USING LAGRANGEAN RELAXATION FOR SOLVING THE MINIMUM SPANNING TREE PROBLEM WITH CONFLICTS

by

Abdulsamed Kağıt

B.S., Industrial Engineering, Boğaziçi University, 2021

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

2023

ACKNOWLEDGEMENTS

First and foremost, I am deeply grateful to my supervisor, Professor İsmail Kuban Altınel. Without his support and guidance, all this work would have never been accomplished. His contributions to this precious faculty has shaped my understanding of this domain immensely. I enjoyed every moment of his supervision to this thesis.

I would like to express my gratitude to Professor Temel Öncan for his guidance in my research and for being a member of the Thesis Defense Jury. His contributions to my academic studies are invaluable to me.

I would like to thank Professor Zeki Caner Taşkın, both for his teaching, which helped me a lot to broaden my knowledge in operations research, and for being a member of the Thesis Defense Jury. It is a great privilege to have him as a member of this faculty.

I would like to thank Bahadır Pamuk, my colleague with whom I have worked back-to-back in BUFAIM Lab. He has always shared his experience and knowledge with me whenever I need assistance. I am grateful for both his friendship and mentorship.

Furthermore, I would like to express my deepest gratitude and love to my best friends Elif Erdem and Yasin Tufan, for making life bearable for me.

Finally, I would like to thank TUBITAK (Scientific and Technological Research Council of Turkey) for their financial support through BIDEB 2210-A scholarship during my graduate studies.

ABSTRACT

USING LAGRANGEAN RELAXATION FOR SOLVING THE MINIMUM SPANNING TREE PROBLEM WITH CONFLICTS

This thesis studies minimum spanning tree problem with conflicts, which is known to be NP-hard. Given an edge weighted graph and a set of conflicting edge pairs, the goal is to find a minimum cost spanning tree which does not contain any conflicting edge pair.

In this study, a Lagrangean Relaxation scheme is proposed to obtain lower bound on the optimum objective value and Lagrangean dual problem is solved via subgradient algorithm. A Lagrangean heuristic is also devised which is combined with a simple local search in order to obtain upper bounds from infeasible solutions obtained throughout the iterations of the subgradient algorithm. Extensive computational experiments show that the proposed Lagrangean relaxation scheme and the Lagrangean heuristic outperforms the ones proposed in the literature either in terms of time efficiency or bound quality. The Lagrangean relaxation scheme and the Lagrangean heuristic are then embedded in a branch-and-bound algorithm, along with a preprocessing procedure, an infeasibility test procedure and valid inequalities to create an exact solution algorithm. The exact solution algorithm proposed in this study is also compared with the ones in the literature and state of the art commercial solver Gurobi 9.5.2. Positive and negative aspects of using Lagrangean relaxation in a branch-and-bound algorithm are also discussed.

ÖZET

ÇATIŞMA KISITLI ENKÜÇÜK KAPSAR AĞAÇ PROBLEMİNİN ÇÖZÜMÜ İÇİN LAGRANGE GEVŞETMESİNİN KULLANILMASI

Bu tezde NP-zor bir problem olan Çatışma Kısıtlı Enküçük Kapsar Ağaç Problemi ele alınmaktadır. Ağırlıklı bir çizge ve birbirleriyle çatışan kenarlar kümesi verildiğinde amaçlanan çizgedeki tüm düğümleri kapsayan ve birbirleriye çatışan herhangi iki kenarı içermeyen enküçük toplam kenar ağırlığına sahip ağacı bulmaktır.

Bu çalışmada eniyi amaç fonksiyon değerine bir alt sınır elde etmek için Lagrange gevşetmesi yöntemi uygulandı ve Lagrange ikil problemi altgradyan algoritması ile çözüldü. Ayrıca problemin eniyi amaç fonksiyonu değerine bir üst sınır bulabilmek için altgradyan algoritmasının her bir yinelenmesinde elde edilen olursuz çözümlere uygulanmak üzere basit bir yerel arama sezgiseli ile birleştirilmiş bir Lagrange sezgiseli tasarlandı. Bilgisayısal sonuçlar, önerilen Lagrange gevşetmesi ve Lagrange sezgiselinin yazında önerilen yöntemleri ya zaman verimliliği olarak ya da amaç fonksiyonu değeri için bulunan alt ve üst sınırların kalitesi olarak geçtiği gösterildi. Sonrasında önerilen Lagrange gevşetmesi ve Lagrange sezgiseli, önişleme süreci, olursuzluk testi ve geçerli eşitsizlikler ile beraber dal-sınır algoritmasının içerisinde kullanıldı ve bir kesin çözüm algoritması önerildi. Önerilen kesin çözüm algoritması yazındaki diğer kesin çözüm algoritmaları ve en güncel ticari tam sayı programlama çözücülerinden Gurobi 9.5.2 ile başarım açısından deneysel olarak karşılaştırıldı. Dal-sınır algoritması içerisinde Lagrange gevşetmesi kullanmanın olumlu ve olumsuz yönleri tartışıldı ve varılan sonuçlar bilgisayısal deneylerle desteklendi.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

$A(D)$	Arc set of D
$\mathcal{B}$	Set of all bridges in $E(T)$
C	Conflict graph
c_e	Cost (weight) of edge e
D	Directed graph
$d_C(u)$	Degree of node u in graph C
$E(G)$	Edge set of G
G	Graph
$G[X]$	Edge induced subgraph of G if $X \subseteq E(G)$ *or* vertex induced subgraph of G if $X \subseteq V(G)$.
$N_C(u)$	Neighborhood of vertex $u \in V(C)$
$N_D^+(u)$	Out-neighbor vertices of vertex $u \in V(D)$
$N_D^-(u)$	In-neighbor vertices of vertex $u \in V(D)$
OC	Odd cycle
T	Spanning tree *or* spanning forest of G
$V(G)$	Vertex set of G
$\alpha(G)$	Stability number of G
Θ	Boolean parameter determining whether to apply UBH or UBH+
$\kappa(G)$	Set of all maximal cliques in G
Λ	List of candidate edges in UBH whose addition to $E(T)$ does not create a cycle or pairwise conflict in $E(T)$
$\nu(G)$	Matching number of G
$\rho(G)$	Edge cover number of G
σ	Boolean parameter determining whether to apply perturbation function
τ	Number of disconnected components in T

$\tau(G)$	Vertex cover number of G
Φ	Boolean parameter determining whether to apply Local Search heuristic
χ	The maximum number of successive runs of Edge_Removal and Edge_Addition procedures until a feasible solution is found
Ψ	Number of edges to be removed from $E(T)$ during Edge_Removal procedure
Ω	The maximum number of random edge sampling from $E(T)$ during Edge_Removal procedure

LIST OF ACRONYMS/ABBREVIATIONS

BFS	Breadth First Search
B&B	Branch-and-Bound Algorithm
B&C	Branch-and-Cut Algorithm
DFS	Depth First Search
IP	Integer Programming
LB	Lower Bound
LP	Linear Programming
LR	Lagrangean Relaxation
LS	Local Search
MST	Minimum Spanning Tree Problem
MSTC	Minimum Spanning Tree Problem with Conflicts
SCF	Single Commodity Flow
SP	Shortest Path
SS	Stable Set Problem
TSP	Travelling Salesman Problem
UB	Upper Bound
UBH	Upper Bound Heuristic

1. INTRODUCTION

Given an edge weighted graph G and a set of conflicting edge pairs, the minimum spanning tree problem with conflicts (MSTC) consists of finding a conflict-free spanning tree with minimum total edge weight. The MSTC, which is first introduced in [1,2] is an extension of the well-known minimum spanning tree problem (MST) [3]. The MST tries to find a tree in an edge weighted graph which has minimum total cost and spans all of the vertices. The MST can be solved in polynomial time by using greedy algorithms such as Prim's [4] and Kruskal's [5] and has various practical applications in the design of computer and communication networks, wiring connections, transportation networks and so on [6, 7]. Conflicting edge pair restrictions can be imposed to denote forbidden or undesired crossing wire segments, railways, roads or pipes.

A design problem arising in the installation of an international oil pipeline system, where each country must be connected through a pipeline, is presented in [1]. In this case, countries correspond to vertices and potential pipes to be constructed between countries correspond to edges in the graph. The cost of constructing a pipeline between two countries corresponds to the edge weight and the objective of the problem is to find the cheapest pipeline connecting all countries. As an additional constraint due to technical or political reasons, assume each pipeline can be constructed by only a specific firm and some of the firms do not want to cooperate with each other which causes conflict relations between edges. The problem is equivalent to determine conflict free minimum spanning tree in the graph representing potential segments of the pipeline.

An offshore wind farm network design problem which can be modeled by MSTC is introduced in [8]. In this case, there are wind turbines to be connected via uncrossed submarine cables whose cost constitute considerable amount of total capital to be invested. The aim is to find minimum spanning tree over wind turbines that does not contain any overlapping submarine cables at minimum total cost.

Another application area is introduced in [9] where a city map is modeled by a graph in which edges represent streets and conflicting edge pairs denote traffic rules prohibiting to make a turn. In that case a conflict free spanning tree ensures the connectivity of any pair of locations in the city.

