem* (MWSSP). The relaxation of the integrality of the decision variables yields *relaxed stable set problem* ($MWSSP_{Relaxed}$). The convex hull of the feasible solutions to $MWSSP_{Relaxed}$ is called the stable set polytope [5].

In a graph, a *cycle* $C = (V(C), E(G))$ is a path which starts and ends at the same node and $|C| = |E(C)| \geq 2$. A cycle is called *odd cycle* and it is denoted by OC if $|C|$ is odd, otherwise it is called *even cycle* and it is denoted by EC. A *chord* is the edge between two nonadjacent nodes in C and a *hole* is a cycle which does not have any chord. Given a cycle C if there is a node $v \in G$ such that for each member $u \in C$, $\{u, v\} \in E(G)$, $C \cup \{v\}$ is called a *wheel* where v is the *center* of the wheel.

3. LITERATURE REVIEW

The conflict constraints are used to describe the incompatibilities between edges. So far in literature, these constraints are studied in various problems such as the minimum spanning tree problem ([6], [7], [8], [9]), the bin packing problem ([10]), the transportation problem ([11]) and the routing problem ([12], [13]). In addition they are used to formulate more specific problems. For example, [4] deals with the problem of assigning nurses to the routes that start and end at the hospital to give health care to the patients in their homes in the presence of conflict and workload constraints. In this problem the incompatibility between the nurse-patient pairs arise as patients cannot receive incompatible services from different nurses within the same day.

Furthermore, conflict constraints are addressed in matching problems as well. While studies such as [14], [15] addressed generalized matching problems with maximization objective and conflict constraints (MWMC), there are also studies on the assignment problem, which is a matching problem with conflict constraints on a bipartite graph. The Maximum Weight Assignment Problem with Conflict Constraints (APC) was firstly introduced by Darmann textitet al. (2011) [3]. They identified the generalized version of APC but with a minimization objective (MCMC) as edges are coupled with costs rather than weights. In addition, in the study it is proved that the problem is $\mathcal{NP}$-hard by showing that the simplest version of the problem where the conflict graph is a 2-ladder graph can be polynomially reduced to an $\mathcal{NP}$-hard problem.

Besides [3], within our knowledge there are only a few studies such as [14], [16], [17] and [18] none of which is identified the problem as a maximization problem. The study [14] puts forward more complexity analysis on MCMC via focusing on the original graph rather than the conflict graph. They have proved that MCMC is $\mathcal{NP}$-hard even if the graph G is a set of disjoint 4-cycles.

Furthermore, they provided polynomial time solvable cases of MCMC and proposed various heuristics by focusing on the case that the original graph is bipartite.

On the other hand, in the study [16], an exact solution procedure for the Minimum Cost Assignment Problem with Conflict Constraints (MCAPC) is proposed together with a local search heuristic algorithm. However, the exact solution algorithm is not in a Branch and Bound (B&B) or Branch and Cut (B&C) scheme; rather it has a structure similar to the cutting plane algorithm. [17], [18] are the only studies that proposed B&B solution approaches. The study [17] proposed two branching strategies, both of which constructs the branching tree as a binary tree. The first method utilized the linear programming relaxation of MCAPC by assigning a fractional variable to its upper bound and to its lower bound in the first and second child branching node, respectively. However, in the latter method, for each edge in the pair, a child branching node is created and its decision variable is set to its lower bound. It was found in the study that the former branching rule outperformed the latter.

Compared to [17], the study [18] proposed B&B algorithm with more branching rules including a clique branching by constructing a preferably maximum size clique K in conflict graph that contains conflicting edge pairs and creating either two branching nodes to separate the conflicting edges or creating $(|V(K)| + 1) - many$ branching nodes so that the decision variable of each node in the clique is set to its upper bound in a branching node and the decision variables of all nodes in the clique are set to their lower bound for the extra branching node.

So far, there is no study that explores the effect of the cuts to the problem's bound and the CPU time required to reach the optimum solution. With this respect, finding the most effective branching rule that can be accompanied with cut separations and their addition to the branching nodes may be a prominent step to discover a solution approach that is significantly more efficient than the current commercial solvers. The aim of this thesis is to propose such B&C strategies and strengthen them with additional algorithms.

4. PROBLEM FORMULATION

Maximum Weight Assignment Problem with Conflict Constraints can be perceived as the composition of two problems: Ordinary Assignment and Maximum Weight Stable Set problems. When conflict constraints are relaxed, APC becomes the ordinary assignment problem, which can be solved to optimality in polynomial time [2]. When the objective is to maximize the total weight of the selected edges, the Maximum Weight Assignment Problem (MWAP) can be formulated as follows [1]:

$$MWAP: \quad \max \sum_{e \in E(G)} w_e x_e \tag{4.1}$$

$$\text{s.t} \sum_{e \in \delta_G(v)} x_e = 1 \qquad v \in X(G) \tag{4.2}$$

$$\sum_{e \in \delta_G(v)} x_e = 1 \qquad v \in Y(G) \tag{4.3}$$

$$x_e \in \{0, 1\} \qquad e \in E(G). \tag{4.4}$$

In this formulation x_e is a binary decision variable that takes value 1 if and only if edge e is decided to be in the assignment. The objective function in (4.1) is the maximization of total weight of the selected edges. The constraint sets (4.2) and (4.3) force each node in $X(G)$ and $Y(G)$ respectively to have exactly one adjacent edge in a feasible solution. Note that $|X(G)| = |Y(G)|$.

Similarly, in APC when we relax the assignment constraints, remaining constraints are the disjunctive constraints. In that case we can make use of the conflict graph C to find a conflict-free set of edges in the original graph G. As a set of nodes in the conflict graph will not have any conflict if there is no edge between them, one can look for a stable set in the C. However, there are several formulation methods for the maximum weight stable set problem.

In the first approach a *strong formulation* is constructed to represent the disjunctive constraints using edge pair inequalities [5]:

$$MWSSP_S: \quad \max \sum_{e \in E(C)} w_e x_e \tag{4.5}$$

$$\text{s.t} \quad x_e + x_f \leq 1 \qquad \{e, f\} \in E(C) \tag{4.6}$$

$$x_e \in \{0, 1\} \qquad e \in V(C). \tag{4.7}$$

In $MWSSP_S$ unlike MWAP, the binary decision variable x_e takes value 1 iff node e is included in the stable set. In addition, the constraint set (4.6) ensures that at most one edge from a conflicting pair can be selected.

To reduce the size of the constraint set to $|E(G)|$, it is possible to aggregate the conflict constraints by making use of the property that each edge $e = \{i, j\} \in E(G)$ has $|N_C(e)| - many$ conflicts in graph G or equivalently each node $e \in V(C)$ has $|d_C(e)| - many$ neighbors in graph C. The aggregation yields the following new set of constraints which can be called as *weak conflict inequalities* [19]:

$$d_C(e)x_e + \sum_{f \in N_C(e)} x_f \leq d_C(e) \qquad e \in V(C). \tag{4.8}$$

The constraint set (4.8) can replace the constraint set (4.6) [19] thus, (4.5), (4.7), and (4.8) constitute the *weak formulation* of the problem and it is denoted by $MWSSP_W$.

Maximum Weight Stable Set Problem can be formulated by using maximal cliques as well. As a clique being a set of pairwise adjacent nodes, in a solution only one node can be selected for each clique K to have a non-conflicting set of edges.

Based on this property, the Maximum Weight Stable Set Problem can be formulated as follows [5]:

$$MWSSP_K: \quad \max \sum_{e \in V(C)} w_e x_e \tag{4.9}$$

$$\text{s.t} \quad \sum_{e \in K} x_e \leq 1 \qquad K \in \mathcal{K}(C) \tag{4.10}$$

$$x_e \in \{0, 1\} \qquad e \in V(C). \tag{4.11}$$

Similar to $MWSSP_S$ and $MWSSP_W$, in $MWSSP_K$ the decision variable x_e takes value 1 iff node $e \in V(C)$ is selected. Furthermore, the constraint set (4.10) ensures that at most one node can be selected from each clique $K \in \mathcal{K}(C)$.

Based on the formulations described so far, we can write the *strong formulation* of APC in the following way:

$$APC_S: \quad \max \sum_{e \in E(G)} w_e x_e \tag{4.12}$$

$$\text{s.t} \quad \sum_{e \in \delta_G(v)} x_e = 1 \qquad v \in X(G) \tag{4.13}$$

$$\sum_{e \in \delta_G(v)} x_e = 1 \qquad v \in Y(G) \tag{4.14}$$

$$x_e + x_f \leq 1 \qquad \{e, f\} \in E(C) \tag{4.15}$$

$$x_e \in \{0, 1\} \qquad e \in E(G). \tag{4.16}$$

The objective (4.12) ensures that the total weight of the selected edges is maximum. Constraints $(4.13) - (4.14)$ are the assignment constraints. Similar to (4.6), constraint (4.15) ensures that at most one edge from a conflicting edge pair can be selected.

In an ordinary assignment problem if edge e is selected, any edge $f \in \sigma_G(e)$ cannot be included in the assignment. Hence, no two adjacent nodes in the extended conflict graph can be selected together in a feasible solution.

Since both (4.13) and (4.14) also imply cardinality restrictions, we can interpret (4.12) − (4.14) together with (4.16) as a maximum weight stable set problem with cardinality restrictions. Such problem is polynomial time solvable because the conflict graph of the implied bipartite graph G is also the line graph of G [20]. However, when constraint set (4.15) is included into the problem, the conflict graph loses its special structure and APC becomes $\mathcal{NP}$-hard [20], [21]. It is also possible to rewrite conflict constraints as proposed in [19]:

$$d_C(e)x_e \; + \sum_{f \in N_C(e)} x_f \leq d_C(e) \qquad e \in \; E(G). \tag{4.17}$$

We can replace (4.15) with (4.17), which yields the *weak formulation* of APC (APC_W). Additionally, we can formulate conflict constraints with the following clique constraints which yields the clique formulation of APC namely APC_K:

$$\sum_{e \in K} x_e \leq \; 1 \qquad K \; \in \; \mathcal{K}(C). \tag{4.18}$$

5. BRANCH AND CUT PROCEDURE

We propose various Branch and Cut (B&C) solution approaches to solve APC by considering various branching strategies and cut separations. Besides, additional algorithms such as *Feasibility Test* are also considered within the solution approach in order to decrease the number of branching nodes created during the B&B procedure, to detect heuristic solutions and to tighten the upper bound.

Among the possible formulations of APC, strong formulation (APC_S) is selected for this study and complete B&C solution procedure is defined by taking the properties of the formulation into account. In addition, branching rules are defined using the problem's original graph whereas cuts are separated from the extended conflict graph. The conflict graph without the edge extensions is not used within the solution. Hence, for the sake of simplicity, from this point on an extended conflict graph will be called conflict graph.

The complete B&C procedure is given in Figure 5.1. Initially the global upper bound (UB) is set to $+\infty$ and the global lower bound (LB) is selected to be $-\infty$. For each feasible solution found during B&C iterations either the LB is updated or the branching node is pruned depending on whether the objective value of the subproblem is better than the current LB. The B&C procedure aims to find as many feasible solutions as possible to increase the LB and at the same time to decrease the UB since having a LB that is equal to the UB results in the optimality.

In the B&C procedure, the root branching node of APC IP problem is the relaxation of the complete problem. Contingent on the branching procedure, the relaxation method is either determined as linear programming relaxation or conflict constraint relaxation each of which is polynomially solvable. The relaxation of APC is the problem of the root node of the branching tree, i.e., in the beginning of the B&C procedure, it is the only branching node to *Bound*.

Within the procedure *bound*, the relaxed subproblem is solved and the objective value is compared with LB. In the case of having an objective value smaller than LB, the branching node is pruned and another branching node is selected to *bound*.

Algorithm 1 APC B&C Procedure

Input: Graph $G = (X(G) \cup Y(G), E(G))$ with $P(G) \neq \emptyset$, $SeparationTimeLimit$
Output: Maximum weight conflict-free assignment A
Initialization $LB = -\infty$, $S \leftarrow \{(0, APC^{(0)}_{Relaxed})\}$
while $S \neq \emptyset$ **do**
  $r \leftarrow PickBranchingNode(S)$
  $S \leftarrow S \backslash r$
  /* SOLVE RELAXED SUBPROBLEM */
  if sol($APC^{(r)}_{Relaxed}$) is infeasible **then**
    Prune r
  else if $IsFeasibleSolnToAPC$(sol($APC^{(r)}_{Relaxed}$)) **then**
    if obj($APC^{(r)}_{Relaxed}$) > LB **then**
      LB $\leftarrow$ obj($APC^{(r)}_{Relaxed}$)
    end if
    Prune r
  else
    /* ADD CUTS */
    $R = \{r_j = 1, 2, \ldots, N\} \leftarrow CreateBranchingNodes(APC^{(r)}_{Relaxed})$
    for $j = 1$ to N **do**
      /* ADD VALID BRANCHING NODES */
    end for
  end if
end while
Return LB, Maximum weight conflict-free assignment

Figure 5.1. APC B&C Procedure.

Otherwise, the cut generation algorithm is run on the subproblem before branching if the solution of the relaxed problem is not feasible. In the occasions where the solution of the subproblem is feasible (thus integral) to APC and where its objective value is higher than LB, both LB and the incumbent solution are updated.

Depending on the branching rule, *IsFeasibleSolnToAPC* function changes as for some branching rules conflict constraints are relaxed while for others linear programming relaxation is utilized. The details of the relaxation methods of each branching rule will be provided in Section 6.

It is important to note that, except for some cases such as pruning the root node directly, during B&C iterations, there will be more than one subproblem to choose from to bound. Which branching node to bound at first is determined via its parent branching node's objective value. The largest objective value of the relaxed subproblems equals to the UB hence, selecting the branching node with larger objective value enables relatively rapid decrease in the UB.

The importance of the cut procedure reveals itself similarly. After bounding a branching node, resolving the relaxed problem by cutting off fractional parts of the feasible region improves the upper bound thus reduces the difference between the fractional and integral solution hence yields a better UB.

With the completion of cut generation and subproblem resolution, the branching process starts. From a branching node at least two new branching nodes are created and this number varies with the selection of the branching rule. For each child branching node created, compared to its parent there is one more decision variable which is fixed to its upper or lower bound. However, the union of the set of integral solutions is in child branching nodes thus B&C iterations guarantee to preserve the optimum solution even though the parent is removed from the active branching nodes list, S.

Generally, in a B&C solution, the iteration is completed and a new branching node is selected when the branching procedure ends. Such execution flow means that there will be as many iterations as the number of created branching nodes and overall CPU time needed for the B&C solution is directly affected by the iteration number. Before *bound* procedure and even before accepting the branching node, it is possible to assess quickly whether the additional decision variable that is assigned to one of its bounds leads to a feasible or an infeasible solution. This questioning prevents evaluation of redundant branching nodes. As a result, it may lead to a more efficient B&C solution approach. However, the success of such algorithms may be subject to various factors such as the graph density and the used branching rule. Hence, one should design the algorithms by taking these factors into account.

6. BRANCHING RULES

In the B&C algorithms, branching strategy plays a key role in the algorithm's performance [22]. Thus, we will consider three different branching rules: edge branching, conflicting pair branching, and maximal matching branching. In these rules, the subproblem relaxation method and the number of child branching nodes that are created during branching the process differ and they will be explained in detail in the next subsections.

6.1. Edge Branching

In edge branching, the branching yields two subproblems. In one subproblem edge e is selected and thus its corresponding binary decision variable x_e is set to 1, and in the other subproblem the edge is not selected thus x_e takes the value 0. To reflect the branching decision on the conflict graph on which the valid inequalities are separated to, edge deletion is performed.

For the branching node which is created by assigning a decision variable to its lower bound, the edge it represents is deleted from the original graph. On the other hand, for the other branching node, edges that are adjacent or in conflict are removed from the original graph. In either way the conflict graph is updated based on the changes made on the original graph.

As edge branching rule is based on the property that the relaxation of APC in the branching nodes yields a fractional solution, it is natural to use linear programming relaxation to solve the subproblems in B&C iterations. In this strategy the branching variable whose value is closest to 0.5 is selected to be branched on to force other decision variables to also take values in their upper or lower bounds.

However, this strategy is likely to result in a unbalanced branching tree [22], since setting a variable to 0 will not have a direct effect on the upper bound of other decision variables, whereas setting a decision variable x_e to 1 will also set upper bounds of decision variables to 0 for each edge either in $N_C(e)$ or in $\sigma_G(e)$.