Looking from a theoretical perspective MST appears in some solution methods for the Travelling Salesman Problem (TSP) [10]. Hence, an efficient solution procedure for MSTC may be useful in devising a solution procedure for Travelling Salesman Problem with Conflicts (TSPC).

Such practical and theoretical motivations constitute the main motivation for this thesis, which aims to propose efficient algorithms for solving MSTC. Considering the fact that MST is efficiently (in polynomial time) solvable with greedy algorithms, we explore Lagrangean relaxation related methods as both heuristic and exact solution approaches.

2. PRELIMINARIES AND NOTATIONS

This section includes definitions and notations that will be frequently used in the upcoming sections. Since MSTC is closely related to MST and stable set problem (SS) as will be explained, following graph theoretical concepts and notations are essential.

Let $G = (V(G), E(G))$ be a weighted and connected graph where $V(G)$ and $E(G)$ are the sets of vertices and edges, respectively. For each edge $e \in E(G)$ a non-negative weight c_e is assigned. Let $N_G(u)$ denote the neighborhood of vertex $u \in V(G)$. Formally defined, $N_G(u) = \{v \in V(G) : \{u, v\} \in E(G)\}$. Let degree of vertex $u \in V(G)$ be denoted by $d_G(u) = |N_G(u)|$.

Let $D = (V(D), A(D))$ be a directed graph where $V(D)$ is the vertex set and $A(D)$ stands for the arc set. Furthermore, let $N_D^+(u)$ ($N_D^-(u)$) represent out-neighbor (in-neighbor) vertices of vertex u. Formally defined, $N_D^+(u) = \{v \in V(D) : (u, v) \in A(D)\}$ and $N_D^-(u) = \{v \in V(D) : (v, u) \in A(D)\}$.

A *subgraph* of G is a graph whose vertices and edges are subsets of $V(G)$ and $E(G)$, respectively. $G[X]$ denotes edge induced subgraph of G when $X \subseteq E(G)$, meaning that $V(G[X])$ is equivalent to the set of vertices which are incident to at least one edge $e \in X$ and $E(G[X]) \equiv X$. However when $X \subseteq V(G)$, $G[X]$ denotes vertex induced subgraph of G, meaning that $V(G[X]) \equiv X$ and $E(G[X])$ is equivalent to the subset of edges whose endpoints are both in X.

A *stable set* is a set of vertices that are pairwise non-adjacent. The *stability number* of graph G is the maximum number of vertices that are pairwise non-adjacent and denoted by $\alpha(G)$. The *matching number* of graph G is the maximum number of edges that are pairwise not incident to the same vertices and denoted by $\nu(G)$. The *vertex cover number* of graph G is the minimum number of vertices required such that every edge in $E(G)$ is incident to at least one of those vertices and denoted by $\tau(G)$.

The *edge cover number* of graph G is the minimum number of edges required such that every vertex in $V(G)$ is incident to at least one of those edges and denoted by $\rho(G)$.

A *clique* is a set of vertices such that every pair of vertices are adjacent. A *maximal clique* is a clique which cannot be expanded by adding any other vertices and the set of all maximal cliques of G is denoted by $\kappa(G)$.

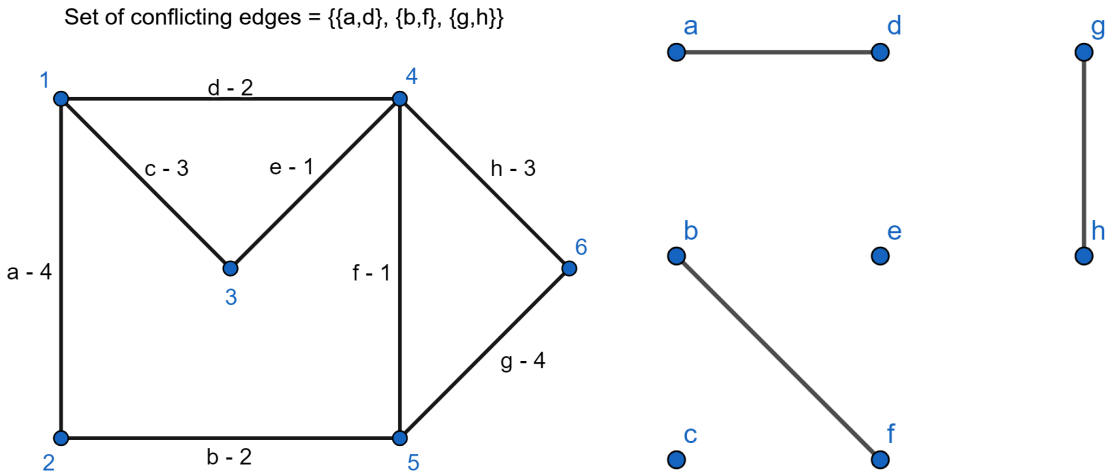
A *path* in G is a sequence of vertices such that all sequential vertices are connected by an edge. A *cycle* is a path which starts and ends at the same vertex. A *simple cycle* is a cycle with no vertex repetition except for the starting and ending vertex. An *odd cycle* is a cycle with odd length (having odd number of edges). An odd cycle in which no two non-sequential vertices are adjacent is called an *odd hole*. A graph G is *acyclic* if it includes no cycles.

A *tree* is an acyclic connected subgraph of G and a *spanning tree* is a tree which covers (spans) every vertex in $V(G)$. A *forest* consists of disjoint set of trees in G. That is to say each component of a forest is a tree. A *spanning forest* is a set of trees which covers (spans) every vertex in $V(G)$. A *bridge* in G refers to an edge which would cause the graph to become disconnected if it is removed. A *cactus* is a connected graph in which any two simple cycles have at most one vertex in common.

Let $T = (V(T), E(T))$ be an acyclic subgraph of G such that $V(T) \equiv V(G)$ and $E(T) \subseteq E(G)$ hold. Note that T is conflict free if $E(T)$ does not include any conflicting edge pair. Both a spanning tree and a spanning forest of G will be represented with $T = (V(T), E(T))$ in the sequel. The weight of T is equal to the sum of the weight of edges in $E(T)$.

For each edge $e \in E(G)$ a set of edges conflicting with it is given. The conflict relations can also be denoted with conflict graph $C = (V(C), E(C))$ where $V(C)$ stands for the vertex set of C such that $V(C) \equiv E(G)$. That is to say, each edge in G corresponds to a vertex in C and vice versa.

$E(C)$ represents the set of edges in C such that each conflicting edge pair $e, f \in E(G)$ corresponds to an edge in $E(C)$. Let $N_C(e)$ stand for the neighborhood of vertex $e \in V(C)$. Namely, $N_C(e)$ is the set of vertices in $V(C)$ which are adjacent to vertex $e \in V(C)$ and $d_C(e) = |N_C(e)|$.



(a) Graph G and set of conflicting edges. (b) Conflict graph C.

Figure 2.1. A MSTC instance.

Figure 2.1 depicts a MSTC instance where labels and costs of edges are indicated next to the relevant edges in Figure 2.1(a). In the following, to provide examples of the aforementioned graph theoretical definitions, we will refer to graph G from the MSTC instance illustrated with Figure 2.1(a). Stability number of G is equal to 3. Vertex set {2,3,6} is an example of stable set. Matching number of G is equal to 3. Egde set {b,c,h} is an example of matching. Vertex cover number of G is equal to 3. Vertex set {1,4,5} is an example of vertex cover. Edge cover number of G is equal to 3. Edge set {b,c,h} is an example of edge cover. Vertex sets {1,3,4} and {4,5,6} are examples of maximal cliques in G. Vertex sequence {1,3,4,1} is an odd hole and vertex sequence {1,2,5,6,4,1} is an odd cycle in G.

3. LITERATURE SURVEY AND CONTRIBUTIONS

3.1. Existing Literature

In contrast with MST, MSTC is strongly NP-hard even if the conflict graph consists only of paths of length two [1]. However, when the conflict graph includes only paths of length one (i.e., disjoint edges), the problem becomes solvable in polynomial time [1]. The decision version of MSTC, which deals with whether there exists a conflict free spanning tree in a given graph, is NP-complete [2]. The hardness results are strengthened further in [11] where it has been proven that the construction of a MST without forbidden transactions is still NP-hard even if the graph is a complete graph. However, when the conflict graph is composed only of disjoint cliques, MSTC is solvable in polynomial time [12]. Besides, when the underlying graph G is a cactus the feasibility problem is also solvable in polynomial time [12].

To the best of our knowledge, in the literature there exists only three studies proposing LR approaches for MSTC [12–14]. Among them, [12] suggests two LR schemes. In the first one, all of the conflicting constraints are dualized and the ordinary MST is solved as a sub-problem. This approach, in theory, yields the same lower bound values as the LP relaxation of the MSTC formulated with exponential number of subtour elimination constraints and is named as LB-MST therein. The second approach, which is named LB-MI, is based on the fact that the MSTC is polynomially solvable when the conflict graph consists of disjoint cliques. The idea is to solve maximum edge clique partitioning problem on the conflict graph with an approximation algorithm and then relaxing the conflicts that do not belong to any clique in the approximate solution. The authors noted that although LB-MI requires excessive CPU times it gives tighter lower bounds than LB-MST provides. However, [14] recently proved that the lower bound values provided by both relaxation schemes are identical. [12] also proposes construction, local search and tabu search heuristic algorithms to provide upper bounds for the MSTC.

In [13] the authors propose to dualize all conflict constraints and employed a hybrid dual ascent and subgradient procedure (HDA) to solve the Lagrangean dual problem. They report that the obtained lower bounds are tighter than that of [12]. To obtain an upper bound, the authors have devised a Lagrangean heuristic, which they call conflict removal, to each spanning tree obtained by solving the Lagrangean subproblem. This heuristic does not take into account the cost of edges while repairing infeasible spanning trees. Hence, although the upper bounds produced are not very tight they are obtained within a relatively short amount of computation time. The relaxation procedure in which the conflict removal heuristic is integrated is called as HDA+.

The third study [14] is based on a Lagrangean decomposition scheme (LD-davol) which consists of two subproblems: MST and minimum weight fixed cardinality stable set problem (KSTAB). The theoretical lower bound quality of the proposed Lagrangean decomposition approach is equivalent to the LR scheme by dualizing exponentially many subtour elimination constraints of MSTC, although the Lagrangean dual problem they propose comprises $|E(G)|$ many Lagrange multipliers. The authors start by solving Lagrangean dual problem using dual-ascent algorithm and then switch to volume algorithm [15] initialized with multipliers obtained by dual-ascent. The cost of such a tight relaxation scheme may require excessive computational time depending on the problem instance due to the fact that one of the subproblems is NP-hard. To the best of our knowledge, this study is the first one that employs a strongly NP-hard problem as a subproblem in a Lagrangean decomposition scheme for MSTC. Since KSTAB is a subproblem solved in the Lagrangean decomposition, authors did not report the lower bound results obtained by Lagrangean decomposition for problem instances whose KSTAB relaxation takes more than 1800 seconds to solve. Their proposed algorithm improves the best known lower bounds for some of the test instances by KSTAB relaxation but for none of the problem instances Lagrangean decomposition enhances the best known lower bounds.

There exist very few meta-heuristics for MSTC. Among them we can mention [16] which proposes a multi-ethnic genetic algorithm where the MSTC is considered with two objective functions. The primary objective function is to find a spanning tree with the minimum number of conflicting edge pairs and the secondary objective function consists of minimizing the total weight of spanning trees without conflicts. Fitness function of the algorithm is designed in such a way that both of these objective functions are concurrently handled. Besides, three local search algorithms are employed within the genetic algorithm in order to improve the fitness of the chromosomes inside the population.

Yet another meta-heuristic for the MSTC is a GRASP approach with adaptive memory developed in [17]. Adaptive memory is used to add a bias to the random selection of elements from restricted candidate list in the constructive phase. For all we know, this heuristic is the best performing one for MSTC in terms of upper bound quality on benchmark instances considered in [12].

As for the exact solution procedures we can refer to the branch-and-cut algorithms [18, 19]. In [18], odd cycle inequalities and maximal clique inequalities derived from conflict graph are used to strengthen LP relaxation bound. Also, a general preprocessing algorithm for MSTC is proposed. In [19], a new set of valid inequalities are introduced and used together with the odd cycle inequalities. In addition, the genetic algorithm proposed in [16] is utilized to obtain upper bounds before starting branch-and-cut so as to prune the nodes of the branch-and-bound search tree earlier.

3.2. Contribution

In this thesis, we propose a LR scheme for the MSTC together with a subgradient algorithm using Polyak's step size rule [20] to solve Lagrangean dual problem. Besides we propose a Lagrangean heuristic and a local search algorithm for obtaining tighter upper bounds. According to extensive computational experiments, we observe that the quality of lower bounds by the novel LR approach is comparable to the ones reported

in [13]. However our approach arises to be more efficient. On the other hand, the proposed Lagrangean heuristic is able to provide tighter upper bounds at the expense of moderate increase in CPU time requirement. For six of the instances addressed in [19] for which an optimal solution is unknown, we have found a better feasible solution than the best known solution. Furthermore, LR heuristic produces a feasible solution for an instance whose feasibility has been unknown.

We also propose a branch-and-bound algoritm (B&B) in which aforementioned LR scheme and upper bound heuristic are embedded together with a preprocessing procedure, an infeasibility test procedure and set of valid inequalities. Performance of the B&B proposed in this study is compared with the existing ones and a state of the art commercial solver Gurobi 9.5.2. Advantages and disadvantages of utilizing Lagrangean relaxation in B&B is discussed and the results are supported with extensive computational experiments.

4. PROBLEM FORMULATION

MSTC consists of finding a conflict free spanning tree with minimum cost on G. Equivalently MSTC deals with determining the spanning tree T having minimum cost in G whose edges $E(T)$ correspond to a stable set of cardinality $|V(G)|-1$ in C since each conflict-free spanning tree in G corresponds to a stable set of cardinality $|V(G)|-1$ in C. Consequently, the MSTC can be considered as a combination of two problems: MST and stable set problem (SS). There exist various formulations for both of them in the literature.

The MST can be solved optimally in polynomial time by greedy algorithms such as Prim's and Kruskal. A MST formulation consists of constraints for the acyclicity and connectedness of a spanning tree. Acyclicity property is guaranteed by subtour elimination constraints and the connectivity is ensured by selecting exactly $|V(G)|-1$ edges from G. In this study, we employ the single commodity flow (SCF) based formulation of the MST [21] due to its effectiveness. A complete catalogue of subtour elimination constraints used in TSP formulations together with performance comparisons are presented in [22]. SCF based formulation is defined by utilizing a directed graph $D=(V(D)\equiv V(G), A(D))$ where $V(D)$ is the vertex set and $A(D)$ stands for the arc set, together with G. For each edge $\{i,j\}\in E(G)$, two arcs (i,j) and (j,i) are defined in $A(D)$. Let x_e denote the binary decision variable representing whether edge e is selected to be in the solution. Let g_{ij} denote the single commodity flow on edge $\{i,j\}$ in the direction from vertex i to vertex j. Then, the SCF based MST formulation is as follows

$$\text{minimize} \quad z = \sum_{e\in E(G)} \mathbf{c}_e x_e \tag{4.1}$$

$$\text{subject to} \quad \sum_{e\in E(G)} x_e = |V(G)|-1 \tag{4.2}$$

$$\sum_{j\in N_D^+(1)} g_{1j} - \sum_{j\in N_D^-(1)} g_{j1} = |V(G)|-1 \tag{4.3}$$

$$\sum_{j\in N_D^-(i)} g_{ji} - \sum_{j\in N_D^+(i)} g_{ij} = 1 \qquad i\in V(G)\setminus\{1\} \tag{4.4}$$

$$g_{ij} \leq (|V(G)| - 1)x_e \qquad e \in E(G), e = \{i, j\} \qquad (4.5)$$

$$g_{ji} \leq (|V(G)| - 1)x_e \qquad e \in E(G), e = \{i, j\} \qquad (4.6)$$

$$x_e \in \{0, 1\}, g_{ji} \geq 0, g_{ij} \geq 0 \qquad e \in E(G), e = \{i, j\}. \qquad (4.7)$$

The objective function (4.1) minimizes the total cost of the selected edges and vertex 1 is an arbitrary vertex in $V(D) \equiv V(G)$. Constraints (4.2) are for the cardinality restrictions which guarantee that $|V(G)| - 1$ edges are selected. Constraints (4.3)-(4.6) are SCF constraints which obviate cycles. Finally, constraints (4.7) are restrictions for the decision variables.

As for SS, we introduce three formulations namely STRONG, WEAK and CLIQUE each of which yields an alternative formulation for MSTC when incorporated with SCF constraints.

The STRONG formulation incorporates $|E(C)|$ many packing constraints

$$x_e + x_f \leq 1 \qquad e \in V(C), f \in N_C(e) \qquad (4.8)$$

to prevent conflicting edge pairs to be in the solution [23]. Therefore, STRONG MSTC formulation consists of (4.1)-(4.7) and (4.8). On the other hand, when we aggregate packing constraints (4.8) for each edge $f \in N_C(e)$ we obtain following $|E(G)|$ many constraints [24]

$$d_C(e)x_e + \sum_{f \in N_C(e)} x_f \leq d_C(e) \qquad e \in V(C). \qquad (4.9)$$

Hence, WEAK MSTC formulation is composed of (4.9) together with (4.1)-(4.7).

Alternatively, SS can be formulated by maximal cliques as follows [23]

$$\sum_{e \in K} x_e \leq 1 \qquad K \in \kappa(C). \qquad (4.10)$$

Here, $\kappa(C)$ denotes the set of all maximal cliques in C. Combining (4.1)-(4.7) with (4.10) yields CLIQUE MSTC formulation. Although the set of feasible solutions for all STRONG, WEAK and CLIQUE MSTC formulations are the same, the LP-relaxation bounds obtained in non-increasing order is CLIQUE MSTC, STRONG MSTC and

WEAK MSTC. WEAK MSTC formulation does not yield much enhancement in terms of efficiency when utilized in an LR scheme. In our LR scheme STRONG MSTC formulation is utilized since listing all maximal cliques takes exponential time [25]. Still, in B&B CLIQUE MSTC formulation is utilized together with odd cycle (OC) inequalities so as to tighten LB obtained by LR and increase the efficiency.