6.2. Conflicting Pair Branching

The property that APC becomes an assignment problem when the conflict constraints are relaxed can also be used to determine another branching rule. The LP relaxation of the subproblem can be obtained by using the Hungarian algorithm [23] with a small adjustment. The classical version of the Hungarian algorithm is designed to find the minimum weight assignment. Therefore, to have a maximum weight assignment, the edge weights should be multiplied with -1. Given an integral solution to APC found in a node in the branching tree, one can check if there is any conflict between the selected edges. If there is none, either the lower bound and incumbent solution is updated or the node is pruned depending on the difference between the objective value and the global lower bound. However, if there is at least one pair of conflicting edges, the solution cannot be feasible and branching becomes necessary before a new branching node is evaluated.

In this case, a conflicting edges pair $\{e, f\} \in P(G)$, which is contained in the solution, can be selected and similar to the edge branching, two subproblems are created using the pair in such a way that in one branching node the decision variable of edge e is set to 0 and the decision variable of edge f is set to 0 in the other branching node. Original and conflict graphs of the subproblems are updated with the method described in edge branching.

6.3. Maximal Matching Branching

Given a subproblem, the Hungarian algorithm finds an assignment so that all nodes in the bipartite graph are assigned to another without considering the conflicting edges pairs. If the cardinality restrictions on the nodes are relaxed but the conflicting edge pairs are taken into account, solving the relaxation of the subproblem becomes finding a conflict-free maximal matching where the size of the matching is not necessarily equal to $|V(G)|/2$.

As suggested in [15], with maximal matching branching, decision whether new branching nodes are to be created is given by checking the size of the maximal matching M constructed after the relaxation of the subproblem r is found to be feasible. In the case of having $|M| \neq |V(G)|/2$, it can be inferred that there is at least one edge $e \notin M$ in the optimum solution to the overall problem therefore by pivoting on a node i, $|\delta_{G^{(r)}}(i) \backslash M| - many$ branching nodes are created.

On the other hand, if $|M| = |V(G)|/2$ but the total weight of the matching is less than the lower bound, branching is again necessary as there may be undiscovered possible matchings of the same size but with higher total weight. When this is the case, for each node outside of the matching a new branching node is created and a total of $|V(G^{(r)}) \backslash M| - many$ branching nodes are obtained.

7. CUT GENERATION

In B&C procedure, cuts which are created using violated valid inequalities may lead to a tighter feasible region of the linear programming relaxation and thus considerable decrease in the value of the upper bound. For that purpose we address various valid inequalities. Using a conflict graph, the inequalities are separated after solving the relaxed problem of each branching node in B&C procedure. Solving APC on graph G is the same as solving MWSSP with cardinality restrictions on conflict graph C. Therefore, inequalities that are valid for MWSSP are selected and considered for cut generation within the proposed B&C solution approach.

The steps of the cut separation procedure are outlined in Figure 7.1. Before branching, valid inequalities are separated using the conflict graph of the subproblem and the solution of the relaxed subproblem since constructing the valid inequalities that are more likely to be violated requires both. The details of the separation procedures together with how the subgraph and the solution of the relaxed subproblem are used will be provided in the subsections. Each separation algorithm will be used in place of *SeparateCut* function in Figure 7.1.

During cut generation, for each valid inequality that is found to be violated, a cut is created. However, to control the CPU time spent for the cut creation procedure, the number of cuts to be created for a subproblem can be restricted with a time limit or a cut count limit and rather than adding the cut to the subproblem and resolving the relaxed problem to update the objective value after each cut is constructed, created cuts can be added to the subproblem at once.

We also would like to note that the order of the cuts or the condition on the type of cuts to be separated may play a significant role in the efficiency of the procedure because the separation algorithms may function better with particular graphs properties such as graph density or a cut created for a certain type can be used in the other.

Therefore the final cut generation procedures will be explained in Section 9 after separation details are provided in the next subsections.

Algorithm 2 Algorithm: Add cuts

Input: Subgraph $G^{(r)}$, relaxed problem $APC^{(r)}_{Relaxed}$, $SeparationTimeLimit$
Output: Updated $obj(APC^{(r)}_{Relaxed})$
$F = \emptyset$
while $t < SeparationTimeLimit$ **do**
  $f_i \leftarrow SeparateCut(G^{(r)}, sol(APC^{(r)}_{Relaxed}))$
  if $f_i \neq \emptyset$ **then**
    $F \leftarrow F \cup f_i$
    $i \leftarrow i + 1$
  end if
end while
if $F \neq \emptyset$ **then**
  obj$(APC^{(r)}_{Relaxed}) \leftarrow UpdateUpperBound(APC^{(r)}_{Relaxed}, F)$
end if
Return $sol(APC^{(r)}_{Relaxed})$

Figure 7.1. Algorithm: Add cuts.

For each subproblem valid inequalities are separated and previously found inequalities are not considered to be added in the subproblem as the conflict graph of the parallel branching nodes may change drastically and finding the violated inequalities among the cumulative set of valid inequalities most likely to result in the loss of CPU time compared to the upper bound improvements.

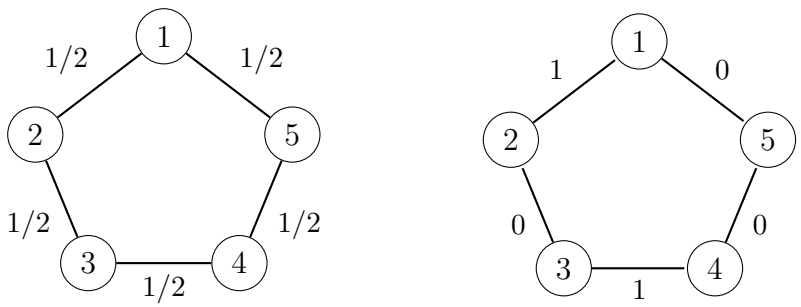
7.1. Odd Cycle Inequalities

Given a conflict graph C, consider an odd cycle with length $2k + 1$. From such cycle, it is possible to select at most k nodes not to violate the stable set property.

However, as stated and described in [22], within a feasible solution to the linear programming relaxation of $MWSSP$ one can find an odd cycle where this condition is violated as in the case in Figure 7.2(a). Therefore, for each odd cycle OC detected in the conflict graph, we can add the following valid inequality as a cut if it is violated:

$$\sum_{e \in V(OC(C))} x_e \leq \frac{|V(OC(C))| - 1}{2}. \tag{7.1}$$

Addition of such inequalities to the subproblem prevents having solutions similar to Figure 7.2(a) and forces the decision variables to take values close to their upper and lower bounds as in the case depicted with Figure 7.2(b). Rebenack S. (2008) states in [22] that separation of such odd-cycle inequalities can be done in polynomial time and its complexity is $O(|V(C)||E(C)| \log |V(C)|)$. The suggested separation algorithm creates a new graph G by duplicating the graph G on which an odd cycle is to be detected and defines an edge between each original node v and its copy v' to determine the shortest path from v to v'. If such a path can be found and if its length is even, there is an odd cycle in G that contains node v.



(a) Solution without odd-cycle inequality

(b) Solution with odd-cycle inequality

Figure 7.2. $MWSSP_{Relaxed}$ feasible solutions.

Similar approach whose steps are given in Figure 7.3 is followed to separate an odd cycle from the conflict graph. After a relaxed subproblem is solved, the node with maximum degree, v, in the subgraph of the original bipartite graph is detected.

Later, the edge $e \in \delta_G(v)$ whose decision variable has the highest fractional value is found to build an odd-cycle to end up with a valid inequality that is more likely to be violated.

Algorithm 3 Create odd cycle

1: Input: Subgraph $G^{(r)}$, relaxed problem $APC^{(r)}_{Relaxed}$
2: Output: An odd cycle OC
3: $v \leftarrow FindMaximumDegreeNode(G^{(r)})$
4: $e \leftarrow FindMaximumWeightDeltaEdge(G^{(r)}$,v)
5: $f \leftarrow FindMaximumWeightSigmaEdge(G^{(r)}$,e)
6: $G^{(r)} \leftarrow G^{(r)}[\{e, f\}]$
7: $(prev, PathLength) \leftarrow Dijkstra(G^{(r)}, APC^{(r)}_{Relaxed}$,$e$,$f)$
8: **if** PathLength $\equiv 0$ (mod2) **then**
9: $OC \leftarrow e$
10: $i \leftarrow f$
11: **while** $i \neq e$ **do**
12: $OC \leftarrow OC \cup i$
13: $i \leftarrow prev[i]$
14: **end while**
15: *Return OC*
16: **else**
17: *Return* $\emptyset$
18: **end if**

Figure 7.3. Algorithm: Create odd cycle.

Once the edge is determined, another edge $f \in \sigma_G(e)$ whose decision variable has the highest fractional value among other edges in $\sigma_G(e)$ is selected. As the found edges are in the original graph and the odd cycle is to be build in the conflict graph, a path with even length, if exists, between e and f where $e, f \in V(C)$ is found. This is done by deleting the edge $\{e, f\}$ and running the Dijkstra algorithm on graph $C[\{e, f\}]$ with weights used in [22].

7.2. Odd Hole Inequalities

It was shown in [24] that if the separated odd cycle has no chords, i.e., if the separated odd cycle is an odd hole, it is facet-defining for the stable set polytope. As it is stated in [22], one can remove the chord of the odd cycle that is separated from a subgraph during B&C solution process to have a dominated valid inequality by splitting the odd cycle into two, an odd cycle and an even cycle, and checking if the created valid inequality constructed using the odd-cycle is violated. That is why, the odd hole creation algorithm described in Figure 7.4 uses the odd cycle creation algorithm presented with Figure 7.3.

Algorithm 4 Create odd hole

Input: Subgraph $G^{(r)}$, relaxed problem $APC^{(r)}_{Relaxed}$, ChordNumberDeletionLimit
Output: An odd hole OH
$OH \leftarrow CreateOddCycle(G^{(r)}, APC^{(r)}_{Relaxed})$
$Count = 0$
while $Count < ChordNumberDeletionLimit$ **do**
  $e = (u, v) \leftarrow FindChord(\text{OH})$
  if $e = \{\}$ **then**
    break
  else
    $OH \leftarrow SplitCycle(OH, u, v)$
    $Count \leftarrow Count + 1$
  end if
end while
Return OH

Figure 7.4. Algorithm: Create odd hole.

7.3. Odd Wheel Inequalities

Given an odd cycle or an odd hole in a conflict graph, denoted by OC one can determine if it is possible to construct an odd wheel using OC via checking if there is a node $v \in V(C) \backslash OC$ such that for each node $u \in OC$, $\{u, v\} \in E(C)$.

Algorithm 5 Create odd wheel

```
1: Input: Subgraph G^(r), relaxed problem APC^(r)_Relaxed
2: Output: An odd wheel OW
3: Center = {}
4: OW ← CreateOddHole(G^(r),APC^(r)_Relaxed)
5: if OW ≠ ∅ then
6:    v ← PickMinDegreeNode(V(G^(r,i))
7:    for e ∈ δ_{G^(r,i)}(v) do
8:       if IsCenterOfTheCycle(OW,e) = 1 then
9:          Center = e
10:         break
11:      end if
12:   end for
13:   if Center ≠ {} then
14:      Return OW ∪ Center
15:   end if
16: end if
17: Return ∅
```

Figure 7.5. Algorithm: Create odd wheel.

During a feasible solution construction to MWSSP, selecting the *center* node to be in the stable set forces all the remaining nodes in the wheel to be left out from the solution. The inequality,

$$\frac{|V(OC)|-1}{2} x_v + \sum_{e \in V(OC)} x_e \quad \leq \quad \frac{|V(OC)|-1}{2}, \tag{7.2}$$

is obtained, if this property is integrated to the odd cycle inequalities. It is important to note that wheel equalities dominate the odd hole thus odd cycle inequalities. Yet, the complexity of the separation of the inequalities is $O(|V(C)||E(C)|\log|V(C)|)$ or $O(|V(C)|^4)$ [22]. Therefore, instead of looking for odd wheel inequalities alone, what may be carried out for separation is separating odd cycles and checking if the odd cycle has any chord or a center and creating a cut for the most dominant violated inequality among the possible inequalities. The odd wheel creation algorithm that utilizes odd cycle creation is provided with Figure 7.5.

7.4. Maximal Clique Inequalities

In a feasible solution to a stable set problem, there can be at most one selected node for each clique $K \in \mathcal{K}(C)$ as each pair of nodes in a clique is adjacent. Hence, during a B&C solution for APC, the following valid inequalities separated from the conflict graph can be added as a cut in the subproblem when it is found to be violated:

$$\sum_{e \in K} x_e \leq 1 \qquad K \in \mathcal{K}(C). \tag{7.3}$$

However, one should detect maximal cliques of the graph instead of detecting all possible cliques to add the violated ones as cuts, because a subset of a clique is a clique as well and as a consequence the inequality created by a maximal clique dominates the inequalities formed from the cliques contained within.

Even so, the number of maximal cliques in a graph is exponential thus the separation of all maximal cliques to create maximal clique cuts require significant amount of CPU time hence, finding the maximal cliques that are more promising to detect valid inequalities with most possible violation and with greater effect to the global upper bound becomes a priori. Finding such promising cliques can be achieved by increasing the size of the cliques as much as possible so that the difference between the right and left side of the inequality (7.3) becomes wider.

The procedure to find a maximal clique is given in Figure 7.6. Similar to the strategy followed to find an odd cycle, one can start building the maximal clique from the node, u, with highest degree in the original graph G to find cliques with comparably larger sizes. Node u has $|d_{C^{(r)}}v| - many$ neighbors and each edge $e \in \delta_{G^{(r)}}(u)$ has $|\sigma_{G^{(r)}}(e)| - many$ neighbors and $|N_{C^{(r)}}(e)| - many$ conflicts. If there is an edge f such that it is in at least 3 sets of $\sigma_{G^{(r)}}(e_i) \cup N_{C^{(r)}}(e_i)$ where $e_i \in \delta_{G^{(r)}}(v)$, a clique can be constructed in conflict graph with the nodes $\{e_i\} \cup f$.

Algorithm 6 Create maximal clique

Input: Conflict graph $C^{(r)}$, relaxed problem $APC^{(r)}_{Relaxed}$
Output: A maximal clique
$v \leftarrow FindMaximumDegreeNode(C^{(r)})$
for $e \in \delta_{C^{(r,i)}}(v)$ **do**
  for $f \in \sigma_C(e_i) \cup N_C(e_i)$ **do**
    if $x_f > 0$ $and f \in \delta_{C^{(r,i)}}(v)$ **then**
      $Count[f] \leftarrow Count[f] + 1$
      $AdjacentEdges[f] \leftarrow AdjacentEdges[f] \cup e$
    end if
  end for
end for
$h \leftarrow FindMaximumCountEdge(Count)$
if $AdjacentEdges[f] < 3$ **then**
  $Return\ \emptyset$
else
  $Return\ AdjacentEdges[h]$
end if

Figure 7.6. Algorithm: Create maximal clique.

7.5. Nodal Inequalities

The strong formulation of the APC problem (APC_s) is determined to be used as the IP formulation of the proposed B&C solution procedure. On the other hand, weak conflict inequalities described with (4.8) can be added as valid inequalities if they are found to be violated in the subproblem. Letchford et. al. defines the following form of inequalities as the *nodal inequalities* in [25]:

$$\alpha(G\backslash S)x_i + \sum_{j \in S} x_j \leq \alpha(G\backslash S), \tag{7.4}$$

for some $i \in E(G)$, $S \subseteq V(G)$ where $\alpha(G\backslash S)$ is the *stability number* of $G\backslash S$. In the study the authors imply that compared to weak conflict inequalities, nodal cuts are more likely to be successful for improving the global upper bound of the stable set problem. A nodal inequality can be constructed in conflict graph C if for a subset S, the stability number is known and there is node $i \in V(C)\backslash S$ such that for each node $j \in S$, $\{i, j\} \in E(C)$. Considering that finding the stability number is $\mathcal{NP}$-hard [25], constructing the nodal inequalities and finding violated ones is not easy. Therefore, they cannot simply replace weak conflict inequalities and they can be constructed using a greedy heuristic.

Since it is necessary to calculate the stability number while creating a nodal inequality, it is useful to increase the stability number subset S in a controlled manner starting from the value 1. We know that the stability number α_K of a clique K is 1 hence, a greedy heuristic can start building S by using a clique. As *nodal inequalities* dominate *clique inequalities*, once a maximal clique is found, a *nodal inequality* can be constructed.

After setting α_K to 1, the node, $e \in K$ that has the highest degree in the original graph is found. Later, another node f is selected from the set $N_c(e)|K$ and α_K is increased to 2 because if α_K remains as 1, the initial clique will not be maximal.

Similar steps are taken for additional nodes $h \in N_c(e)|(K \cup f)$ but that time it should be checked if the stability number is increased via controlling whether there is an edge $\{f,\ h\}$ is in the subgraph. Next, the nodal inequality is constructed using e and $S = (K \cup f \cup \{h\})\backslash e$. The overall procedure is provided in Figure 7.7.

Algorithm 7 Create nodal set

Input: Conflict graph $C^{(r)}$, relaxed problem $APC^{(r)}_{Relaxed}$, $SetSizeLimit$
Output: A nodal set S, α_S
$\alpha_S = 0$
$S \leftarrow$ *CreateMaximalClique*($C^{(r)}$,$APC^{(r)}_{Relaxed}$)
$e \leftarrow$ *FindMaximumDegreeEdge*($C^{(r)}$,S)
if $|S| > 3$ **then**
  while $|S| \leq SetSizeLimit$ **do**
    Pick node $f \in \sigma_{C^{(r,i)}}(e)\backslash S$
    $S \leftarrow S \cup f$
    $\alpha_S \leftarrow$ *CalculateStabilityNumber*(S)
  end while
  Return (S, α_S)
else
  Return $(\emptyset, \alpha_S)$
end if

Figure 7.7. Algorithm: Create nodal set.

8. PROCEDURES TO IMPROVE THE EFFICIENCY

The additional procedures, feasibility test and probing, are used to decrease the number of branching nodes that are evaluated in the B&C solution procedure and improve the overall performance. These algorithms are run on each candidate branching node for a limited amount of time and the nodes are accepted to be included for *bound* procedure later on depending on the outcomes obtained with these algorithms.

8.1. Feasibility Test

Feasibility test is the set of sequential implications of the decision variable values determined in the branching or in the feasibility test itself. Each decision variable represents the selection decision of an edge on the graph and is associated with a node at its each end. It means that, determining an edge as selected to be in the solution reserves these two nodes for the very edge forcing both the neighbor and the conflicting edges not to be selected in the same solution. The necessity can be represented in the graph via deleting the neighbors and the conflicting edges.

However, deletion may lead to having nodes which have no neighbor edge in the subproblem. In that case, one can quickly conclude that the problem is infeasible as an assignment requires all nodes to be assigned to another and can prune the branching node without the need to bound at first. On the other hand, if one finds a node with degree 1, the neighbor edge must be in the solution for the node to be included in the assignment. Finding such node may bring about another set of implications because its selection leads to the deletion of its neighbors and the conflicting edges from the graph.

The iterations stop when a feasible solution is reached or a node with degree 0 is found or no node with degree1 or 0 is left in the induced subgraph.

Only for the last case the branching node is kept within the B&C procedure and even then feasibility test may decrease the additional branching node creations once additional decision variables whose edges are determined to be selected within the feasibility test are assigned to their upper bounds. The complete feasibility test algorithm is presented with the Figure 8.1.

Algorithm 8 Feasibilty test

Input: Subgraph $G^{(r,i)}$ for branching node i, relaxed problem $APC^{(r,i)}_{Relaxed}$
Output: Feasibility Status
$deg \leftarrow DetermineNodeDegrees(G^{(r,i)})$
while $min(deg) = 1$ *or* $min(deg) = 0$ **do**
  if $min(deg) = 0$ **then**
    Return Infeasible
  else
    for $u \in G^{(r,i)}$, $deg(u) = 1$ **do**
      $e \leftarrow FindSingleNeighborEdge(G^{(r,i)},u)$
      $deg(u) = -1$
      for $f \in \sigma_{G^{(r,i)}}(e) \cup N_{G^{(r,i)}}(e)$ **do**
        $(G^{(r,i)}, deg, APC^{(r,i)}) \leftarrow DeleteEdge(G^{(r,i)}, f, deg)$
      end for
    end for
  end if
end while
$obj \leftarrow CalculateObjective(APC^{(r,i)}_{Relaxed})$
if $max(deg) > 1$ **then**
  Return Feasible
else
  if $LB < obj$ **then**
    $LB \leftarrow obj$
  end if
  Return Integral
end if

Figure 8.1. Algorithm: Feasibility test.

8.2. Probing

Feasibility test iteratively detects nodes in the subgraph with degree either 0 or 1 by examining the graph induced by the selection of additional edges. It is also possible to perform what-if analysis after the feasibility test is completed with a variation of the feasibility test algorithm.

Once there is no node with degree 0 or 1 left in the subgraph, instead of adding the branching node to the active branching nodes list one can check if the problem remains feasible after a free variable is set to its upper (lower) bound. Setting the value of the decision variable initiates the feasibility test procedure again. If the problem is found to be infeasible with the latter test, it can be concluded that the decision variable must not be assigned to its upper (lower) bound to have a feasible solution and its value is determined to be 0 (1).

Algorithm 9 Probing

1: Input: Subgraph $G^{(r,i)}$ for branching node i, relaxed problem $APC^{(r,i)}_{Relaxed}$, *Probing-TimeLimit*
2: Output: Updated subproblem $APC^{(r,i)}_{Relaxed}$
3: **if** $FeasibilityTest(G^{(r,i)}, APC^{(r,i)}_{Relaxed}) = Feasible$ **then**
4: **while** *ProbingTimeLimit is not reached* **do**
5: Pick node $v \in V(G^{(r,i)})$
6: $(G^{(r,i,v)}, APC^{(r,i,v)}_{Relaxed}) \leftarrow SelectEdge(G^{(r,i)}, APC^{(r,i)}_{Relaxed}, \mathrm{v})$
7: **if** $FeasibilityTest(G^{(r,i,v)}, APC^{(r,i,v)}_{Relaxed}) = Infeasible$ **then**
8: $(G^{(r,i)}, APC^{(r,i)}_{Relaxed}) \leftarrow RemoveEdge(G^{(r,i)}, APC^{(r,i)}_{Relaxed}, \mathrm{v})$
9: **end if**
10: **end while**
11: **end if**
12: *Return* $APC^{(r,i)}_{Relaxed}$

Figure 8.2. Algorithm: Probing.

The whatif analysis procedure is called *probing* whose detailed steps are given in Figure 8.2. Probing can be called for each branching node created for a limited amount of time. As assigning a decision variable x_e to its lower bound results in deletion of only the edge e from subgraph while assigning the decision variable to its upper bound results in deletion of the set $\sigma_G(e) \cup N_C(e)$, one may prefer to assign decision variables to 1 during probing.

Which node to choose from the graph to run probing is determined through the degree of the nodes updated after feasibility test and former probing algorithms on the subproblem. The lower the node degree, the more likely to have an edge selection decision effective enough to make the subproblem infeasible.

9. COMPUTATIONAL EXPERIMENTS AND RESULTS

Computational experiments are conducted to evaluate the performance of the proposed branching rules, cuts and additional algorithms in the B&C and B&B procedures. Final approaches are built upon these results. Experiments are run on randomly generated bipartite graphs with predefined graph and conflict densities. All algorithms are coded in C++ language and each experiment is run in the same computer with processor Intel(R) Xeon(R) CPU E5-2650 w4 @ 2.20 GHz and 288 GB RAM. As for the commercial solver to be utilized within solution approaches and to compare the performances, the state-of-art commercial solver Gurobi 9.5.1 is used and the number of threads allocated to Gurobi is not explicitly changed but rather all available 24 threads are enabled. On the other hand, each proposed solution is run in a single thread. Furthermore, each run is limited to 3600 seconds and for the cases where optimality is reached, lower bound and optimality gap values are recorded (if applicable).

To assess the performance of proposed approaches *performance profile* suggested in [26] is utilized along with average CPU time comparisons. The performance profile is presented on graph where x-axis is formed with τ values and the y-axis values correspond to the fraction of instances whose $\frac{instanceCPUtime}{bestCPUtime}$ value is less than or equal to 2^{τ}. We can also interpret the y-axis values as the probability of having a test instance whose value of $\frac{instanceCPUtime}{bestCPUtime}$ ratio is less than or equal to τ which is denoted by $prob(\frac{instanceCPUtime}{bestCPUtime} \leq \tau)$.

The computational experiments are presented in the following order. First the branching rules are compared and the effect of the feasibility test and probing on their performance is evaluated. Afterwards, the most efficient branching rule is selected and it is compared with Gurobi's own branching scheme. Later, the performance of the cuts are evaluated by grouping similar cuts. Based on the result obtained with these experiments, the final procedures are determined and their performance is compared with the default Gurobi MILP solver.

9.1. Test Instance Generation

The test instance set on which experiments are performed is created randomly before computational experiments. The created problem is expressed as a graph which has distinct *Node Size - Graph Density - Conflict Density - Initial Solution Size* combination referred with N, ϕ, ω and ρ symbols respectively. The parameter N is the size of the nodes in the test instance graph G, thus it is calculated with $|V(G)|$. We used the graph density to determine the number of edges, $|E(G)|$, in the graph G with formula $\phi^{\frac{N(N-1)}{2}}$ whereas we used the conflict density to determine the conflicting edge pairs in the graph with the formula $\omega^{\frac{|E(G)|(|E(G)|)}{2}}$. The list of all parameter values are included in Table 9.1. A total of, $5{\times}3{\times}3{\times}3 = 135$ problem instances are created for test runs. While the comparisons of B&B procedures are realized with $\rho = 1$, experiments on B&C procedures and the comparison of B&B and B&C procedures are accomplished with $\rho = 1, 2, 3$.

We want to note that the node size used in problem generation is limited to the range 34-60. The motivation behind such decision is that as APC cannot be solved to optimality for larger instances either by the proposed procedures or Gurobi MILP solver, it is hard to retrieve mathematically commentable comparisons. In addition, the range of conflict density parameters may seem to be narrow, however for the larger ω values the amount of iterations to solve APC becomes significantly small thus it becomes again quite hard to distinguish between the performance of the solution approaches.

For each problem instance generation, at first $\rho - many$ feasible solutions are determined and edges in each feasible solution are assigned to minimum possible weights. Afterwards, remaining edge number is calculated by subtracting the number of edges used in the feasible solution(s) from $|E(G)|$ and that many edges are created by preserving the simple bipartite graph structure. Once all remaining edges are created, the conflicting edge pairs are determined in such a way that not two edges in the same feasible solution are defined as a conflicting edge pair.

Table 9.1. Test instance parameters.

	Values
Node Number (N)	34, 40, 44, 50, 60
Graph Density (ϕ)	0.6, 0.8, 1
Conflict Density (ω)	0.15, 0.25, 0.4
Initial Solution Size (ρ)	1, 2, 3

9.2. Branching Rule Selection

As branching strategy, three different branching rules are proposed. To evaluate their own performances without the effect of the cuts created from valid inequalities and determine the rule with the smallest CPU time requirement we first neglect the cuts and solve APC with B&B algorithms. Therefore, for branching rule selection, we removed *Add Cuts* function of the B&C procedure described in Figure 5.1. In addition, we evaluated the performance of branching rules with and without the feasibility test. If the feasibility test is not utilized, together with its outer iteration *Add Valid Branching Nodes* function is removed as well.

Independent of which branching rule is used, while creating a new branching node, at least one decision variable is assigned to its upper or lower bound. Therefore, a branching node inherits some decisions on the value of the decision variables from its parent branching node together with the newly added upper-lower bound fixations. This structure requires to preserve these decisions, to prepare the subproblem and subgraph using the information. With this regard, on a branching node there are two recorded lists: the *excluded* edge set and the *included* edge set. In the subproblem while the decision variables of these two sets are kept fixed, the value of the remaining *free* variables are tried to be determined.

Besides the structure of recording included and excluded edge sets, all functions and data structures within the algorithm is the same for each B&B procedure except *Solve Relaxed Subproblem*, *IsFeasibleSolnToAPC* and *CreateBranchingNodes* functions which are the building blocks to identify a branching strategy. While additional information on the development of the first two functions is provided below, the details of *CreateBranchingNodes* given in Section 6 is sufficient and will not be referred here. We denote the B&B procedures defined with the differentiated functions as APC_EP, APC_CP and APC_MW for the B&B algorithms with edge branching, conflicting pair branching and maximal matching branching rules respectively.

For APC_EP, *Solve Relaxed Subproblem*, i.e., the *bound* procedure solves the linear programming relaxation of the subproblem with Gurobi default LP solver. The function *IsFeasibleSolnToAPC* checks whether there is a decision variable with value strictly between 0 and 1.

For APC_CP and APC_MW, the *bound* procedure solves the subproblem without considering the conflict constraints by using the Hungarian algorithm which is implemented in C++. To find the maximum weight assignment, weights of the free edges are multiplied with -1, the weights of the included edges are defined as $-M$ and the weights of the excluded edges are defined as $+M$ where M is a large number. The function *IsFeasibleSolnToAPC* checks whether there is a pair in the relaxed solution that is in conflict. Given an edge pair the complexity of the search procedure is $O(1)$ as conflict information is kept in the edge-edge adjacency matrix.

We would like to emphasize that while using a commercial solver to solve linear programming relaxation is necessary for APC_EP, it is not required for the other two B&B approaches. In addition, B&B procedure will require less CPU time when fewer branching nodes are evaluated until reached to optimality. Therefore, in the computational experiments we coupled the feasibility test with the branching methodology to select the best performing branching rule.

The branching procedures are compared with each other and also with Gurobi's default MILP solver which is denoted as APC_GRBC to assess their performance and to select the best branching strategy. Based on the results of the test, the average CPU time requirements of each branching rule except the conflicting pair branching is incorporated in the Table 9.2 and the Table 9.3 as the conflict branching rule performed very poorly such that no test instance is solved to optimality within the time limit.

Table 9.2. Average CPU times: branching strategies versus graph density.

N	CPU Time	$\phi = 0.6$	$\phi = 0.8$	$\phi = 1.0$	Avg
34	APC_GRBC	2.57	6.41	16.56	8.51
	APC_EP	51.29	800.68	1616.00	822.65
	APC_MW	3.47	1.71	6.11	3.76
40	APC_GRBC	8.25	44.11	119.51	57.29
	APC_EP	799.92	1457.43	2489.62	1582.32
	APC_MW	6.26	46.50	140.27	64.34
44	APC_GRBC	19.61	199.03	681.54	300.06
	APC_EP	1255.22	2516.78	2713.28	2161.76
	APC_MW	9.71	208.16	646.49	288.12
50	APC_GRBC	84.76	1290.80	2430.07	1268.54
	APC_EP	1455.95	2561.73	3600.00	2539.23
	APC_MW	69.87	1207.10	1257.80	844.92
60	APC_GRBC	3600.00	2585.21	1750.37	2645.19
	APC_EP	2511.09	3428.67	3600.00	3179.92
	APC_MW	1203.23	59.45	1851.69	1038.12
Avg	APC_GRBC				855.92
	APC_EP				2057.18
	APC_MW				447.85

Table 9.3. Average CPU times: branching strategies versus conflict density.

N	CPU Time	$\omega = 0.15$	$\omega = 0.25$	$\omega = 0.40$	Avg
34	APC_GRBC	4.17	10.69	10.68	8.51
	APC_EP	1964.68	483.23	20.04	822.65
	APC_MW	7.11	1.14	3.05	3.76
40	APC_GRBC	87.28	57.90	26.69	57.29
	APC_EP	3184.95	1457.81	104.20	1582.32
	APC_MW	185.62	7.31	0.10	64.34
44	APC_GRBC	648.11	188.32	63.75	300.06
	APC_EP	3600.00	2449.26	436.02	2161.23
	APC_MW	850.52	13.65	0.19	288.12
50	APC_GRBC	2438.13	1208.35	159.14	1268.54
	APC_EP	3600.00	2638.92	1378.76	2539.76
	APC_MW	2469.11	65.23	0.43	844.92
60	APC_GRBC	1511.01	3420.00	3004.57	2645.19
	APC_EP	3600.00	3600.00	2339.76	3179.92
	APC_MW	2401.20	711.30	1.87	1037.72
Avg	APC_GRBC				855.92
	APC_EP				2057.18
	APC_MW				447.85

These tables illustrate the average CPU usage during test instance solutions. Each cell value in the tables is the average CPU time of 3 instances. In addition, a row in the table represents the CPU time averages varying with conflict or graph density. The last column of the tables provides the average CPU time usage of the solution procedure with a specific node size, N. For example, for the test instances with $N = 34$, while average CPU time usage for APC_EP is 822.65, it is 3.76 for APC_MW.