4.1. Relaxation of STRONG Formulation

A LR of the STRONG formulation can be obtained by dualizing the packing constraints (4.8). To this end let λ_{ef} be the non-negative Lagrangian multiplier corresponding to packing constraint for $e \in V(C), f \in N_C(e)$ and $\boldsymbol{\lambda}$ stands for a vector of λ_{ef} values. Then we obtain the following Lagrangean subproblem

$$L(\boldsymbol{\lambda}):$$

$$\text{minimize} \quad z(\boldsymbol{\lambda}) = \sum_{e \in E(G)} \mathbf{c}_e x_e + \sum_{e \in V(C)} \sum_{f \in N_C(e)} \lambda_{ef}(x_e + x_f - 1) \tag{4.11}$$

$$\text{subject to} \quad (4.2) - (4.7). \tag{4.12}$$

Notice that $z(\boldsymbol{\lambda}) \leq z$ holds for any feasible solution $\mathbf{x}$ and $\boldsymbol{\lambda} \geq \mathbf{0}$. Here, L($\boldsymbol{\lambda}$) is a piecewise linear concave function and the Lagrangean dual problem is defined as

$$\text{LD:} \max_{\boldsymbol{\lambda} \geq \mathbf{0}} L(\boldsymbol{\lambda}), \tag{4.13}$$

which can be solved by the subgradient algorithm outlined with Figure 4.1.

The subgradient of λ_{ef} at iteration k of the algorithm is defined as

$$s_{ef}^{(k)} = x_e^*(\boldsymbol{\lambda}^{(k-1)}) + x_f^*(\boldsymbol{\lambda}^{(k-1)}) - 1 \qquad e \in V(C), f \in N_C(e), \tag{4.14}$$

where $x_e^*(\boldsymbol{\lambda}^{(k-1)})$ denotes an optimal value of the variable x_e, obtained by solving $L(\boldsymbol{\lambda}^{(k-1)})$.

Input: A graph $G = (V(G), E(G))$, weights $\mathbf{c}_e$ associated with $e \in E(G)$, and conflict graph $C = (V(C), E(C))$;

Output: Best found Lagrangean dual solution, LB and UB;

begin

Initialize iteration counter $k \leftarrow 1$, division counter $d \leftarrow 0, \pi \leftarrow 2, \lambda_{ef}^{(0)} \leftarrow 0$ for $f \in N_C(e), e \in V(C)$;

$LB \leftarrow 0$ and $UB \leftarrow$ Weight of the spanning tree with largest weight;

$direction_{ef}^{(0)} \leftarrow 0$ for $f \in N_C(e), e \in V(C)$;

repeat

Update the weights

$\mathbf{c}_e^{(k-1)} \leftarrow \mathbf{c}_e + \sum_{f \in N_C(e)} \lambda_{ef}^{(k-1)} \quad e \in V(C)$;

Solve MST with the updated weights $\mathbf{c}^{(k-1)}$ and let $\mathbf{x}^*(\boldsymbol{\lambda}^{(k-1)})$ and $z_1^*(\boldsymbol{\lambda}^{(k-1)})$ denote the optimal solution and its value respectively;

Update

$z^*(\boldsymbol{\lambda}^{(k-1)}) \leftarrow z_1^*(\boldsymbol{\lambda}^{(k-1)}) - \sum_{e \in V(C)} \sum_{f \in N_C(e)} \lambda_{ef}^{(k-1)}$

which is the value of $\mathrm{L}(\boldsymbol{\lambda}^{(k-1)})$;

if $z^*(\boldsymbol{\lambda}^{(k-1)}) >$ LB **then**

LB $\leftarrow z^*(\boldsymbol{\lambda}^{(k-1)})$, $\mathbf{x}_{LB} \leftarrow \mathbf{x}^*(\boldsymbol{\lambda}^{(k-1)})$ and $d \leftarrow 0$;

else

$d \leftarrow d + 1$;

if $d >= 20$ **then**

$d \leftarrow 0$ and $\pi \leftarrow \pi/2$;

end if

end if

$UB \leftarrow min\{UB, DUBH + LS(G, C, \mathbf{x}^*(\boldsymbol{\lambda}^{(k-1)}), \Theta, \chi, \Phi, \sigma)\}$;

Determine the subgradient $\mathbf{s}^{(k)}$ by setting

$s_{ef}^{(k)} \leftarrow x_e^*(\boldsymbol{\lambda}^{(k-1)}) + x_f^*(\boldsymbol{\lambda}^{(k-1)}) - 1 \qquad \forall e \in V(C),\ f \in N_C(e)$;

Compute the step length

$\alpha^{(k)} \leftarrow \pi \frac{1.05UB - z^*(\boldsymbol{\lambda}^{(k-1)})}{\epsilon + \|s^{(k)}\|^2}$;

Update the direction vector by setting

$direction_{ef}^{(k)} \leftarrow 0.25 direction_{ef}^{(k-1)} + s_{ef}^{(k)} \quad \forall e \in V(C), f \in N_C(e)$;

Update the Lagrange multipliers by setting

$\boldsymbol{\lambda}_{ef}^{(k)} \leftarrow \max\{\mathbf{0}, \boldsymbol{\lambda}_{ef}^{(k-1)} + \alpha^{(k)} direction_{ef}^{(k)}\} \quad \forall e \in V(C), f \in N_C(e)$;

$k \longleftarrow k + 1$;

until 1000 iterations, 1800 seconds, $\pi < 0.005$ and $UB - LB < 1$;

end

Figure 4.1. Subgradient algorithm for the solution of MSTC.

4.2. Relaxation of WEAK Formulation

The Lagrangean subproblem for WEAK formulation of MSTC is

$L(\boldsymbol{\lambda})$:

$$\text{minimize} \quad z(\boldsymbol{\lambda}) = \sum_{e \in E(G)} \mathbf{c}_e x_e + \sum_{e \in V(C)} \lambda_e \left(d_C(e) x_e - d_C(e) + \sum_{f \in N_C(e)} x_f \right) \tag{4.15}$$

$$\text{subject to} \quad (4.2) - (4.7). \tag{4.16}$$

The Lagrangean dual problem is defined in a similar way and it can be solved by using Algorithm 4.1 with modifications. The subgradient algorithm for the Lagrangean dual problem of WEAK formulation is obtained by changing line 2, 4, 6, 8, 18, 20 and 21 of Algorithm 4.1. In line 2, Lagrange multipliers are initialized as

$$\lambda_e^{(0)} \leftarrow 0 \quad e \in V(C), \tag{4.17}$$

since there are $|V(C)|$ many constraints to relax in WEAK formulation. Due to the same reason, direction vector is initialized as

$$direction_e^{(0)} \leftarrow 0 \quad e \in V(C), \tag{4.18}$$

in line 4. In line 6, the weights are updated as

$$\mathbf{c}_e^{(k-1)} \leftarrow \mathbf{c}_e + \lambda_e^{(k-1)} d_C(e) + \sum_{f \in N_C(e)} \lambda_f^{(k-1)} \quad e \in V(C). \tag{4.19}$$

In line 8, the value of $\mathrm{L}(\boldsymbol{\lambda}^{(k-1)})$ is calculated as

$$z^*(\boldsymbol{\lambda}^{(k-1)}) \leftarrow z_1^*(\boldsymbol{\lambda}^{(k-1)}) - \sum_{e \in V(C)} \lambda_e^{(k-1)} d_C(e). \tag{4.20}$$

The subgradient of λ_e at iteration k of the algorithm is is defined as

$$s_e^{(k)} = d_C(e) x_e^*(\boldsymbol{\lambda}^{(k-1)}) - d_C(e) + \sum_{f \in N_C(e)} x_f^*(\boldsymbol{\lambda}^{(k-1)}) \qquad e \in V(C). \tag{4.21}$$

Hence, line 18 changes accordingly. In line 20, the direction vector is updated as

$$direction_e^{(k)} \leftarrow 0.25 direction_e^{(k-1)} + s_e^{(k)} \quad e \in V(C). \tag{4.22}$$

Finally, line 21 changes to

$$\boldsymbol{\lambda}_e^{(k)} = \max\{\mathbf{0}, \boldsymbol{\lambda}_e^{(k-1)} + \alpha^{(k)} direction_e^{(k)}\} \quad e \in V(C). \tag{4.23}$$

Note that, the subgradient method employs subgradients of the objective function of Lagrangean subproblem with respect to Lagrange multipliers as stepping direction. When the angle between two subgradients obtained by consecutive iterations is obtuse, to some extend the next iteration counteracts the previous one. Such consecutive subgradients may lead Lagrange multipliers to follow a zigzag pattern that slow down the convergence. To mitigate the effect of such potential situation, the proposed remedy is to calculate the stepping direction by adding a percentage of the previous one to the current subgradient. In our experiments we implemented this strategy which is also named as the deflected subgradient method [26].