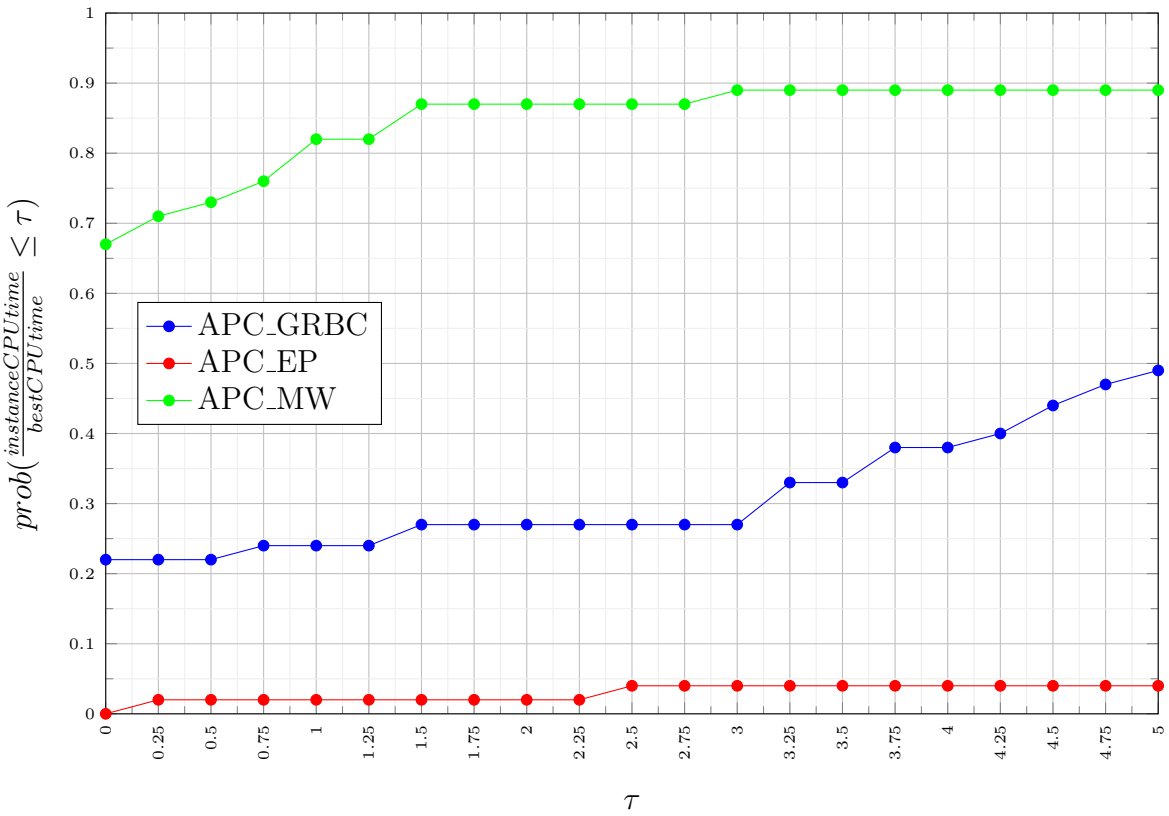


Figure 9.1. Performance comparison of Gurobi MILP solver and Cycle cuts.

It can be observed from the tables that even for the smallest size of N while for APC_GRBC and APC_MW it took on average less than 10 seconds, it was more than 800 seconds for APC_EP. Therefore, we may say that APC_EP performed also poorly and it can be verified with the performance profile illustrated in Figure 9.1.

While the performance of APC_GRBC and APC_EP was much closer, APC_GRBC on average doubled the CPU time required by APC_EP. This means that the commercial solver's default solution strategy which uses preprocessing, cut separation algorithms and heuristics to solve APC failed to find the optimum solutions efficiently compared to the solution procedure which is the combination of maximal matching branching strategy and the feasibility test. Again, the performance profile given with Figure 9.1 supports this inference.

Furthermore, both Table 9.2 and Table 9.3 put forth the effect of graph density and conflict density on the difficulty of the problem instance.

The increase in graph density in overall, resulted in a decrease in the CPU time while the reverse was true for conflict density.

9.3. Branching with Probing

Probing aims to decrease the number of branching nodes evaluated within B&B iterations by making what-if analysis on the values of the decision variables. The algorithm roots from the feasibility test; that is why, it is reasonable to expect improvements on the efficiency of B&B algorithm only when the feasibility test is useful with the selected branching rule.

With the computational results obtained with maximal matching branching, the combination of feasibility test with the branching strategy is shown to be effective as it performed significantly better then Gurobi's default MILP Solver. To assess the contribution of probing to APC_MW, we added the probing functionality into APC_MW and conditionally enabled probing in the branch and bound tree.

Given a subproblem, probing determines at least one free edge and for each such edge, it sets its decision variable to the upper or lower bound then runs a feasibility test to determine whether the assignment leads to a feasible or an infeasible solution. Since in the higher levels of the branching tree there are a large number of free edges whose assignment will less likely to lead to obtaining restrictions on the value of decision variables, the procedure will possibly fail to compensate for the extra time with the benefit it provides. Hence APC_MW with probing (APC_MWP) is conditionally enabled.

In addition, as mentioned in Section 8.2, assigning the decision variable to its upper bound is more likely to be successful to detect feasibilities or infeasibilities. That is why, we decided to assign variables only to their upper bounds within the procedure. On the other hand, the probing condition is determined by using the level of the branching node in the branching tree.

The procedure accepted the branching nodes which are at least $\lceil \frac{|V(G)|}{4} \rceil - many$ levels away from the root node. However, when tests are run on APC_MWP, no decision variable is set to its lower bound with probing. This outcome showed that probing will not have a contribution on the performance of APC_MW and hence additional solution methods and their comparison with APC_MW disregarded the approach.

9.4. Cut Separation with Proposed Branching Rule

Based on the computational results obtained in Section 9.2, it is found that B&B algorithm where maximal matching branching rule and feasibility test are used has significantly better performance than that of Gurobi's default MILP solver. This performance could not be enhanced by probing; however, adding cuts proposed in Section 7 to the subproblems during the solution process may improve the UB thus the overall computation time.To assess the contribution of the separated cuts to the performance, B&C algorithms are formed that use maximal matching branching rule together with a cut group and tests are run on these new procedures. The cut groups are defined to be cycle cuts and clique cuts and corresponding approaches are denoted as APC_MWCycle and APC_MWClique respectively.

Cycle cut group consists of odd cycle cuts, odd hole cuts and odd wheel cuts. As explained in Section 7, odd hole cuts are created using odd cycle cuts and odd wheel cuts are created using the other two. We also know that an odd wheel cut dominates the odd cycle or odd hole cut and similarly an odd hole cut dominates the input odd cycle cut. Hence, one can start building an odd cycle in the subproblem and when such cycle is found, an odd hole and then an odd wheel can be built. The most dominant cut is added to the subproblem whenever it is detected to be violated.

On the other hand, maximal clique cuts and nodal cuts constitute the clique cuts. In order to separate nodal cuts efficiently, they are constructed using maximal cliques and as a consequence, the nodal cuts dominate the maximal clique cuts. Similar to the cycle cut separation, before branching a maximal clique K is built.

If $|K| \geq 3$, a nodal set is constructed. The most dominant violated cut is added to the subproblem.

Apart from the details of the separation procedure, we would like to note that APC_MW does not use Gurobi MILP solver in any part of its algorithm and the complete procedure uses only the combinatorial structure of the problem. However, separated cuts should be added using the Gurobi MILP solver because the subproblem cannot be adjusted with combinatorial algorithms after cuts, i.e., linear inequalities are added. Therefore, before branching, the solution of the relaxed subproblem found with the Hungarian algorithm is defined as the starting point of the IP problem and the cuts are added using the Gurobi's *callback* procedure.

The solution time limit is set to the MILP solver because solving branching nodes is not the main purpose but the focus is on improving the upper bound of the subproblem. In the case of finding an optimal solution of the subproblem, the branching node is pruned directly and the branching process is skipped. During the solution iterations, when Gurobi's callback function is triggered, cuts are separated and added to the subproblem. Gurobi's user cuts are disabled; only cuts added into the subproblems are the ones among the cut groups.

The results of the experiments are summarized with both Table 9.4 and Table 9.5. In contrast to the expectations from adding cuts to improve the upper bound, using the cut callback mechanism deteriorates the performance of the overall solution mechanism for both of the cut groups. The difference between APC_MW and the B&C procedures become larger when N is decreased which reveals the extent of the performance difference even further because due to time limit, larger instances' solution time is restricted but the exact solution time requirement can be fully compared with small sizes of N.

Table 9.4. Average CPU times: maximal matching branching B&C procedures versus graph density.

N	CPU Time	$\phi = 0.6$	$\phi = 0.8$	$\phi = 1.0$	Avg
34	APC_MWCycle	18.21	27.12	217.02	87.45
	APC_MWClique	24.32	26.14	289.94	113.47
	APC_MW	3.47	1.71	6.11	3.76
40	APC_MWCycle	161.31	1286.52	1216.22	888.02
	APC_MWClique	231.71	1300.74	1842.65	1125.04
	APC_MW	6.26	46.50	140.27	64.34
44	APC_MWCycle	703.83	1399.16	2427.71	1510.23
	APC_MWClique	922.68	1399.93	2434.81	1585.81
	APC_MW	9.71	208.16	646.49	288.12
50	APC_MWCycle	1245.02	1965.96	2461.32	1890.77
	APC_MWClique	1258.85	2228.10	2481.73	1989.56
	APC_MW	69.87	1207.10	1257.80	844.92
60	APC_MWCycle	2406.92	2499.04	2843.57	2583.17
	APC_MWClique	1727.25	2487.39	3600.00	2604.88
	APC_MW	1203.23	59.45	1851.69	1038.12
Avg	APC_MWCycle				1391.93
	APC_GRBClique				1483.75
	APC_MW				447.85

Table 9.5. Average CPU times: maximal matching branching B&C procedures versus conflict density.

N	CPU Time	$\omega = 0.15$	$\omega = 0.25$	$\omega = 0.4$	Avg
34	APC_MWCycle	134.96	121.89	5.49	87.45
	APC_MWClique	206.76	127.32	6.33	113.47
	APC_MW	7.11	1.14	3.05	3.76
40	APC_MWCycle	2555.75	91.40	16.90	888.02
	APC_MWClique	2625.07	729.96	20.08	1125.04
	APC_MW	185.62	7.31	0.10	64.34
44	APC_MWCycle	3086.47	1410.83	33.40	1510.23
	APC_MWClique	3307.34	1406.98	43.11	1585.81
	APC_MW	850.52	13.65	0.19	288.12
50	APC_MWCycle	3600	1988.14	84.16	1890.77
	APC_MWClique	3600.00	2255.08	113.60	1989.56
	APC_MW	2469.00	65.23	0.43	844.92
60	APC_MWCycle	3600	3600	549.52	2583.17
	APC_MWClique	3600.00	2915.97	1298.67	2604.88
	APC_MW	2400.00	711.30	1.87	1038.12
Avg	APC_MWCycle				1391.93
	APC_GRBClique				1483.75
	APC_MW				447.85

The performance profile of the solution procedures is shown with Figure 9.2 and the difference can be observed from the graph as well. However, it does not mean that in general the cut separations in APC consumes more than it contributes. APC_MW procedure did not use the MILP solver until an integration with the solver is required to add the separated cuts to the problem. The time required to initialize a model with starting point and variable bounds is not negligible hence, they contribute to the failure of the B&C algorithms.

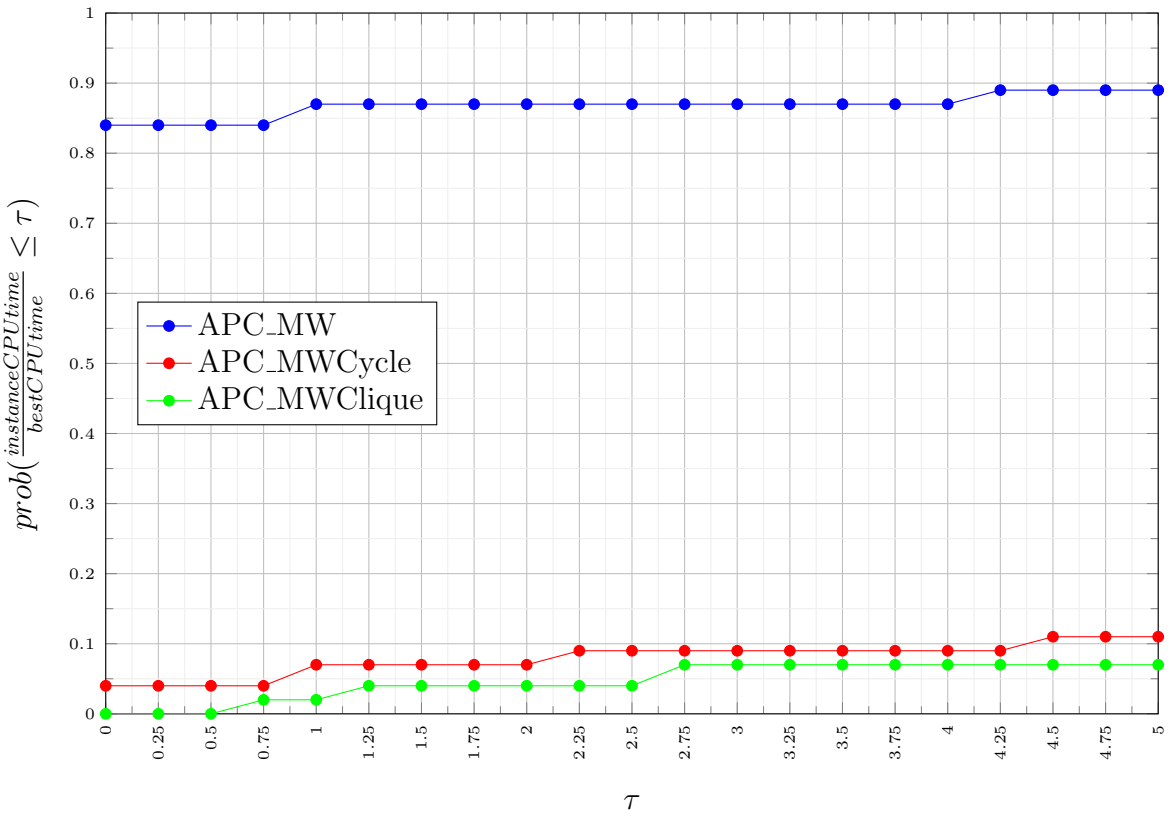


Figure 9.2. Performance comparison of Gurobi MILP solver and Cycle cuts.

9.5. Cut Separation with Default Branching Rule

Having found that using combinatorial branching rule and a MILP solver together may not result in an efficient solution procedure, it may be beneficial to assess the proposed cuts' contribution to the performance of a B&C algorithm when Gurobi's MILP solver's default branching scheme is used. For that purpose, we grouped the cuts and evaluated corresponding B&C algorithms' performances. In addition, a cut combination is formed and its performance with the default branching scheme is assessed.

We want to emphasize that similar to B&C tests using maximal matching branching, when Gurobi's default branching scheme is used, Gurobi user cuts are disabled. It means that the improvement (if there is any) on the UB and on the solution's overall performance can be attributed to the cuts' fitness to APC and the separation efficiency.

To construct cuts, the methods explained in Section 7 are used. The solution method with cycle cuts and clique cuts are denoted as APC_GRBCycle and APC_GRBClique, respectively. On the other hand, the solution method where odd cycle, odd wheel, maximal clique and nodal cuts are used is denoted as APC_GRBCC.

While the order and the condition on the cuts to be separated in a subproblem is determined with the methods described in Subsection 9.4, a combination of the modified versions of the methods is used for APC_GRBCC. The reason behind such strategy is that all cut procedures use the node with the maximum degree; therefore, to maximize the gain obtained from cuts compared to the separation time, we can add only one cut of a type for each subproblem. In the callback algorithm, at first a clique cut is tried to be separated. However, when no violated inequality is found for the cut group, a cycle cut is tried to be built. In addition, to increase the ratio of number of added cycle cuts to the callback algorithm run count, we checked the density of the subgraph and enabled cycle separation only when it is beyond a certain level.

The B&C methods are used to solve before-mentioned problem instances and their performance is compared with APC_GRBC and each other. The number of optimum solutions and the number of solutions which are not solved to optimality with each strategy is given with Table 9.6. While problem instances with node size 34, 40, and 44 are solved to optimality with all solution procedures; this case is not observed for other test instances. Even though there are variations on which problem cannot be solved to optimality between the approaches, the results are similar.

It can easily be observed that when the conflict density is decreased and N is increased, the number of test instances that cannot be solved to optimality decreases. On the other hand, the increase in the graph density resulted in a decrease in optimum solutions.

Table 9.6. Optimum and feasible solution number versus graph density.

		N					
ϕ	Cut Type	34	40	44	50	60	Total
0.6	APC_GRBCycle	9/0	9/0	9/0	9/0	6/3	42/3
	APC_GRBClique	9/0	9/0	9/0	9/0	6/3	42/3
	APC_GRBCC	9/0	9/0	9/0	9/0	6/3	42/3
	APC_GRBC	9/0	9/0	9/0	9/0	6/3	42/3
0.8	APC_GRBCycle	9/0	9/0	9/0	6/3	4/5	37/8
	APC_GRBClique	9/0	9/0	9/0	6/3	5/4	38/7
	APC_GRBCC	9/0	9/0	9/0	6/3	6/3	39/6
	APC_GRBC	9/0	9/0	9/0	6/3	3/6	36/9
1.0	APC_GRBCycle	9/0	9/0	9/0	3/6	1/8	31/14
	APC_GRBClique	9/0	9/0	9/0	5/4	2/7	34/11
	APC_GRBCC	9/0	9/0	9/0	6/3	2/7	35/10
	APC_GRBC	9/0	9/0	9/0	5/4	6/2	39/6
0.15	APC_GRBCycle	9/0	9/0	9/0	3/6	0/9	30/15
	APC_GRBClique	9/0	9/0	9/0	3/6	0/9	30/15
	APC_GRBCC	9/0	9/0	9/0	3/6	0/9	30/15
	APC_GRBC	9/0	9/0	9/0	3/6	0/9	30/15
0.25	APC_GRBCycle	9/0	9/0	9/0	6/3	4/5	37/8
	APC_GRBClique	9/0	9/0	9/0	8/1	5/4	40/5
	APC_GRBCC	9/0	9/0	9/0	9/0	6/3	42/3
	APC_GRBC	9/0	9/0	9/0	7/2	3/6	37/8
0.40	APC_GRBCycle	9/0	9/0	9/0	9/0	7/2	43/2
	APC_GRBClique	9/0	9/0	9/0	9/0	8/1	44/1
	APC_GRBCC	9/0	9/0	9/0	9/0	8/1	44/1
	APC_GRBC	9/0	9/0	9/0	9/0	9/0	45/0

Even though the odd hole cuts are within the cycle cut group and their separation is took place in APC_GRBCycle, no odd holes whose valid inequality is violated is detected within APC_GRBCycle for any of the problem instances.

This may be due to odd holes being the intermediate cut that is tried to be separated after an odd cycle is detected and before an odd wheel is to be constructed. Thus, the number of odd hole cuts found in problem instances are not reported with Table 9.7 and Table 9.8.

Table 9.7. APC_GRBCycle - Average separated cuts versus graph density.

N	Cut Type	$\phi = 0.6$	$\phi = 0.8$	$\phi = 1.0$	Avg
34	Odd Cycle	99	169	326	198
	Odd Wheel	1	3	6	3
40	Odd Cycle	285	1611	4739	2212
	Odd Wheel	2	13	35	17
44	Odd Cycle	526	6927	37076	15148
	Odd Wheel	4	81	219	101
50	Odd Cycle	1191	24756	7128	11025
	Odd Wheel	10	171	114	98
60	Odd Cycle	8454	4315	5980	6250
	Odd Wheel	76	73	114	88

Table 9.8. APC_GRBCycle - Average separated cuts versus conflict density.

N	Cut Type	$\omega = 0.15$	$\omega = 0.25$	$\omega = 0.4$	Avg
34	Odd Cycle	254	312	28	198
	Odd Wheel	1	7	1	3
40	Odd Cycle	6021	568	46	2212
	Odd Wheel	36	13	1	17
44	Odd Cycle	46919	1961	93	15148
	Odd Wheel	251	49	4	101
50	Odd Cycle	30493	2473	111	11025
	Odd Wheel	214	75	5	98
60	Odd Cycle	15580	2851	318	6250
	Odd Wheel	148	103	12	88

These tables, on the other hand, show the number of odd cycle and odd wheel cuts that are separated throughout the solutions using APC_GRBCycle to examine the effect of the graph and conflict density on the separated cut number. Even for the smallest and the most sparse graph ($N = 34, \phi = 0.6$) a considerable number of odd cycle cuts are separated. This number is increased with the increase in the graph size obtained by larger N and ϕ values. Even though this relation does not seem to be linear when node size is increased to 50 and 60, this result may be due to reaching the time limit without having a chance to separate cuts for each active branching node. The same results are obtained for the odd wheel cuts except the fact that the number of odd wheel cuts separated is significantly smaller than that of the odd cycle cuts.

When the number of cut separations is listed with different values of conflict density as it is presented with Table 9.8, unlike graph density, the increase in conflict density resulted in a decrease in the number of separated odd cycle or odd wheel cuts. This is also the direct implication of the combinatorial property that, within a graph with large conflict density, selecting an edge e leads to removal of $\delta_G(e) - many$ edges from the conflict graph.

Similar to the results obtained for cycle cuts created within APC_GRBCycle, both maximal clique and nodal cuts tend to increase with larger N and ϕ values as presented in Table 9.9 and Table 9.10. However, on average the number of maximal cliques separated within APC_GRBClique is much less than the number of odd cycles separated within APC_GRBCycle. This is also true for nodal cut and odd wheel cut comparison. Therefore, the average number of clique cuts and cycle cuts created in a branching node is significantly different.

We want to also note that considering the results presented with Table 9.10, the relation between the value of ω and the number of created clique cuts are not straightforward as in the case of cycle cuts.

Table 9.9. APC_GRBClique - Average separated cuts versus graph density.

N	Cut Type	$\phi = 0.6$	$\phi = 0.8$	$\phi = 1.0$	Avg
34	Maximal Clique	1	6	17	8
	Nodal	3	2	3	2
40	Maximal Clique	4	24	74	34
	Nodal	3	25	50	26
44	Maximal Clique	4	51	128	61
	Nodal	3	49	97	49
50	Maximal Clique	10	95	232	112
	Nodal	3	82	129	72
60	Maximal Clique	59	150	303	171
	Nodal	3	12	27	14

Table 9.10. APC_GRBClique - Average separated cuts versus conflict density.

N	Cut Type	$\omega = 0.15$	$\omega = 0.25$	$\omega = 0.4$	Avg
34	Maximal Clique	2	9	13	8
	Nodal	7	1	0	2
40	Maximal Clique	46	24	33	34
	Nodal	77	0	1	26
44	Maximal Clique	91	46	47	62
	Nodal	148	0	1	49
50	Maximal Clique	151	116	70	112
	Nodal	214	0	1	72
60	Maximal Clique	109	198	205	171
	Nodal	42	0	0	14

Table 9.11 and Table 9.12 introduce the results on the number of cuts that are separated during solutions of APC_GRBCC which combined the cuts of the cut groups. The number of separated nodal cuts is increased when APC_GRBCC is used compared to the average number of nodal cuts separated in APC_GRBClique. However, the reverse is true for all remaining cut types.

In fact, the decrease in the cut separations is expected especially for the cycle cuts which are conditionally separated in the subproblem.

Table 9.11. APC_GRBCC - Average separated cuts versus graph density.

N	Cut Type	$\phi = 0.6$	$\phi = 0.8$	$\phi = 1.0$	Avg
34	Odd Cycle	101	150	210	160
	Odd Wheel	1	6	8	5
	Maximal Clique	2	6	17	9
	Nodal	3	3	4	4
40	Odd Cycle	262	610	323	404
	Odd Wheel	4	8	11	8
	Maximal Clique	3	13	64	29
	Nodal	5	7	80	33
44	Odd Cycle	366	1220	265	661
	Nodal	4	19	10	12
	Maximal Clique	5.	27	138	58
	Nodal	3	18	213	88
50	Odd Cycle	1008	341	69	543
	Odd Wheel	12	10	1	9
	Maximal Clique	10	97	253	120
	Nodal	4	204	325	192
60	Odd Cycle	2279	113	37	1183
	Odd Wheel	33	3	1	20
	Maximal Clique	46.	147	321	171
	Nodal	6	52	64	43

Table 9.12. APC_GRBCC - Average separated cuts versus conflict density.

N	Cut Type	$\omega = 0.15$	$\omega = 0.25$	$\omega = 0.4$	Avg
34	Odd Cycle	118	283	39	160
	Odd Wheel	1	7	3	5
	Maximal Clique	2	11	19	9
	Nodal	4	3	0	4
40	Odd Cycle	693	403	77	404
	Odd Wheel	9	9	6	8
	Maximal Clique	23	21	43	29
	Nodal	50	2	2	33
44	Odd Cycle	1974	375	72	661
	Odd Wheel	19	12	4	12
	Maximal Clique	86	42	47	58
	Nodal	142	2	2	88
50	Odd Cycle	1472	380	68	543
	Odd Wheel	13	13	2	9
	Maximal Clique	166	111	84	120
	Nodal	240	2	1	192
60	Odd Cycle	6246	306	81	1183
	Odd Wheel	74	11	5	20
	Maximal Clique	97	193	224	171
	Nodal	59	16	1	43

As for the required CPU times to solve problem instances over graph and conflict density, both Table 9.13 and Table 9.14 present the values with varying node sizes. However, in these tables, cell values present the average CPU time usage of 9 test instances, where it is 3 in the former CPU time comparison tables.

Similar to the trend in the number of separated cuts, CPU times spent during problem solution tend to increase as the N and ϕ values increase for all procedures. On the other hand, the reverse is true for the effect of conflict density on the CPU time except the case $N = 34$ for which the trend is not straightforward.

Table 9.13. Average CPU times: B&C procedures versus graph density.

N		$\phi = 0.6$	$\phi = 0.8$	$\phi = 1.0$	Avg
34	APC_GRBCycle	2.17	8.87	17.80	9.61
	APC_GRBClique	1.81	6.96	13.81	7.52
	APC_GRBCC	1.47	6.13	14.80	7.46
	APC_GRBC	2.12	7.40	18.12	9.21
40	APC_GRBCycle	11.10	55.90	199.69	88.90
	APC_GRBClique	7.89	42.21	157.13	69.08
	APC_GRBCC	7.51	42.73	153.35	67.86
	APC_GRBC	9.67	48.02	154.48	70.72
44	APC_GRBCycle	29.67	252.80	969.56	417.34
	APC_GRBClique	21.12	271.20	587.19	293.17
	MAPC_GRBCC	25.55	275.77	583.29	294.87
	APC_GRBC	24.28	200.61	703.96	309.61
50	APC_GRBCycle	132.29	1421.96	2495.77	1350.01
	APC_GRBClique	84.02	1346.96	2186.45	1205.81
	APC_GRBCC	136.65	1334.31	2100.15	1190.37
	APC_GRBC	138.64	1380.87	2497.99	1339.17
60	APC_GRBCycle	1427.11	2507.21	3525.95	2486.75
	APC_GRBClique	1410.97	2361	3270.76	2347.57
	APC_GRBCC	1396.01	2244.59	3189.58	2276.72
	APC_GRBC	1474.41	2894.11	3242.18	2536.90
Avg	APC_GRBCycle				870.52
	APC_GRBClique				784.63
	APC_GRBCC				767.46
	APC_GRBC				853.12

Table 9.14. Average CPU times: B&C procedures versus conflict density.

N		$\omega = 0.15$	$\omega = 0.25$	$\omega = 0.4$	Avg
34	APC_GRBCycle	4.32	14.00	10.52	9.61
	APC_GRBClique	3.11	11.56	7.91	7.52
	APC_GRBCC	3.17	11.80	7.44	7.46
	APC_GRBC	3.17	13.20	11.27	9.21
40	APC_GRBCycle	145.63	85.88	35.18	88.90
	APC_GRBClique	111.09	63.54	32.61	69.08
	APC_GRBCC	112.32	68.52	22.74	67.86
	APC_GRBC	88.82	76.02	47.34	70.72
44	APC_GRBCycle	806.60	375.08	70.35	417.34
	APC_GRBClique	616.13	206.60	56.78	293.17
	APC_GRBCC	654.74	188.08	41.79	294.87
	APC_GRBC	580.45	263.90	84.49	309.61
50	APC_GRBCycle	2494.27	1396.05	159.70	1350.01
	APC_GRBClique	2453.34	1021.63	142.47	1205.81
	APC_GRBCC	2514.66	905.98	150.47	1190.37
	APC_GRBC	2495.07	1328.01	194.41	1339.17
60	APC_GRBCycle	3600	2399.99	1460.27	2486.75
	APC_GRBClique	3600	2319.89	1122.83	2347.57
	APC_GRBCC	3600.00	2172.28	1057.89	2276.72
	APC_GRBC	3600	2554.08	1456.62	2536.90
Avg	APC_GRBCycle				870.52
	APC_GRBClique				784.63
	MAPC_GRBCC				767.46
	APC_GRBC				853.12

When we compare the average CPU time required to solve the instances, there is no procedure that stands out. However, we can highlight that the average CPU time required for APC_GRBClique is %10 more than the average CPU time of APC_GRBC.

Knowing that compared to APC_GRBCycle, less number of cuts are created within the procedure, we may conclude that clique cuts are more effective to cut off the fractional sections of the relaxed problem's feasible region and also to yield a more efficient solution of APC. To compare the solution procedures in more detail we can use the performance profile depicted with Figure 9.3.

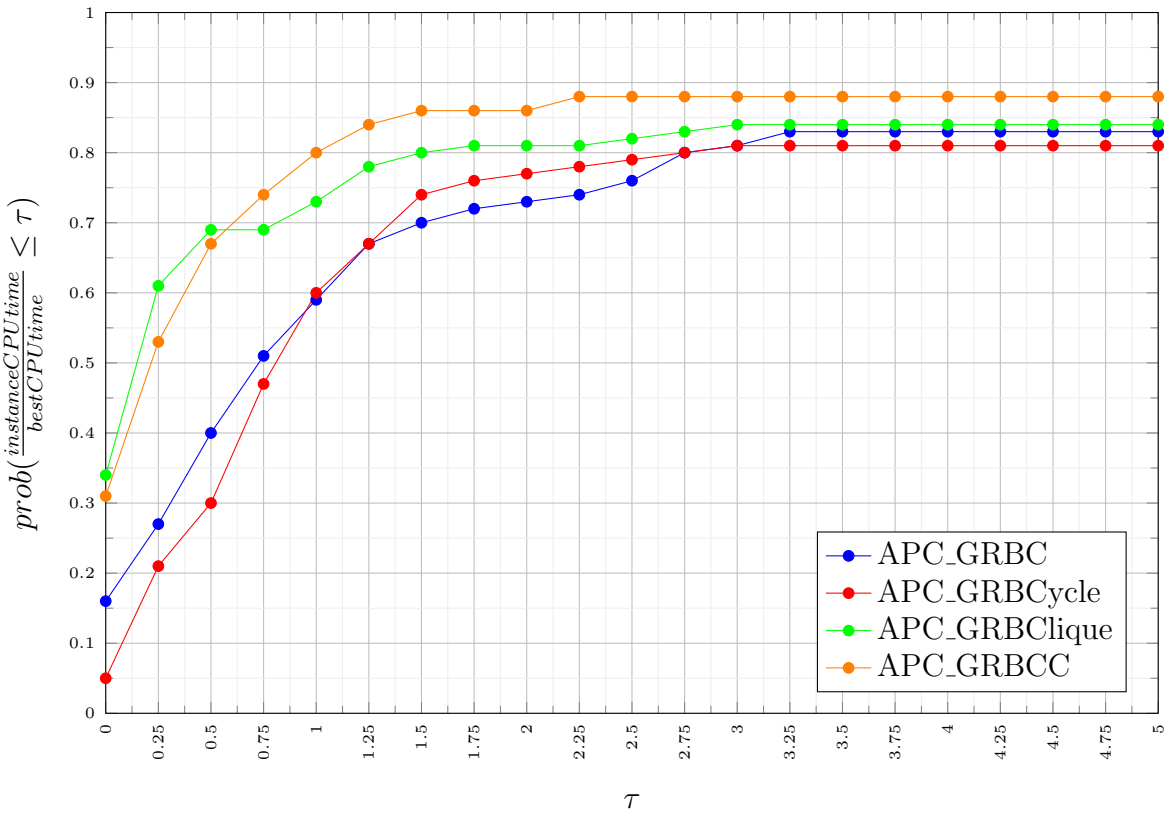


Figure 9.3. Performance comparison of B&C strategies.

Even though there were no striking differences on the average CPU times, the performance profiles reveal the differences on the performances of the solution approaches. While APC_GRBClique is shown to be the most effective procedure when $\tau < 0.5$, APC_GRBCC yields better performance for greater τ values. On the other hand, the difference between the performance of APC_GRBC and APC_GRBCycle is hard to distinguish.