5. UPPER BOUND HEURISTIC

During the run of the subgradient algorithm, the solution of the Lagrangean dual problem may yield a conflict-free spanning tree and hence an upper bound for the MSTC. On the other hand, whenever an infeasible spanning tree is obtained, it is possible to perform a Lagrangean heuristic to attain a feasible spanning tree from the infeasible one. Clearly, with the increasing number of feasible solutions output by the subgradient algorithm we have a greater chance to produce tighter upper bounds for the MSTC. To this end, we propose an upper bounding heuristic (UBH) which tries to restore a feasible spanning tree from an infeasible one. Furthermore, we develop a straightforward and efficient local search procedure (LS) to improve upper bounds of the solutions obtained with UBH. The steps of the UBH are outlined below with Figure 5.1. For the sake of clearness, the notation used within the procedures of the UBH is included with Table 5.1.

Given a graph $G = (V(G), E(G))$, a conflict graph $C = (V(C), E(C))$ and a spanning tree $T = (V(T), E(T))$, the UBH tries to output a conflict free spanning tree. UBH includes four procedures Conflict_Clearing, Minimum_Cost_Candidate, Edge_Removal and Edge_Addition which will be presented below.

Conflict_Clearing procedure is formally depicted with Figure 5.2. First, all edges in $E(T)$ conflicting with any bridge of G are deleted. Next, an edge $e \in E(T)$ which conflicts with the maximum number of edges in $E(T)$ is selected. Then e is removed from $E(T)$ and this process is repeated until T becomes conflict free. At the end of this procedure we obtain an acyclic subgraph of G. When a spanning tree is obtained the UBH stops. Otherwise, the acyclic subgraph output by Conflict_Clearing procedure is a forest which consists of at least two components. Let τ denote the number of disjoint components. Then, UBH will try to connect τ disjoint components where each of which is a tree.

Input: $G = (V(G), E(G))$, $C = (V(C), E(C))$, a spanning tree $T = (V(G), E(T))$, parameters Θ, χ, Φ and σ;
Output: A conflict free spanning tree $\overline{T}$, if succeeded;

begin
$\overline{T} = (V(G), E(\overline{T}))$ ← Conflict_Clearing(G,C,T);
τ ← number of disconnected components in $\overline{T}$;
$\Lambda \leftarrow \{e : e \in E(G) \setminus E(\overline{T}), e \cup E(\overline{T})$ is cycle-free and conflict-free$\}$. Sort the edges in Λ by their weights and the number of edges in Λ which are conflicting with them respectively, in ascending order;
for τ -1 many times **do**
  if Λ is empty and $\Theta = False$ **then**
    if $\sigma = True$ **then**
      $\overline{T} \leftarrow$ Perturbation$(G, C, \overline{T}, \Theta, \chi, \Phi)$;
    end if
    if $\overline{T}$ is a spanning tree **then**
      Return $\overline{T}$ and STOP;
    else
      No feasible solution found. STOP;
    end if
  else if Λ is empty and $\Theta = True$ **then**
    for χ many times **do**
      $\overline{T}$ ←Edge_Removal(G,C,$\overline{T}$);
      $\overline{T}$ ← Edge_Addition(G,C,$\overline{T}$);
      if $\overline{T}$ is a spanning tree **then**
        if $\Phi = True$ **then**
          $\overline{T} \leftarrow$ Local Search$(\overline{T}, G, C)$;
        end if
        if $\sigma = True$ **then**
          $\overline{T} \leftarrow$ Perturbation$(G, C, \overline{T}, \Theta, \chi, \Phi)$;
        end if
        Return $\overline{T}$ and STOP;
      end if
    end for
    No feasible solution found. STOP;
  else
    $\overline{T}, \Lambda$ ←Minimum_Cost_Candidate(C, $\overline{T}$, Λ);
  end if
end for
if $\Phi = True$ **then**
  $\overline{T} \leftarrow$ Local Search$(\overline{T}, G, C)$;
end if
if $\sigma = True$ **then**
  $\overline{T} \leftarrow$ Perturbation$(G, C, \overline{T}, \Theta, \chi, \Phi)$;
end if
Return $\overline{T}$ and STOP;
end

Figure 5.1. UBH algorithm for infeasible solution recovery.

Table 5.1. Notation used within the UBH procedures.

Notation	Description
Θ	Boolean parameter determining whether to apply UBH or UBH+.
Λ	List of candidate edges in UBH whose addition to $E(T)$ does not create a cycle or pairwise conflict in $E(T)$.
σ	Boolean parameter determining whether to apply perturbation function.
τ	Number of disconnected components in spanning tree/forest T.
Φ	Boolean parameter determining whether to apply Local Search heuristic.
χ	The maximum number of successive runs of Edge_Removal and Edge_Addition procedures until a feasible solution is found.
Ψ	Number of edges to be removed from $E(T)$ during Edge_Removal procedure. Calculated as $\min\left(\left\lfloor \frac{\lvert V(G)\rvert}{10} \right\rfloor, \left\lfloor \frac{\lvert E(T)\rvert}{2} \right\rfloor\right)$.
Ω	Equals to $\lvert E(T)\rvert$, denoting the maximum number of random edge sampling from $E(T)$ during Edge_Removal procedure.
$\mathcal{B}$	Set of all bridges in $E(T)$.
T	A spanning tree/forest.

Input: $G = (V(G), E(G))$, $C = (V(C), E(C))$, a spanning tree $T = (V(T), E(T))$;
Output: An acyclic subgraph of G;

1 **begin**
2: Delete all edges in $E(T)$ which are in conflict with any of the bridges in G;
3: **repeat**
4: $e \leftarrow \text{argmax}_{e \in E(T)} |\{f : f \in E(T), \{e, f\} \in E(C)\}|$. Break ties arbitrarily, if any;
5: $E(T) \leftarrow E(T) \setminus e$;
6: **until** T is conflict free;
7: Return T;
8 **end**

Figure 5.2. Conflict_Clearing procedure.

To this end, let Λ be a list of candidate edges which consists of edges in $E(G) \setminus E(T)$ whose addition to $E(T)$ does not create a cycle or conflict in T. Among the edges in Λ, we select the one having the minimum weight and insert it to $E(T)$. In case there are more than one such edges, we choose the one conflicting with the minimum number of edges in the candidate edge list Λ. This process is called Minimum_Cost_Candidate depicted with Figure 5.3 and repeated for $\tau - 1$ times in order to connect τ disconnected components. In case a spanning tree is obtained, it is a conflict free spanning tree which yields an upper bound for MSTC.

Input: $C = (V(C), E(C))$, an acyclic subgraph of G denoted by $T = (V(T), E(T))$, Λ;
Output: A forest or a tree T, Candidate edge list Λ;
1**begin**
2: $e \leftarrow$ the first edge in top of Λ;
3: $E(T) \leftarrow E(T) \cup e$;
4: Remove the edges from Λ that are in conflict with e or creates a cycle when added to $E(T)$. Preserve the order of edges in Λ;
5: Return T and Λ;
6**end**

Figure 5.3. Minimum_Cost_Candidate procedure.

During the course of this process, when the candidate edge list Λ is empty, we conclude that there is no candidate edge to be added into $E(T)$. Hence, UBH stops and fails to output a feasible solution. In that case, additional computational effort is required, which is optional and must be decided by means of a boolean parameter Θ which is set to either True or False at the very beginning of UBH. When Θ is set to True, we name the UBH algorithm as UBH+ which requires successive runs of Edge_Removal and Edge_Addition procedures for at most χ number of times until a feasible solution is found where χ is a predefined parameter. Based on preliminary tests, we have chosen to set $\chi = 8$ as a reasonable value.

Edge_Removal procedure is formally described with Figure 5.4. An edge $e \in E(T)$ is randomly selected and if e is not a bridge, e is removed from $E(T)$. When e is a bridge, e is ignored and another edge is randomly chosen. This process is repeated for $\Psi = \min\left(\left\lfloor \frac{|V(G)|}{10} \right\rfloor, \left\lfloor \frac{|E(T)|}{2} \right\rfloor\right)$ times. Randomly selected bridges do not count as an

iteration since they are ignored. Note that the selection of random edges can result in a bridge at most $\Omega = |E(T)|$ times. When the limit is reached, the process is halted.

Input: $G = (V(G), E(G))$, $C = (V(C), E(C))$, an acyclic subgraph of G denoted by $T = (V(T), E(T))$;
Output: A acyclic subgraph of G;

1:**begin**
2: $\Psi \leftarrow \min\left(\left\lfloor \frac{|V(G)|}{10} \right\rfloor, \left\lfloor \frac{|E(T)|}{2} \right\rfloor\right)$;
3: $\Omega \leftarrow |E(T)|$;
4: $\mathcal{B} \leftarrow$ set of all bridges in $E(T)$;
5: **while** $\Psi > 0$ **do**
6: **if** $\Omega = 0$ **then**
7: break;
8: **end if**
9: $e \leftarrow$ a randomly selected edge from $E(T)$;
10: **if** $e \in \mathcal{B}$ **then**
11: $\Omega \leftarrow \Omega - 1$;
12: continue;
13: **end if**
14: $E(T) \leftarrow E(T) \setminus e$;
15: $\Psi \leftarrow \Psi - 1$;
16: **end while**
17: Return T;
18:**end**

Figure 5.4. Edge_Removal procedure.