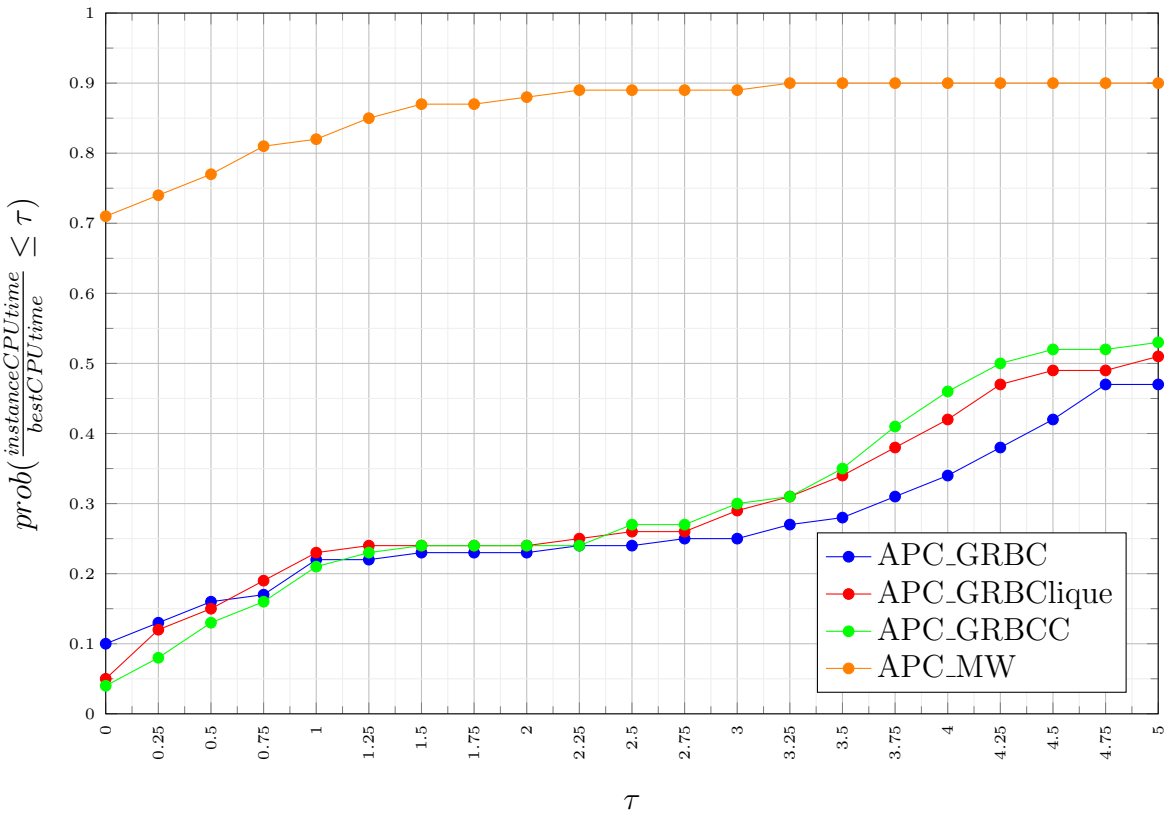


Figure 9.4. Performance comparison of the best solution methods.

Based on the test outcomes, APC_GRBClique and APC_GRBCC are found to be better than the APC_GRBC. This means that two B&C strategies are also determined to be more efficient to solve APC problems compared to the state-of-art MILP commercial solver. We can compare these approaches using the performance profile again; they are presented with Figure 9.4. Even though the proposed B&C strategies are found to be better than APC_GRBC, when they are compared with APC_MW, we can observe the significant difference between the performance of the procedures and APC_MW. While %70 of the problem instances APC_MW solved the problem to optimality in a smaller amount of time, other procedures failed to solve almost %45 of the problem instances within [0,5] τ interval.

10. CONCLUSION

In this thesis, we aim to propose exact solution algorithms to solve the Maximum Weight Assignment Problem with Conflict Constraints (APC). We begin building our approaches by formulating the problem. For that purpose, we use Maximum Weight Assignment Problem (MWAP) and Maximum Weight Stable Set Problem (MWSSP), which are the relaxations of APC. By using the formulations of the relaxed problems we construct *weak*, *strong* and *clique* formulations and prefer to employ the strong formulation in the solution procedures.

As the first step to define exact solution algorithms, we introduce *edge*, *conflicting pair* and *maximal matching* branching rules. These branching rules are utilized with B&B and some B&C strategies. Afterwards, we introduce various cuts and their cut separation procedures all of which are designed to be separated from the conflict graph by using the property that solving APC is realized when MWSSP with cardinality restrictions is solved on the conflict graph.

Besides the branching rules and cut separations, we suggest two heuristics, namely *feasibility test* and *probing* to enhance the performance of branching strategies by proactively pruning branching nodes and making decision variable assignments.

We start evaluating the performance of the procedures with the comparison of branching strategies which are integrated with feasibility test. As a benchmark, we compare their performance with the Gurobi MILP solver with default configurations. The best performing branching rule is found to be the maximal matching branching rule. When it is compared with the commercial MILP solver, it distinguishingly stands out as well.

To understand the effect of the probing on the branching rule, the former B&B algorithm with maximal matching branching rule is compared with its integration with probing. However, no contribution to the success of the solution procedure is observed. Instead, we add proposed cuts to the B&B algorithm by using Gurobi's callback mechanism and observe the result that adding cuts with maximal matching branching strategy deteriorates the overall performance.

Since such outcome can be attributed to using combinatorial branching rule together with cut separations, we decide to use Gurobi default branching strategy and assess the B&C algorithms' performance. Three different B&C algorithms are constructed where the first and second procedure uses cycle and clique cuts respectively, the third procedure uses a mixture of the members of the cut groups. Their performance evaluation reveals that, compared to Gurobi MIP solver, second and third approaches were better.

As the final step, we compare the performance of the best B&B algorithm with the best B&C algorithms. The results show that the B&B algorithm with maximal matching branching rule is significantly better than all other procedures including Gurobi's MILP solver with default configurations.

REFERENCES

1. Burkard, R., M. Dell'Amico and S. Martello, *Assignment Problems*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2008.

2. Bazaraa, M. S., J. J. Jarvis and H. D. Sherali, *Linear Programming and Network Flows*, John Wiley & Sons, Inc., New Jersey, NY, USA, 2010.

3. Darmann, A., U. Pferschy, J. Schauer and G. J. Woeginger, "Paths, Trees And Matchings Under Disjunctive Constraints", *Discrete Applied Mathematics*, Vol. 159, pp. 1726–1735, 2011.

4. Manerba, D. and R. Mansini, "The Nurse Routing Problem with Workload Constraints and Incompatible Services", *IFAC-PapersOnLine*, Vol. 49(12), pp. 1192–1197, 2016.

5. Padberg, M. W., "On the Facial Structure of Set Packing Polyhedra", *Mathematical Programming*, Vol. 5, pp. 199–215, 1973.

6. Darmann, A., U. Pferschy and J. Schauer, "Determining a Minimum Spanning Tree with Disjunctive Constraints", *International Conference on Algorithmic Decision Theory*, pp. 414–423, Springer, 2009.

7. Carrabs, F. and M. Gaudioso, "A Lagrangian Approach for The Minimum Spanning Tree Problem with Conflicting Edge Pairs", *Networks*, Vol. 78, No. 1, pp. 32–45, 2021.

8. Samer, P. and S. Urrutia, "A Branch and Cut Algorithm for Minimum Spanning Trees under Conflict Constraints", *Optimization Letters*, Vol. 9, No. 1, pp. 41–55, 2015.

9. Carrabs, F., R. Cerulli, R. Pentangelo and A. Raiconi, "Minimum Spanning Tree

with Conflicting Edge Pairs: A Branch-And-Cut Approach", *Annals of Operations Research*, Vol. 298, pp. 65–78, 2021.

10. Sadykov, R. and F. Vanderbeck, "Bin Packing With Conflicts: A Generic Branch-and-Price Algorithm", *INFORMS Journal on Computing*, Vol. 25, No. 2, pp. 244–255, 2013.

11. Ficker, A. M. C., F. C. R. Spieksma and G. J. Woeginger, "The Transportation Problem with Conflicts", *Annals of Operations Research*, Vol. 298, pp. 207–227, 2021.

12. Manerba, D. and R. Mansini, "A Branch-and-Cut Algorithm for The Multi-Vehicle Traveling Purchaser Problem with Pairwise Incompatibility Constraints", *Networks*, Vol. 65, No. 2, pp. 139–154, 2015.

13. Gendreau, M., D. Manerba and R. Mansini, "The Multi-Vehicle Traveling Purchaser Problem with Pairwise Incompatibility Constraints and Unitary Demands: A Branch-and-Price Approach", *European Journal of Operational Research*, Vol. 248, No. 1, pp. 59–71, 2016.

14. Öncan, T., R. Zhang and A. P. Punnen, "The Minimum Cost Perfect Matching Problem With Conflict Pair Constraints", *Computers & Operations Research*, Vol. 40, pp. 920–930, 2013.

15. Öncan, T., M. H. Akyüz and İ. K. Altınel, "A Branch-and-Bound Algorithm for the Maximum Weight Perfect Matching Problem with Conflicting Edge Pairs", B. Darties and M. Poss (Editors), *Network Optimization INOC: 9th International Network Optimization Conference*, pp. 31–36, OpenProceedings, 2019.

16. Öncan, T. and İ. K. Altınel, "Iterated Exact And Heuristic Algorithms For The Minimum Cost Bipartite Perfect Matching Problem With Conflict Constraints", *IEEE International Conference on Industrial Engineering and Engineering Man-*

agement (IEEM), pp. 1032–1036, 2017.

17. Öncan, T. and İ. K. Altınel, "A Branch-and-Bound Algorithm for the Minimum Cost Bipartite Perfect Matching Problem with Conflict Pair Constraints", *Electronic Notes in Discrete Mathematics*, Vol. 64, pp. 5–14, 2018.

18. Öncan, T., Z. Şuvak, M. H. Akyüz and İ. K. Altınel, "Assignment Problem with Conflicts", *Computers & Operations Research*, Vol. 111, pp. 214–229, 2019.

19. Croce, F. D. and R. Tadei, "A Multi-KP Modeling for the Maximum-Clique Problem", *European Journal of Operational Research*, Vol. 73, pp. 555–561, 1994.

20. Lovász, L., "Normal Hypergraphs and the Perfect Graph Conjecture", *Discrete Mathematics*, Vol. 2, pp. 253–267, 1972.

21. Chvátal, V., "On Certain Polytopes Associated With Graphs", *Journal of Combinatorial Theory, Series B*, Vol. 18, pp. 138–154, 1975.

22. Rebennack, S., "Stable Set Problem: Branch & Cut Algorithms", *Encyclopedia of Optimization*, Vol. 73, p. 3676–3688, 2008.

23. Kuhn, H. W., "The Hungarian Method for the Assignment Problem", *Naval Research Logistics Quarterly*, Vol. 2, pp. 83–97, 1955.

24. Nemhauser, G. L. and L. E. Trotter, "Properties of Vertex Packing and Independence System Polyhedra", *Mathematical Programming*, Vol. 6, pp. 48–61, 1974.

25. Letchford, A. N., F. Rossi and S. Smriglio, "The Stable Set Problem: Clique and Nodal Inequalities Revisited", *Computers & Operations Research*, Vol. 123, p. 105024, 2020.

26. Dolan, E. D. and J. J. Moré, "Benchmarking Optimization Software With Performance Profiles", *Mathematical Programming*, Vol. 91, No. 2, pp. 201–213, 2002.

APPENDIX A: DETAILED TEST RESULTS ON THE CPU USAGES

The detailed information on the test results regarding the CPU usages of the solution proedures are provided in following tables.

Table A.1. CPU Usages of APC_GRBC, APC_MW, and APC_GRBClique.

N	ϕ	ω	ρ	APC_GRBC	APC_MW	APC_GRBClique
34	1	0.15	1	5.47758	15.462	4.87008
34	1	0.25	1	24.2863	2.769	24.8978
34	1	0.4	1	19.9157	0.096	16.2636
40	1	0.15	1	167.585	409.55	213.039
40	1	0.25	1	130.997	11.073	113.317
40	1	0.4	1	59.9591	0.196	99.8405
44	1	0.15	1	1511.66	1903.597	1514.72
44	1	0.25	1	455.653	35.497	703.57
44	1	0.4	1	77.2997	0.373	85.6296
50	1	0.15	1	3600	3600	3600
50	1	0.25	1	3338.01	172.431	3600
50	1	0.4	1	352.187	0.973	249.84
60	1	0.15	1	3600	3600	3600
60	1	0.25	1	3600	1950.572	3600
60	1	0.4	1	1813.71	4.484	2061
34	0.8	0.15	1	3.1056	4.458	3.87421
34	0.8	0.25	1	5.21336	0.625	11.3355
34	0.8	0.4	1	10.8991	0.042	10.78
40	0.8	0.15	1	77.8691	137.399	128.298
40	0.8	0.25	1	34.8572	2.03	38.5442
40	0.8	0.4	1	19.5924	0.083	18.3813

Table A.1. CPU Usages of APC_GRBC, APC_MW, and APC_GRBClique (cont.).

N	ϕ	ω	ρ	APC_GRBC	APC_MW	APC_GRBClique
44	0.8	0.15	1	406.074	619.451	804.275
44	0.8	0.25	1	94.6872	4.879	95.3608
44	0.8	0.4	1	96.3406	0.14	102.405
50	0.8	0.15	1	3600	3600	3600
50	0.8	0.25	1	186.252	21.044	219.607
50	0.8	0.4	1	86.1621	0.253	70.2838
60	0.8	0.15	1	3600	3.6	3600
60	0.8	0.25	1	3600	173.828	3600
60	0.8	0.4	1	3060	0.933	941.508
34	0.6	0.15	1	3.91724	1.403	4.13346
34	0.6	0.25	1	2.56214	0.011	2.76127
34	0.6	0.4	1	1.22102	9	0.241228
40	0.6	0.15	1	16.3887	9.921	16.6559
40	0.6	0.25	1	7.85855	8.835	6.66758
40	0.6	0.4	1	0.517412	0.017	0.307259
44	0.6	0.15	1	26.5902	28.521	23.9791
44	0.6	0.25	1	14.6222	0.57	17.3275
44	0.6	0.4	1	17.6067	0.044	18.7787
50	0.6	0.15	1	114.401	207.331	125.177
50	0.6	0.25	1	100.798	2.211	89.3167
50	0.6	0.4	1	39.0807	0.059	35.1664
60	0.6	0.15	1	3600	3600	3600
60	0.6	0.25	1	555.631	9.5	399.718
60	0.6	0.4	1	377.394	0.185	225.583
34	1	0.15	2	2.0337	16.382	1.93162
34	1	0.25	2	21.6351	3.697	8.23589
34	1	0.4	2	25.9667	0.112	19.5804

Table A.1. CPU Usages of APC_GRBC, APC_MW, and APC_GRBClique (cont.).

N	ϕ	ω	ρ	APC_GRBC	APC_MW	APC_GRBClique
40	1	0.25	2	210.638	14.32	176.683
40	1	0.4	2	132.913	0.216	14.7043
44	1	0.15	2	905.672	1748.058	755.586
44	1	0.25	2	628.26	41.876	487.054
44	1	0.4	2	192.714	0.405	52.674
50	1	0.15	2	3600	3600	3600
50	1	0.25	2	3600	194.43	1828.09
50	1	0.4	2	355.781	1.658	283.271
60	1	0.15	2	3600	3600	3600
60	1	0.25	2	3600	2272.717	3600
60	1	0.4	2	2832.39	6.581	2175.81
34	0.8	0.15	2	3.41085	6.081	2.96448
34	0.8	0.25	2	14.0788	0.699	10.807
34	0.8	0.4	2	7.21	0.037	6.49243
40	0.8	0.15	2	45.415	100.592	49.5164
40	0.8	0.25	2	49.5795	1.951	8.1308
40	0.8	0.4	2	21.6731	0.098	13.2981
44	0.8	0.15	2	272.496	629.223	363.856
44	0.8	0.25	2	97.3174	5.066	24.7227
44	0.8	0.4	2	63.5191	0.133	61.8177
50	0.8	0.15	2	3600	3600	3600
50	0.8	0.25	2	510.728	21.624	368.963
50	0.8	0.4	2	104.127	0.366	34.636
60	0.8	0.15	2	3600	3600	3600
60	0.8	0.25	2	3600	251.212	2232.54
60	0.8	0.4	2	609.86	1.533	106.236

Table A.1. CPU Usages of APC_GRBC, APC_MW, and APC_GRBClique (cont.).

N	ϕ	ω	ρ	APC_GRBC	APC_MW	APC_GRBClique
34	0.6	0.15	2	2.92344	1.283	2.41802
34	0.6	0.25	2	2.37298	0.089	1.25005
34	0.6	0.4	2	0.246206	0.015	0.232166
40	0.6	0.15	2	19.7625	9.614	16.4297
40	0.6	0.25	2	5.80585	0.336	4.72795
40	0.6	0.4	2	5.03021	0.019	2.17382
44	0.6	0.15	2	27.2648	30.986	40.9683
44	0.6	0.25	2	36.3044	0.599	19.6024
44	0.6	0.4	2	21.549	0.038	3.73509
50	0.6	0.15	2	347.036	225.386	184.699
50	0.6	0.25	2	73.4344	1.698	12.8247
50	0.6	0.4	2	47.7369	0.074	23.867
60	0.6	0.15	2	3600	3600	3600
60	0.6	0.25	2	458.62	3.03	310.048
60	0.6	0.4	2	449.721	0.225	268.553
34	1	0.15	3	2.32191	10.518	2.46806
34	1	0.25	3	34.1616	2.967	33.4229
34	1	0.4	3	27.2928	0.19	12.6025
40	1	0.15	3	176.799	229.045	224.459
40	1	0.25	3	187.174	13.066	177.913
40	1	0.4	3	162.855	0.496	116.768
44	1	0.15	3	1415.52	1827.633	1140.48
44	1	0.25	3	939.115	40.072	394.59
44	1	0.4	3	209.703	0.666	150.423
50	1	0.15	3	3600	3600	3600
50	1	0.25	3	3600	193.119	2557.9
50	1	0.4	3	435.907	1.318	358.982

Table A.1. CPU Usages of APC_GRBC, APC_MW, and APC_GRBClique (cont.).

N	ϕ	ω	ρ	APC_GRBC	APC_MW	APC_GRBClique
60	1	0.15	3	3600	3600	3600
60	1	0.25	3	3600	2040.971	3600
60	1	0.4	3	2933.52	5.618	3600
34	0.8	0.15	3	2.07811	1.957	2.08074
34	0.8	0.25	3	12.1563	0.765	9.59503
34	0.8	0.4	3	8.43753	0.119	4.70491
40	0.8	0.15	3	116.516	56.499	57.0958
40	0.8	0.25	3	48.1253	2.579	41.7999
40	0.8	0.4	3	18.5313	0.234	24.8137
44	0.8	0.15	3	620.645	802.606	861.261
44	0.8	0.25	3	86.4381	7.319	104.314
44	0.8	0.4	3	67.9536	0.327	22.748
50	0.8	0.15	3	3600	3600	3600
50	0.8	0.25	3	448.8	21.63	421.631
50	0.8	0.4	3	291.768	0.58	207.517
60	0.8	0.15	3	3600	3600	3600
60	0.8	0.25	3	3600	176.932	3104.78
60	0.8	0.4	3	777.13	1.78	463.903
34	0.6	0.15	3	3.23594	2.003	3.2577
34	0.6	0.25	3	2.3259	0.22	1.69241
34	0.6	0.4	3	0.270254	0.058	0.281235
40	0.6	0.15	3	17.6067	11.442	16.8161
40	0.6	0.25	3	9.13061	0.516	4.03537
40	0.6	0.4	3	4.96916	0.112	3.17165
44	0.6	0.15	3	38.0918	40.945	40.0725
44	0.6	0.25	3	22.718	1.131	12.8257
44	0.6	0.4	3	13.7445	0.212	12.8127

Table A.1. CPU Usages of APC_GRBC, APC_MW, and APC_GRBClique (cont.).

N	ϕ	ω	ρ	APC_GRBC	APC_MW	APC_GRBClique
50	0.6	0.15	3	394.233	270.594	170.158
50	0.6	0.25	3	94.0407	2.348	96.3296
50	0.6	0.4	3	36.9769	0.375	18.6676
60	0.6	0.15	3	3600	3600	3600
60	0.6	0.25	3	372.487	14.661	431.911
60	0.6	0.4	3	255.826	0.883	262.894

Table A.2. CPU Usages of APC_GRBCycle and APC_GRBCC.

N	ϕ	ω	ρ	APC_GRBCycle	APC_GRBCC
34	1	0.15	1	9.52797	5.81251
34	1	0.25	1	31.9948	15.2814
34	1	0.4	1	17.2704	7.7188
40	1	0.15	1	271.11	270.596
40	1	0.25	1	173.296	127.392
40	1	0.4	1	72.6508	14.297
44	1	0.15	1	1576.47	1892.31
44	1	0.25	1	1186.4	347.518
44	1	0.4	1	76.0446	28.6408
50	1	0.15	1	3600	3600
50	1	0.25	1	3600	2652.82
50	1	0.4	1	303.158	427.956
60	1	0.15	1	3600	3600
60	1	0.25	1	3600	3600
60	1	0.4	1	3600	1315.26
34	0.8	0.15	1	5.30143	4.00735
34	0.8	0.25	1	8.53214	7.51929
34	0.8	0.4	1	12.7757	4.72695

Table A.2. CPU Usages of APC_GRBCycle and APC_GRBCC (cont.).

N	ϕ	ω	ρ	APC_GRBCycle	APC_GRBCC
40	0.8	0.15	1	116.003	144.046
40	0.8	0.25	1	38.7514	13.9357
40	0.8	0.4	1	19.4783	12.2442
44	0.8	0.15	1	553.612	962.078
44	0.8	0.25	1	103.347	170.034
44	0.8	0.4	1	109.845	12.9849
50	0.8	0.15	1	3600	3600
50	0.8	0.25	1	306.773	230.099
50	0.8	0.4	1	97.2453	29.6838
60	0.8	0.15	1	3600	3600
60	0.8	0.25	1	3600	2187.12
60	0.8	0.4	1	800.328	491.582
34	0.6	0.15	1	4.36965	1.75147
34	0.6	0.25	1	2.66222	1.61935
34	0.6	0.4	1	0.247206	0.261219
40	0.6	0.15	1	21.2668	8.49107
40	0.6	0.25	1	7.35815	4.23054
40	0.6	0.4	1	0.308262	0.583485
44	0.6	0.15	1	40.4258	42.8668
44	0.6	0.25	1	20.0958	7.22104
44	0.6	0.4	1	20.8294	4.55081
50	0.6	0.15	1	189.183	207.353
50	0.6	0.25	1	111.56	17.0032
50	0.6	0.4	1	40.0045	17.6067
60	0.6	0.15	1	3600	3600
60	0.6	0.25	1	450.15	64.1707
60	0.6	0.4	1	296.735	351.752

Table A.2. CPU Usages of APC_GRBCycle and APC_GRBCC (cont.).

N	ϕ	ω	ρ	APC_GRBCycle	APC_GRBCC
34	1	0.15	2	3.84318	2.7344
34	1	0.25	2	12.4304	19.2033
34	1	0.4	2	23.5897	16.0001
40	1	0.15	2	406.89	264.814
40	1	0.25	2	199.783	223.423
40	1	0.4	2	12.2912	30.8752
44	1	0.15	2	1930.02	684.786
44	1	0.25	2	842.449	474.269
44	1	0.4	2	22.9872	21.3126
50	1	0.15	2	3600	3600
50	1	0.25	2	3600	2024.12
50	1	0.4	2	281.712	341.862
60	1	0.15	2	3600	3600
60	1	0.25	2	3600	3600
60	1	0.4	2	2933.52	2190.44
34	0.8	0.15	2	3.16161	3.51564
34	0.8	0.25	2	14.5221	11.3595
34	0.8	0.4	2	4.91807	6.07817
40	0.8	0.15	2	105.061	84.4068
40	0.8	0.25	2	11.5647	10.6876
40	0.8	0.4	2	22.5819	13.0313
44	0.8	0.15	2	510.554	272.627
44	0.8	0.25	2	40.3697	52.5785
44	0.8	0.4	2	100.046	74.2974
50	0.8	0.15	2	3600	3600
50	0.8	0.25	2	528.77	256.314
50	0.8	0.4	2	21.7482	33.4846

Table A.2. CPU Usages of APC_GRBCycle and APC_GRBCC (cont.).

N	ϕ	ω	ρ	APC_GRBCycle	APC_GRBCC
60	0.8	0.15	2	3600	3600
60	0.8	0.25	2	2370.64	2385.89
60	0.8	0.4	2	565.474	479.988
34	0.6	0.15	2	2.72528	2.14064
34	0.6	0.25	2	1.45519	1.45
34	0.6	0.4	2	0.252209	0.234379
40	0.6	0.15	2	19.9166	20.672
40	0.6	0.25	2	8.2789	5.60942
40	0.6	0.4	2	2.20381	2.31252
44	0.6	0.15	2	51.362	46.7347
44	0.6	0.25	2	20.0378	22.0783
44	0.6	0.4	2	3.21766	3.59378
50	0.6	0.15	2	349.562	484.269
50	0.6	0.25	2	26.7143	28.4377
50	0.6	0.4	2	25.4383	29.1408
60	0.6	0.15	2	3600	3600
60	0.6	0.25	2	318.148	454.769
60	0.6	0.4	2	253.203	262.486
34	1	0.15	3	2.59217	2.39097
34	1	0.25	3	35.2785	36.4235
34	1	0.4	3	23.6428	27.5991
40	1	0.15	3	236.688	143.323
40	1	0.25	3	264.189	199.753
40	1	0.4	3	160.309	105.691
44	1	0.15	3	1862.77	1133.28
44	1	0.25	3	1024.26	483.684
44	1	0.4	3	204.595	183.852

Table A.2. CPU Usages of APC_GRBCycle and APC_GRBCC (cont.).

N	ϕ	ω	ρ	APC_GRBCycle	APC_GRBCC
50	1	0.15	3	3600	3600
50	1	0.25	3	3600	2385.63
50	1	0.4	3	277.076	268.991
60	1	0.15	3	3600	3600
60	1	0.25	3	3600	3600
60	1	0.4	3	3600	3600
34	0.8	0.15	3	2.69425	2.21986
34	0.8	0.25	3	16.1295	11.7058
34	0.8	0.4	3	11.7809	4.04839
40	0.8	0.15	3	106.22	56.7665
40	0.8	0.25	3	61.3974	27.2067
40	0.8	0.4	3	22.0314	22.2556
44	0.8	0.15	3	668.152	796.757
44	0.8	0.25	3	109.107	114.534
44	0.8	0.4	3	80.1541	26.0788
50	0.8	0.15	3	3600	3600
50	0.8	0.25	3	695.438	472.771
50	0.8	0.4	3	347.674	186.472
60	0.8	0.15	3	3600	3600
60	0.8	0.25	3	3600	3328.78
60	0.8	0.4	3	828.416	527.909
34	0.6	0.15	3	4.63188	3.80315
34	0.6	0.25	3	3.02052	1.6734
34	0.6	0.4	3	0.186155	0.26222
40	0.6	0.15	3	27.516	17.8009
40	0.6	0.25	3	8.26792	4.47872
40	0.6	0.4	3	4.76398	3.41385

Table A.2. CPU Usages of APC_GRBCycle and APC_GRBCC (cont.).

N	ϕ	ω	ρ	APC_GRBCycle	APC_GRBCC
44	0.6	0.15	3	66.0592	61.2573
44	0.6	0.25	3	29.6147	20.8044
44	0.6	0.4	3	15.4169	20.8044
50	0.6	0.15	3	309.723	340.321
50	0.6	0.25	3	95.1526	86.6645
50	0.6	0.4	3	43.2642	19.0479
60	0.6	0.15	3	3600	3600
60	0.6	0.25	3	460.961	329.808
60	0.6	0.4	3	264.759	301.09

Table A.3. CPU Usages of the procedures that are run on 45 test instances.

N	ϕ	ω	ρ	APC_EP	APC_CP	APC_MWCycle	APC_MWClique
17	0.15	1	1	3600	3600	320.225	518.636
17	0.25	1	1	1203.1	3600	316.451	334.179
17	0.4	1	1	44.887	3600	14.369	17.011
20	0.15	1	1	3600	3600	3600	3600
20	0.25	1	1	3600	3600	6.142	1879.865
20	0.4	1	1	268.849	3600	42.518	48.099
22	0.15	1	1	3600	3600	3600	3600
22	0.25	1	1	3600	3600	3600	3600
22	0.4	1	1	939.841	3600	83.121	104.418
25	0.15	1	1	3600	3600	3600	3600
25	0.25	1	1	3600	3600	3600	3600
25	0.4	1	1	3600	3600	183.967	245.189
30	0.15	1	1	3600	3600	3600	3600
30	0.25	1	1	3600	3600	3600	3600
30	0.4	1	1	3600	3600	1330.696	3600

Table A.3. CPU Usages of the procedures that are run on 45 test instances. (cont.)

N	ϕ	ω	ρ	APC_EP	APC_CP	APC_MWCycle	APC_MWClique
17	0.15	0.8	1	2148.841	3600	31.433	30.026
17	0.25	0.8	1	240.258	3600	48.34	47.029
17	0.4	0.8	1	12.933	3600	1.584	1.372
20	0.15	0.8	1	3600	3600	3600	3600
20	0.25	0.8	1	731.836	3600	252.619	291.609
20	0.4	0.8	1	40.442	3600	6.941	10.615
22	0.15	0.8	1	3600	3600	3600	3600
22	0.25	0.8	1	3600	3600	582.871	577.21
22	0.4	0.8	1	350.325	3600	14.616	22.583
25	0.15	0.8	1	3600	3600	3600	3600
25	0.25	0.8	1	3600	3600	2235.988	2995.962
25	0.4	0.8	1	485.2	3600	61.895	88.337
30	0.15	0.8	1	3600	3600	3600	3600
30	0.25	0.8	1	3600	3600	3600	3600
30	0.4	0.8	1	3086.013	3600	297.107	262.168
17	0.15	0.6	1	145.206	3600	53.226	71.611
17	0.25	0.6	1	6.34	3600	0.883	0.746
17	0.4	0.6	1	2.309	3600	0.528	0.612
20	0.15	0.6	1	2354.858	3600	467.237	675.224
20	0.25	0.6	1	41.582	3600	15.43	18.399
20	0.4	0.6	1	3.32	3600	1.249	1.515
22	0.15	0.6	1	3600	3600	2059.401	2722.007
22	0.25	0.6	1	147.768	3600	49.617	43.723
22	0.4	0.6	1	17.9	3600	2.464	2.322
25	0.15	0.6	1	3600	3600	3600	3600
25	0.25	0.6	1	716.759	3600	128.42	169.291
25	0.4	0.6	1	51.082	3600	6.626	7.262

Table A.3. CPU Usages of the procedures that are run on 45 test instances. (cont.)

N	ϕ	ω	ρ	APC_EP	APC_CP	APC_MWCycle	APC_MWClique
30	0.15	0.6	1	3600	3600	3600	3600
30	0.25	0.6	1	3600	3600	3600	1547.915
30	0.4	0.6	1	333.278	3600	20.75	33.847

APPENDIX B: DETAILED TEST RESULTS ON THE NUMBER OF SEPARATED CUTS

The detailed information on the test results regarding the the number of separated cuts of the solution proedures are provided in following tables where OCC, OWC, MCC, and NC correspond to odd cycle, odd wheel, maximal clique and nodal cuts, respectively.

Table B.1. Separated cut number of the procedure APC_GRBCycle.

N	ϕ	ω	ρ	OCC	OWC
34	1	0.15	1	535	1
34	1	0.25	1	543	18
34	1	0.4	1	11	0
40	1	0.15	1	9866	63
40	1	0.25	1	1106	16
40	1	0.4	1	98	1
44	1	0.15	1	130573	630
44	1	0.25	1	3516	79
44	1	0.4	1	142	8
50	1	0.15	1	14106	128
50	1	0.25	1	5866	210
50	1	0.4	1	89	7
60	1	0.15	1	11373	120
60	1	0.25	1	4301	172
60	1	0.4	1	305	10
34	0.8	0.15	1	259	1
34	0.8	0.25	1	216	6
34	0.8	0.4	1	11	0
40	0.8	0.15	1	6819	43

Table B.1. Separated cut number of the procedure APC_GRBCycle. (cont.)

N	ϕ	ω	ρ	OCC	OWC
40	0.8	0.25	1	269	5
40	0.8	0.4	1	3	0
44	0.8	0.15	1	49276	243
44	0.8	0.25	1	328	11
44	0.8	0.4	1	107	4
50	0.8	0.15	1	58243	407
50	0.8	0.25	1	868	31
50	0.8	0.4	1	57	2
60	0.8	0.15	1	9763	137
60	0.8	0.25	1	3277	100
60	0.8	0.4	1	46	2
34	0.6	0.15	1	315	5
34	0.6	0.25	1	14	0
34	0.6	0.4	1	0	0
40	0.6	0.15	1	713	1
40	0.6	0.25	1	22	0
40	0.6	0.4	1	0	0
44	0.6	0.15	1	1312	9
44	0.6	0.25	1	75	1
44	0.6	0.4	1	0	0
50	0.6	0.15	1	3825	25
50	0.6	0.25	1	272	4
50	0.6	0.4	1	2	0
60	0.6	0.15	1	7272	103
60	0.6	0.25	1	448	19
60	0.6	0.4	1	56	4
34	1	0.15	2	266	0

Table B.1. Separated cut number of the procedure APC_GRBCycle. (cont.)