Immediately after the Edge_Removal procedure, we apply the Edge_Addition procedure which is described with Figure 5.5. Let Λ be a candidate edge list which consists of the edges in $E(G) \setminus E(T)$ whose addition into $E(T)$ does not create any cycle or pairwise conflict in $E(T)$. Among the edges in the candidate edge list Λ, the one conflicting with the minimum number of edges in the candidate edge list is selected and added into $E(T)$. When there are more than one of these edges, the one having minimum weight is chosen. The Edge_Addition procedure is performed until either a spanning tree is obtained or the candidate edge list is empty. When a spanning tree is obtained, it is a feasible solution and gives an upper bound. When the candidate edge list is empty it is concluded that there is no candidate edge to add $E(T)$.

Input: $G = (V(G), E(G))$, $C = (V(C), E(C))$, an acyclic subgraph of G denoted by $T = (V(T), E(T))$;
Output: An acyclic subgraph of G;

begin
$\Lambda \leftarrow \{e : e \in E(G) \setminus E(T), e \cup E(T)$ is cycle-free and conflict-free$\}$. Sort the edges in Λ by the number of edges in Λ which are conflicting with them and their weights respectively, in ascending order.;
repeat
    if Λ is empty **then**
        break;
    end if
    $e \leftarrow$ the first edge in top of Λ;
    $E(T) \leftarrow E(T) \cup e$;
    Remove the edges from Λ that are in conflict with e or creates a cycle when added to $E(T)$. Preserve the order of edges in Λ;
until T is a spanning tree of G;
Return T;
end

Figure 5.5. Edge_Addition procedure.

Further enhancements of the UBH+ algorithm can be achieved by integrating with a local search algorithm (LS) by means of neighborhood search operations once a feasible solution, which is a conflict free spanning tree, is obtained. For that purpose, we consider edge exchange operations which consists of replacing one edge of the conflict free spanning tree with another one in $E(G) \setminus E(T)$ such that the feasibility is not harmed and the total cost is decreased. The edge exchange operation moves to the best neighbor as long as it is possible to find feasible solutions with less total cost. In case no further improved solutions are obtained the neighborhood search process is stopped. Integration of the local search algorithm is also optional and must be decided by means of a boolean parameter Φ which is set to either True or False at the very beginning of UBH+. When Φ is set to True, we name the UBH+ algorithm as UBH+LS which requires the application of the local search algorithm to the feasible solutions found by UBH+.

Moreover, applying a diversification scheme to evenly explore all neighbors of the search space, thus avoiding a narrow focus on a specific area, may further enhance UBH+LS. To this end, within the UBH+LS, we consider perturbation procedures and we employ a tabu list. To diversify the search space, during the first call of the Mini-

mum_Cost_Candidate procedure, the edge e is selected randomly from Λ in line 2. Also, a perturbation function is applied to the tree or forest $\overline{T}$ just before returning from UBH+LS if boolean parameter σ is set to True. Perturbation function is nearly identical to the UBH+LS with a minor difference, it does not start with Conflict_Clearing procedure since $\overline{T}$ is kept conflict free through UBH+LS. Instead, min $\left(\left\lfloor \frac{|V(G)|}{10} \right\rfloor, \left\lfloor \frac{|E(\overline{T})|}{2} \right\rfloor\right)$ many randomly selected edges from $E(\overline{T})$, which are not bridges, are removed. Then UBH+LS is run from the very beginning on this perturbed $\overline{T}$, as if the perturbed $\overline{T}$ is the output of the Conflict_Clearing procedure. If the perturbation function obtains a conflict free spanning tree with a lower cost than that of $\overline{T}$ before perturbation, then the newly obtained spanning tree is replaced with the $\overline{T}$ before perturbation. Removing randomly selected edges enables to focus on various regions of the search space. A tabu list is also employed to further diversify the search process. When an edge is removed from $E(T)$ during the run of Edge_Removal or Conflict_Clearing procedures, it is inserted into the tabu list with probability 0.1. The edges in the tabu list are not considered in the next two candidate egde list Λ creation steps. Note that the size of the tabu list is not limited but since the probability of adding a removed edge to the tabu list is 0.1, the size is kept very small in practice. When candidate edge list Λ is created two times after an edge is added to the tabu list, the edge is removed from the tabu list. Such diversified version of UBH+LS is named as DUBH+LS and this enhanced version is employed in the experiments.

6. INFEASIBILITY DETECTION

Let $G = (V(G), E(G))$ be a given graph for which problem P is defined as the determination of a conflict free minimum weight spanning tree according to the conflict relations between the edges represented by conflict graph $C = (V(C), E(C))$. Let P' be a subproblem of P obtained by partitioning the edges $E(G)$ into three subsets I', X' and F' where $F' = E(G) \setminus (I' \cup X')$. In such partition, I' denotes the set of edges that are forced to be in any feasible solution; X' denotes the set of edges that are forced not to be in any feasible solution; and F' denotes the rest of the edges in $E(G)$. We let G' be the spanning subgraph of G with edge set $I' \cup F' \equiv E(G) \setminus X'$ (i.e., $G' = (V(G'), E(G'))$ with $V(G') \equiv V(G)$ and $E(G') \equiv I' \cup F'$), and $C' \equiv C[F']$ be the subgraph of the conflict graph C induced by $F' \subseteq E(G)$. Besides I' is a stable set of C and there is no edge joining any vertex of C' (element of F') to an element of I'.

If there is no conflict free spanning tree of G that includes the edges of I' and no edges of X', then P' is infeasible. A set of sufficient conditions for the infeasibility of P' are listed in Proposition 6.2. They either determine whether conflict subgraph $C' = C[F']$ has a stable set of size at least $(|V(G)| - 1) - |I'|$ or whether G' has a spanning tree. The elements of any such stable set in $C' = C[F']$ together with the elements of I' form a conflict free edge subset of G of size at least $|V(G)| - 1$. Clearly, if such stable set does not exist, i.e., all stable sets of C that includes I' has size strictly less than $|V(G)| - 1$, then a conflict free spanning tree of G including the edges of I' does not exist either.

Given a graph $G = (V(G), E(G))$ we can define the stability, vertex cover, matching and edge cover numbers, associated with it as

$$\alpha(G) = \max\{|S| : S \text{ is a stable set}\}, \tag{6.1}$$

$$\tau(G) = \min\{|W| : W \text{ is a vertex cover}\}, \tag{6.2}$$

$$\nu(G) = \max\{|M| : M \text{ is a matching}\}, \tag{6.3}$$

$$\rho(G) = \min\{|F| : F \text{ is an edge cover}\}. \tag{6.4}$$

It is possible to show that U is a stable set if and only if $V(G) \setminus U$ is a vertex cover. Besides, $\alpha(G) \leq \rho(G)$ and $\nu(G) \leq \tau(G)$ are obvious inequalities. The following theorem establishes a stronger relation between them, which we use to derive sufficient infeasibility conditions in Proposition 6.2.

Theorem 6.1. *(Gallai's theorem) If $G = (V(G), E(G))$ is a graph without isolated vertices, then*

$$\alpha(G) + \tau(G) = |V(G)| = \nu(G) + \rho(G).$$

Proposition 6.2. *G does not have a conflict free spanning tree including the edges of I' and no edges of X' if one of the following conditions holds:*

i. *G' is disconnected*

ii. *G' has at least two bridges that are not in I' and in conflict,*

iii. $|X'| > |E(G)| - (|V(G)| - 1)$,

iv. $\alpha(C') < (|V(G)| - 1) - |I'|$,

v. $\tau(C') > (|E(G)| - |X'|) - (|V(G)| - 1)$,

vi. $|X'| > |E(G)| - (|V(G)| - 1) + (r - \sum_{j=1}^{r} |K'_j|)$, *where $K'_1, K'_2, \ldots, K'_r$ is a clique partitioning of C'.*

vii. $\rho(C') < (|V(G)| - 1) - |I'|$,

viii. $\nu(G') > (|E(G)| - |X'|) - (|V(G)| - 1)$.