N	ϕ	ω	ρ	OCC	OWC
34	1	0.25	2	640	15
34	1	0.4	2	113	5
40	1	0.15	2	4096	45
40	1	0.25	2	1645	49
40	1	0.4	2	64	1
44	1	0.15	2	11894	148
44	1	0.25	2	3233	85
44	1	0.4	2	152	7
50	1	0.15	2	16356	175
50	1	0.25	2	6217	183
50	1	0.4	2	373	20
60	1	0.15	2	12351	132
60	1	0.25	2	5216	197
60	1	0.4	2	921	41
34	0.8	0.15	2	208	3
34	0.8	0.25	2	365	8
34	0.8	0.4	2	12	1
40	0.8	0.15	2	2146	29
40	0.8	0.25	2	305	10
40	0.8	0.4	2	55	1
44	0.8	0.15	2	4520	44
44	0.8	0.25	2	573	14
44	0.8	0.4	2	145	7
50	0.8	0.15	2	12317	153
50	0.8	0.25	2	1477	38
50	0.8	0.4	2	36	1
60	0.8	0.15	2	10237	105

Table B.1. Separated cut number of the procedure APC_GRBCycle. (cont.)

N	ϕ	ω	ρ	OCC	OWC
60	0.8	0.25	2	3408	121
60	0.8	0.4	2	172	11
34	0.6	0.15	2	220	1
34	0.6	0.25	2	24	2
34	0.6	0.4	2	0	0
40	0.6	0.15	2	722	4
40	0.6	0.25	2	97	4
40	0.6	0.4	2	2	0
44	0.6	0.15	2	1025	8
44	0.6	0.25	2	178	2
44	0.6	0.4	2	1	0
50	0.6	0.15	2	3456	27
50	0.6	0.25	2	155	5
50	0.6	0.4	2	12	0
60	0.6	0.15	2	7760	89
60	0.6	0.25	2	515	16
60	0.6	0.4	2	74	3
34	1	0.15	3	77	0
34	1	0.25	3	673	11
34	1	0.4	3	80	3
40	1	0.15	3	24551	115
40	1	0.25	3	1099	24
40	1	0.4	3	126	5
44	1	0.15	3	174855	771
44	1	0.25	3	9165	235
44	1	0.4	3	153	6
50	1	0.15	3	14920	121

Table B.1. Separated cut number of the procedure APC_GRBCycle. (cont.)

N	ϕ	ω	ρ	OCC	OWC
50	1	0.25	3	5922	172
50	1	0.4	3	303	12
60	1	0.15	3	12711	120
60	1	0.25	3	5645	205
60	1	0.4	3	1000	32
34	0.8	0.15	3	144	0
34	0.8	0.25	3	279	6
34	0.8	0.4	3	23	1
40	0.8	0.15	3	4344	18
40	0.8	0.25	3	496	7
40	0.8	0.4	3	61	2
44	0.8	0.15	3	0	396
44	0.8	0.25	3	354	7
44	0.8	0.4	3	115	2
50	0.8	0.15	3	148461	872
50	0.8	0.25	3	1249	26
50	0.8	0.4	3	100	6
60	0.8	0.15	3	9344	97
60	0.8	0.25	3	2493	83
60	0.8	0.4	3	92	3
34	0.6	0.15	3	262	0
34	0.6	0.25	3	53	0
34	0.6	0.4	3	0	0
40	0.6	0.15	3	934	7
40	0.6	0.25	3	71	3
40	0.6	0.4	3	6	1
44	0.6	0.15	3	1898	12

Table B.1. Separated cut number of the procedure APC_GRBCycle. (cont.)

N	ϕ	ω	ρ	OCC	OWC
44	0.6	0.25	3	224	7
44	0.6	0.4	3	22	0
50	0.6	0.15	3	2749	19
50	0.6	0.25	3	227	7
50	0.6	0.4	3	25	0
60	0.6	0.15	3	59408	430
60	0.6	0.25	3	356	14
60	0.6	0.4	3	200	6

Table B.2. Separated cut number of the procedure APC_GRBClique.

N	ϕ	ω	ρ	MCC	NC
34	1	0.15	1	4	6
34	1	0.25	1	12	1
34	1	0.4	1	9	0
40	1	0.15	1	59	113
40	1	0.25	1	45	0
40	1	0.4	1	75	0
44	1	0.15	1	179	299
44	1	0.25	1	119	1
44	1	0.4	1	76	0
50	1	0.15	1	191	243
50	1	0.25	1	295	1
50	1	0.4	1	71	0
60	1	0.15	1	131	56
60	1	0.25	1	302	0
60	1	0.4	1	245	0
34	0.8	0.15	1	2	5

Table B.2. Separated cut number of the procedure APC_GRBClique. (cont.)

N	ϕ	ω	ρ	MCC	NC
34	0.8	0.25	1	11	1
34	0.8	0.4	1	2	0
40	0.8	0.15	1	91	134
40	0.8	0.25	1	9	0
40	0.8	0.4	1	1	0
44	0.8	0.15	1	102	145
44	0.8	0.25	1	11	0
44	0.8	0.4	1	31	0
50	0.8	0.15	1	161	220
50	0.8	0.25	1	22	0
50	0.8	0.4	1	33	0
60	0.8	0.15	1	102	26
60	0.8	0.25	1	196	0
60	0.8	0.4	1	86	0
34	0.6	0.15	1	2	13
34	0.6	0.25	1	0	0
34	0.6	0.4	1	0	0
40	0.6	0.15	1	3	7
40	0.6	0.25	1	1	0
40	0.6	0.4	1	0	0
44	0.6	0.15	1	1	8
44	0.6	0.25	1	0	0
44	0.6	0.4	1	2	0
50	0.6	0.15	1	11	4
50	0.6	0.25	1	6	0
50	0.6	0.4	1	1	0
60	0.6	0.15	1	68	8

Table B.2. Separated cut number of the procedure APC_GRBClique. (cont.)

N	ϕ	ω	ρ	MCC	NC
60	0.6	0.25	1	22	0
60	0.6	0.4	1	60	0
34	1	0.15	2	0	6
34	1	0.25	2	20	3
34	1	0.4	2	51	0
40	1	0.15	2	70	118
40	1	0.25	2	82	2
40	1	0.4	2	47	0
44	1	0.15	2	150	236
44	1	0.25	2	130	0
44	1	0.4	2	93	0
50	1	0.15	2	284	490
50	1	0.25	2	277	0
50	1	0.4	2	179	0
60	1	0.15	2	133	67
60	1	0.25	2	404	0
60	1	0.4	2	442	0
34	0.8	0.15	2	1	10
34	0.8	0.25	2	10	0
34	0.8	0.4	2	10	0
40	0.8	0.15	2	19	43
40	0.8	0.25	2	8	1
40	0.8	0.4	2	29	0
44	0.8	0.15	2	53	105
44	0.8	0.25	2	29	0
44	0.8	0.4	2	52	0
50	0.8	0.15	2	214	300

Table B.2. Separated cut number of the procedure APC_GRBClique. (cont.)

N	ϕ	ω	ρ	MCC	NC
50	0.8	0.25	2	42	0
50	0.8	0.4	2	33	0
60	0.8	0.15	2	106	47
60	0.8	0.25	2	183	0
60	0.8	0.4	2	199	0
34	0.6	0.15	2	6	2
34	0.6	0.25	2	0	0
34	0.6	0.4	2	0	0
40	0.6	0.15	2	10	0
40	0.6	0.25	2	1	0
40	0.6	0.4	2	2	0
44	0.6	0.15	2	6	6
44	0.6	0.25	2	5	0
44	0.6	0.4	2	1	0
50	0.6	0.15	2	19	8
50	0.6	0.25	2	3	0
50	0.6	0.4	2	9	0
60	0.6	0.15	2	70	11
60	0.6	0.25	2	22	0
60	0.6	0.4	2	99	0
34	1	0.15	3	1	8
34	1	0.25	3	21	0
34	1	0.4	3	34	0
40	1	0.15	3	125	214
40	1	0.25	3	52	0
40	1	0.4	3	111	0
44	1	0.15	3	184	336

Table B.2. Separated cut number of the procedure APC_GRBClique. (cont.)

N	ϕ	ω	ρ	MCC	NC
44	1	0.25	3	92	0
44	1	0.4	3	132	0
50	1	0.15	3	276	428
50	1	0.25	3	334	2
50	1	0.4	3	184	0
60	1	0.15	3	164	118
60	1	0.25	3	399	0
60	1	0.4	3	507	0
34	0.8	0.15	3	2	1
34	0.8	0.25	3	7	0
34	0.8	0.4	3	9	0
40	0.8	0.15	3	19	39
40	0.8	0.25	3	13	0
40	0.8	0.4	3	25	6
44	0.8	0.15	3	138	184
44	0.8	0.25	3	20	0
44	0.8	0.4	3	26	5
50	0.8	0.15	3	179	217
50	0.8	0.25	3	62	0
50	0.8	0.4	3	109	1
60	0.8	0.15	3	135	36
60	0.8	0.25	3	227	0
60	0.8	0.4	3	120	0
34	0.6	0.15	3	2	10
34	0.6	0.25	3	1	1
34	0.6	0.4	3	0	0
40	0.6	0.15	3	18	23

Table B.2. Separated cut number of the procedure APC_GRBClique. (cont.)

N	ϕ	ω	ρ	MCC	NC
40	0.6	0.25	3	1	0
40	0.6	0.4	3	3	0
44	0.6	0.15	3	4	9
44	0.6	0.25	3	6	2
44	0.6	0.4	3	8	0
50	0.6	0.15	3	21	13
50	0.6	0.25	3	6	0
50	0.6	0.4	3	15	5
60	0.6	0.15	3	69	7
60	0.6	0.25	3	27	0
60	0.6	0.4	3	90	0

Table B.3. Separated cut number of the procedure APC_GRBCC.

N	ϕ	ω	ρ	OCC	OWC	MCC	NC
34	1	0.15	1	47	0	2	8
34	1	0.25	1	502	11	25	0
34	1	0.4	1	20	1	6	0
40	1	0.15	1	20	0	65	167
40	1	0.25	1	599	14	40	0
40	1	0.4	1	75	3	44	0
44	1	0.15	1	0	0	272	492
44	1	0.25	1	230	9	83	1
44	1	0.4	1	129	10	0	0
50	1	0.15	1	0	0	264	345
50	1	0.25	1	1	0	314	0
50	1	0.4	1	149	1	196	0
60	1	0.15	1	0	0	149	66

Table B.3. Separated cut number of the procedure APC_GRBCC. (cont.)

N	ϕ	ω	ρ	OCC	OWC	MCC	NC
60	1	0.25	1	0	0	330	0
60	1	0.4	1	5	0	339	0
34	0.8	0.15	1	234	0	3	2
34	0.8	0.25	1	245	0	6	8
34	0.8	0.4	1	9	0	8	0
40	0.8	0.15	1	1857	19	11	7
40	0.8	0.25	1	336	7	10	0
40	0.8	0.4	1	28	0	12	0
44	0.8	0.15	1	4216	57	17	16
44	0.8	0.25	1	686	18	27	0
44	0.8	0.4	1	23	1	5	0
50	0.8	0.15	1	41	2	219	229
50	0.8	0.25	1	634	15	29	0
50	0.8	0.4	1	57	2	26	0
60	0.8	0.15	1	0	0	118	66
60	0.8	0.25	1	71	1	139	0
60	0.8	0.4	1	83	1	93	0
34	0.6	0.15	1	127	0	2	1
34	0.6	0.25	1	21	1	0	0
34	0.6	0.4	1	0	0	0	0
40	0.6	0.15	1	436	4	3	8
40	0.6	0.25	1	28	0	1	0
40	0.6	0.4	1	0	0	0	0
44	0.6	0.15	1	865	5	4	3
44	0.6	0.25	1	45	3	1	0
44	0.6	0.4	1	2	0	2	0
50	0.6	0.15	1	2235	0	17	3

Table B.3. Separated cut number of the procedure APC_GRBCC. (cont.)

N	ϕ	ω	ρ	OCC	OWC	MCC	NC
50	0.6	0.25	1	116	3	2	0
50	0.6	0.4	1	3	0	5	0
60	0.6	0.15	1	5912	70	20	0
60	0.6	0.25	1	427	18	17	0
60	0.6	0.4	1	152	8	151	0
34	1	0.15	2	14	0	1	4
34	1	0.25	2	520	14	24	3
34	1	0.4	2	62	2	38	0
40	1	0.15	2	10	0	69	147
40	1	0.25	2	967	14	58	1
40	1	0.4	2	99	1	51	0
44	1	0.15	2	0	0	150	236
44	1	0.25	2	376	10	119	0
44	1	0.4	2	126	6	65	0
50	1	0.15	2	0	0	313	525
50	1	0.25	2	0	0	277	0
50	1	0.4	2	111	2	190	0
60	1	0.15	2	0	0	133	67
60	1	0.25	2	0	0	374	0
60	1	0.4	2	15	1	445	0
34	0.8	0.15	2	145	0	1	2
34	0.8	0.25	2	320	0	9	5
34	0.8	0.4	2	10	0	10	0
40	0.8	0.15	2	1348	5	11	18
40	0.8	0.25	2	273	11	8	0
40	0.8	0.4	2	27	2	26	0
44	0.8	0.15	2	1589	6	26	16

Table B.3. Separated cut number of the procedure APC_GRBCC. (cont.)

N	ϕ	ω	ρ	OCC	OWC	MCC	NC
44	0.8	0.25	2	557	22	15	0
44	0.8	0.4	2	104	1	58	0
50	0.8	0.15	2	69	2	212	246
50	0.8	0.25	2	714	16	24	0
50	0.8	0.4	2	65	2	47	0
60	0.8	0.15	2	0	0	110	55
60	0.8	0.25	2	18	1	211	0
60	0.8	0.4	2	150	8	162	0
34	0.6	0.15	2	156	1	2	0
34	0.6	0.25	2	17	1	1	0
34	0.6	0.4	2	0	0	0	0
40	0.6	0.15	2	694	6	1	4
40	0.6	0.25	2	68	0	1	0
40	0.6	0.4	2	2	1	2	0
44	0.6	0.15	2	908	3	6	0
44	0.6	0.25	2	202	6	5	0
44	0.6	0.4	2	1	0	0	0
50	0.6	0.15	2	3400	36	6	0
50	0.6	0.25	2	132	6	3	0
50	0.6	0.4	2	8	0	9	0
60	0.6	0.15	2	6217	82	14	2
60	0.6	0.25	2	580	20	20	16
60	0.6	0.4	2	68	4	68	0
34	1	0.15	3	12	0	2	6
34	1	0.25	3	593	12	12	1
34	1	0.4	3	123	6	46	0
40	1	0.15	3	6	0	39	86

Table B.3. Separated cut number of the procedure APC_GRBCC. (cont.)

N	ϕ	ω	ρ	OCC	OWC	MCC	NC
40	1	0.25	3	796	14	42	1
40	1	0.4	3	332	20	172	0
44	1	0.15	3	0	0	184	336
44	1	0.25	3	525	19	95	2
44	1	0.4	3	202	5	138	0
50	1	0.15	3	0	0	275	428
50	1	0.25	3	1	0	288	2
50	1	0.4	3	84	1	162	0
60	1	0.15	3	0	0	172	123
60	1	0.25	3	0	0	393	0
60	1	0.4	3	90	0	552	1
34	0.8	0.15	3	79	0	2	2
34	0.8	0.25	3	298	10	6	1
34	0.8	0.4	3	8	1	5	0
40	0.8	0.15	3	1069	10	4	8
40	0.8	0.25	3	503	5	10	2
40	0.8	0.4	3	51	6	29	2
44	0.8	0.15	3	3168	40	26	36
44	0.8	0.25	3	603	20	22	0
44	0.8	0.4	3	38	2	47	3
50	0.8	0.15	3	210	1	166	138
50	0.8	0.25	3	1145	47	47	0
50	0.8	0.4	3	135	5	107	0
60	0.8	0.15	3	0	0	129	36
60	0.8	0.25	3	245	3	234	0
60	0.8	0.4	3	109	5	126	0
34	0.6	0.15	3	252	0	2	8

Table B.3. Separated cut number of the procedure APC_GRBCC. (cont.)

N	ϕ	ω	ρ	OCC	OWC	MCC	NC
34	0.6	0.25	3	31	1	2	4
34	0.6	0.4	3	0	0	0	0
40	0.6	0.15	3	801	7	8	5
40	0.6	0.25	3	61	1	0	2
40	0.6	0.4	3	5	0	6	0
44	0.6	0.15	3	1099	3	0	3
44	0.6	0.25	3	151	1	8	3
44	0.6	0.4	3	19	0	11	1
50	0.6	0.15	3	2878	22	22	7
50	0.6	0.25	3	296	3	11	0
50	0.6	0.4	3	3	1	13	1
60	0.6	0.15	3	6609	69	25	0
60	0.6	0.25	3	494	21	19	0
60	0.6	0.4	3	53	5	81	1
60	0,6	0,4	3	53	5	81	1