Proof.

i. It is not possible to cover all the vertices of G by a spanning tree including the edges of I'.

ii. Clearly any bridge must be in a spanning tree. If two bridges are in conflict then the subgraph obtained by deleting one of these edges would be disconnected and thus a conflict free spanning tree of G including the edges of I' cannot not exist.

iii. If $|F'| < (|V(G)|-1)-|I'|$, then subgraph G' cannot have at least $(|V(G)|-1)-|I'|$ conflict free edges that form a conflict free spanning tree of G when combined with the edges in I'. The right hand side is the number of edges that have to be

combined with I' in order to obtain a spanning tree of G. Then, the condition directly follows since $|F'| = |E(G)| - |I'| - |X'|$.

iv. Since any conflict free spanning tree of G is a stable set of C, this inequality implies that the stable sets of C' are not large enough to include enough number of conflict free edges whose union with I' form a conflict free spanning tree of G [12].

v. As a consequence of Gallai's theorem $\alpha(C')+\tau(C') = |V(C')|$, from which $\alpha(C') = |F'| - \tau(C')$ follows since $E(G') \equiv V(C') \equiv F'$. Then, $|F'| - \tau(C') < (|V(G)| - 1) - |I'|$ by (iii) and $|F'| = |E(G)| - |I'| - |X'|$ by definition, and the condition is obtained [12].

vi. If $K'_1, K'_2, \ldots, K'_r$ is a clique partitioning of C', then any vertex cover must select at least $|K'_j| - 1$ vertices from K_j for each j. Thus, $\sum_{j=1}^{r} |K'_j| - r$ is a lower bound of a minimum vertex cover of C' and the condition follows from (v).

vii, viii. They follow as consequences of the obvious inequalities $\alpha(G) \leq \rho(G)$ and $\nu(G) \leq \tau(G)$ respectively from (iv) and (v).

□

Notice that for (iv) and (vii) we can use any upper bound on $\alpha(C')$ and $\rho(C')$, and for (v) and (viii) we can use any lower bound on $\tau(C')$ and $\nu(C')$, which can be computed more efficiently instead of the original ones requiring the solutions of NP-hard problems. Clearly, (iv) is satisfied if and only if (v) is satisfied, and (vii) is satisfied if and only if (viii) is satisfied.

The infeasibility detection procedure we devise starts with I', X', and $F' \equiv E(G)\backslash(I' \cup X')$ and enlarges I' and X' with the edges deleted from F'. The enlargement of I' is a result of the fact that every bridge must be present in every feasible solution of P'. Tarjan's Algorithm [27] is used to find all bridges in G'. The enlargement of X' is a result of the constraint that no edge conflicting with any edge in I' can be present in any feasible solution. Also note that enlargement of X' may lead to the emergence of new bridges in G', hence further enlargement of I'. Eventually, it stops either by

finding a feasible solution of P' or detecting its infeasibility if G' does not have any conflict free spanning tree, or without reaching any conclusion with possibly enlarged I' and X', and reduced F'. The formal listing of the feasibility test procedure is given as Figure 6.1 below. G' and C' are respectively the subgraphs of G and C as defined above. Notice that a check whether $I' \cup b$ is acyclic is not necessary since b is a bridge. Again because of this property any such edge must be in an optimal solution, which makes a final feasible I' also optimal for P'.

The infeasibility detection procedure can also be integrated in a B&B scheme to to determine the status of a subproblem $P^{(t)}$. When the procedure stops without reaching any conclusion in parent $P^{(t)}$, edge sets $I^{(t)}$ and $X^{(t)}$ can be enlarged by branching on edges which leads to the reduction of $F^{(t)}$. Then the procedure can be applied to resulting subproblems $P^{(ti)}$ after initializing with $I' \equiv I^{(ti)}$, $X' \equiv X^{(ti)}$ and $F' \equiv F^{(ti)}$ with $F^{(ti)} \equiv E(G) \backslash (I^{(ti)} \cup X^{(ti)})$. In our implementation we consider only the first three conditions of Proposition 6.2, since they are computationally cheap; but we cannot say the same for the remaining ones even when a heuristic is used to compute upper bounds on $\alpha(C^{(ti)})$ and $\rho(C^{(ti)})$, and lower bounds on $\tau(C^{(ti)})$ and $\nu(C^{(ti)})$. Note that when LB on B&B nodes is obtained by LP relaxation, infeasibility of a node is detected by the infeasibility of LP relaxation which is a stronger indicator since it concludes infeasibility by considering constraints from both SS and MST together while the proposed procedure identifies infeasibility only when G' is disconnected.

Input: Graph $G = (V(G), E(G))$ and conflict graph $C = (V(C), E(C))$, the partition (I', X', F') of P', subgraphs $G' = (V(G'), E(G'))$ and $C' = (V(C'), E(C'))$ with $V(G') \equiv V(G)$, $E(G') = I' \cup F'$, and $C' \equiv C[F']$;

Output: An infeasibility decision or a feasible solution if possible or no decision sign;

begin
Generate all bridges in $F' \subseteq E(G')$ and let B be the list of them;
if $B \neq \emptyset$ **then**
  if G' and C' do not pass the tests of Proposition 6.2 **then**
    P' is infeasible return current I', X', F', G', C' and STOP;
  else
    while $B \neq \emptyset$ **do**
      Choose an edge b from B;
      $B \leftarrow B \setminus b$;
      Set $I' \leftarrow I' \cup b$, $X' \leftarrow X' \cup N_{C'}(b)$, $F' \leftarrow F' \setminus (N_{C'}(b) \cup b)$;
      $E(G') \leftarrow E(G') \setminus N_{C'}(b)$;
      $C' \leftarrow C[F']$;
    end while
    if $|I'| = |V(G)| - 1$ **then**
      P' is feasible and I' is the edge set of a conflict free spanning tree, return current I', X', F', G', C' and STOP;
    else
      Go to 2;
    end if
  end if
else
  No decision about P', return current I', X', F', G', C' and STOP;
end if
end

Figure 6.1. Infeasibility test.

7. BRANCH-AND-BOUND ALGORITHM

As an exact solution procedure, we propose a branch-and-bound (B&B) algorithm consisting of the previously introduced Lagrangean relaxation (LR) scheme, infeasibility test and upper bound heuristic (DUBH+LS), combined with a branching rule, maximal clique and odd cycle valid inequalities. An exact separation procedure for odd cycle inequalities is provided in [28] when lower bound (LB) for a node of B&C search tree is obtained by LP relaxation. Since Lagrangean relaxation is utilized in the proposed B&B algorithm to obtain LB, separation procedures of valid inequalities that are designed for LP cannot be used, thus necessitating the development of new separation methods which is discussed in Section 7.2.

7.1. Derivation of the Algorithm

We also attempted to solve the Lagrangean dual problem using the cutting plane method instead of the subgradient algorithm, but the results were incomparable in terms of time efficiency and the LB obtained was almost equal. Hence, subgradient algorithm is used to solve Lagrangean dual problem in *bound* procedure. Subgradient algorithm have been introduced in Section 4.1 together with its pseudocode outlined with Figure 4.1. *Bound* procedure of the B&B algorithm includes Figure 4.1 with the following modifications.

In the LR scheme, we use CLIQUE MSTC formulation with odd cycle inequalities to further tighten LB. The odd cycle and maximal clique inequalities are relaxed, and the resulting subproblem is solved using Prim's algorithm. Therefore, the choice of the formulation for the MST problem does not affect the efficiency of the proposed B&B algorithm. It is worth noting that in the LR scheme, the resulting subproblem is MST for which the convex hull of its feasible region can be represented by exponentially many subtour elimination constraints. Consequently, the LB obtained with the LR scheme is equivalent to the LB obtained with LP relaxation whose feasible region is

defined by the exponentially many subtour elimination constraints and the relaxed odd cycle and maximal clique inequalities combined. The subgradient of maximal clique and odd cycle inequalities are discussed in Section 7.2.

The efficiency of B&B algorithm is extremely affected by the number of subgradient iterations required for each node in B&B search tree. To limit the number of subgradient iterations, parameter π is divided into 2 when division parameter d reached to 5 instead of 20 in every node but the root node.

To obtain tight upper bounds (UB), we apply DUBH+LS to every spanning tree obtained from subgradient iterations of the first node. Hence, to increase the chance of obtaining tight UB, we do not limit the number of subgradient iterations in the root node as described in the previous paragraph. Tight UB allow B&B to prune nodes sooner and increase its efficiency. Every feasible spanning tree obtained through iterations of subgradient algorithm in the child nodes are also used to update UB. Since any UB to the child nodes are also an UB to the original problem, we denote global UB with UB^* in the pseudocode Figure 7.1 and update it whenever a feasible solution having a lower objective value is found.

When LB is obtained via LP relaxation, solvers use the dual simplex method to re-optimize child nodes and decrease the total number of pivot operations significantly. To decrease the total number of subgradient iterations, we adopt a similar strategy and initialize the Lagrange multipliers of a child node with those of its parent. Compared to initializing multipliers by 0, initialization with the multipliers of parent node provide a better starting point and this strategy further increases efficiency.

Lastly, initializing the parameter π to 2 in LR may sometimes lead the subgradient algorithm to diverge in early iterations and delay convergence until π is set to an appropriate value. To mitigate this in child nodes, we keep track of the multipliers that yield the highest LB during the subgradient algorithm. If the LB obtained in an iteration is lower than 20% of the highest LB obtained for the corresponding node

so far, then the multipliers are set to the best recorded multipliers and π is divided by two. Also, for the child nodes while calculating the *step length* in LR, parameter UB is substituted with the cost of spanning tree with maximum weight in G since the corresponding subproblem may have a higher LB than UB to the original problem. These strategies also significantly increase efficiency.

Bound procedure of B&B involves comparing the LB of the node being visited, which is obtained by the given LR scheme, with UB and pruning the node if LB is greater than UB. Each node being visited in the search tree is subject to the *bound* procedure first. When a node in the search tree cannot be pruned by *bound* procedure, *branch* procedure is applied to the node.

Branching rule is an integral component of B&B algorithms and plays a crucial role in the performance. We are inspired by [29, 30] and adapt the branching rule proposed therein to maximize the effect of branching on the conflict graph C. At the very beginning of B&B, the edges are inserted into *edge_list* in decreasing order of their degrees in conflict graph and a B&B search tree is initialized with the original problem P. Then, the subproblems in the search tree are traversed using DFS algorithm. Let the subproblem being visited be denoted by P' in sequel.

In the *branch* procedure, starting from the first edge in *edge_list* denoted as e_1, all edges are selected one by one. Each time an edge e_k is selected, a subproblem P^k is generated by forcing e_k to be in the feasible solution and forcing previously selected edges not to be in the feasible solution along with the edges that are in conflict with e_k. Hence $I^k \leftarrow I' \cup e_k$ and $X^k \leftarrow X' \cup (N_{C'}(e_k) \cup \{e_1, e_2, \ldots, e_{k-1}\})$. The infeasibility test is applied to P^k and if the test does not indicate infeasibility or find a feasible solution, potentially modified by the test, P^k is added into search tree as a child node. Note that if branching procedure creates a cycle in I^k, then P^k is infeasible, hence pruned. If e_k is not in F', then the edge had already been branched on, hence skip to next iteration of procedure *branch*. The pseudocode of the proposed B&B algorithm is given with Figure 7.1.

Note that *branch* procedure and infeasibility test remove some edges from G and force some edges to be in any feasible solution. When the *bound* procedure is applied to a subproblem from the search tree, the MST needs to be solved on the corresponding graph G' of the subproblem, with the restriction that edges in I' must be in the feasible solution. This restricted version of MST can also be solved with Prim's algorithm, with required changes on G'.

For every edge $e_i \in I'$, delete the vertices that are incident to e_i and define a new vertex v_i. Then, for every edge that is incident to only one of the deleted vertices, define an edge which is incident to both v_i and the other vertex to which the deleted edge was incident. In short, combine the vertices that are incident to edges in I' and make them one vertex. Then, solve the MST on the resulting graph and if any newly defined v_i appears in an optimal solution, replace v_i with the original vertex to which the corresponding edge was incident. Finally, add the edge set I' to the obtained solution. This procedure results in an optimal solution of the MST, restricted with edges in I' to be in a feasible solution.

7.2. Valid Inequalities

Valid inequalities are commonly utilized in B&C algorithms to tighten LB of relaxed problems and reduce the size of B&C search tree by pruning nodes earlier. Since MSTC includes constraints from both MST and SS, valid inequalities derived for SS are also valid for MSTC. Among various alternatives, we utilize maximal clique and odd cycle valid inequalities. Typically, there exist exponentially many valid inequalities and identifying all of them may be impractical. When the LB is obtained by LP relaxation, the convention is to solve the relaxed subproblem first and then apply a separation procedure by adding the most violated valid inequality as constraint to the subproblem and re-optimizing simplex algorithm with dual simplex method. The separation procedures vary for each type of valid inequality.

Input: Graph $G = (V(G), E(G))$ and conflict graph $C = (V(C), E(C))$, the partition (I, X, F) of P;

Output: An infeasibility decision or an optimal solution;

begin
$UB^* \leftarrow$ Weight of the spanning tree with largest weight;
$edge\_list \leftarrow E(G)$ ordered in decreasing order of their degrees in conflict graph $d_C(e)$.;
Generate *odd_cycle* and *maximal_clique* inequalities;
Initialize $S \leftarrow \{0, 0, P(I, X, F), odd\_cycle, maximal\_clique\}$;
while $S \neq \emptyset$ and $time \leq 5000$ seconds **do**
    $\{parent\_lb, \boldsymbol{\lambda}, P', (I', X', F'), active\_odd\_cycle, active\_maximal\_clique\} \leftarrow S_{last\_element}$;
    $S \leftarrow S \setminus S_{last\_element}$;
    $\{\boldsymbol{\lambda}, UB^*, LB, I^*\} \leftarrow LR(G', C', \boldsymbol{\lambda}, P'(I', X', F'), active\_odd\_cycle, active\_maximal\_clique)$;
    if $UB^* - LB < 1$ **then**
        continue;
    end if
    for $i = 1$ to $|E(G)|$ **do**
        if $I' \cup edge\_list_i$ is acyclic and $edge\_list_i \in F'$ **then**
            $I_i \leftarrow I' \cup e_i$;
            $X_i \leftarrow X' \cup (N_{C'}(e_i) \cup \{e_1, e_2, \ldots, e_{i-1}\})$;
            $F_i \leftarrow F' \setminus (N_{C'}(e_i) \cup \{e_1, e_2, \ldots, e_i\})$;
            $\{P_i(I_i, X_i, F_i), G', C'\} \leftarrow Infeasibility\_test(G, C, P_i(I_i, X_i, F_i), G', C')$;
            if Infeasibility test stop by infeasiblity decision on P_i **then**
                continue;
            else
                Identify OC and MC inequalities that are still valid for child node P_i. Put them in lists *active_odd_cycle_child* and *active_maximal_clique_child*;
                $S \leftarrow S \cup \{LB, \boldsymbol{\lambda}, P_i(I_i, X_i, F_i), active\_odd\_cycle\_child, active\_maximal\_clique\_child\}$;
            end if
        end if
    end for
end while
if B&B stops by time limit and a feasible solution I^* found **then**
    LB to P can be obtained by minimum $parent\_lb$ value in S;
    Return I^* as the best feasible solution found;
else if B&B stops by time limit and no feasible solution I^* found **then**
    LB to P can be obtained by minimum $parent\_lb$ value in S;
    No feasible solution found;
else if B&B stops by condition $S \equiv \emptyset$ and a feasible solution I^* found **then**
    Return I^* as the optimum solution;
else if B&B stops by condition $S \equiv \emptyset$ and no feasible solution I^* found **then**
    P is infeasible;
end if
end

Figure 7.1. Branch-and-bound algorithm.

When the subgradient algorithm converges, it yields a set of Lagrange multipliers along with a feasible solution to the LR subproblem (MST). Identifying the most violated odd cycle inequality is not possible with this setup since the solution to the Lagrangean dual problem is not exact. Additionally, re-optimizing the subgradient algorithm becomes an issue when an additional constraint is added and relaxed. One option could be initializing the corresponding Lagrange multiplier by setting to zero and keeping the rest of the multipliers intact, then continuing iterations further. However, the stopping conditions of the subgradient algorithm are not well defined, unlike dual simplex method. These limitations motivates us to propose a heuristic approach for odd cycle inequality separation which has been demonstrated to be efficient through experiments.

We separate valid inequalities before starting the B&B algorithm. Therefore, valid inequalities are generated using the conflict graph C' of the problem in the root node of the B&B search tree. We do not separate any other valid inequality throughout the algorithm. As branching procedure generates subproblems by forcing certain edges to be included in or excluded from the feasible solution, thereby expanding sets I' and X', valid inequalities involving the corresponding binary decision variables may become redundant or require modification.

7.2.1. Odd Cycle Valid Inequalities

Odd cycle (OC) inequalities are one of the most commonly used valid inequalities in the SS. As stated in Section 7.1, our LR scheme provides the same LB information as LP relaxation. Therefore, incorporating OC inequalities into the LR scheme and relaxing them have similar effect as adding them as cuts in LP relaxation.

As described in Chapter 2, conflict relations between edges can be represented with a conflict graph C and every feasible solution to MSTC corresponds to a stable set in C. By the definition of a stable set, no two adjacent vertices can be selected. Hence, among vertices of an OC with lenght $2k+1$, at most k vertices can be selected.

Corresponding inequality for any OC can be expressed as

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

where $V(OC)$ denotes the vertices of the odd cycle.

Note that the inequality is scaled for numerical stability issues in the subgradient algorithm, ensuring that the right hand side of the inequality is equal to 1. By incorporating OC inequalities, the resulting formulations become tighter which can significantly increase efficiency of the B&B algorithm depending on the structure of C.

The subgradient of the corresponding Lagrangean multiplier λ_{OC} at iteration k of the algorithm is defined as

$$s_{OC}^{(k)} = \frac{2}{|V(OC)|-1}\sum_{e\in V(OC)} x_e^*(\boldsymbol{\lambda}^{(k-1)}) - 1, \tag{7.2}$$

where $x_e^*(\boldsymbol{\lambda}^{(k-1)})$ denotes the optimal value of the variable x_e, obtained by solving Lagrangean subproblem $L(\boldsymbol{\lambda}^{(k-1)})$ for given Lagrange multipliers.

Through iterations of the subgradient algorithm, λ_{OC} values are updated, thereby the costs of edges in graph G' of corresponding subproblem changes accordingly. For each OC inequality, corresponding increase in the cost of edges that are in $V(OC)$ at iteration k is defined as $\frac{2}{|V(OC)|-1}\lambda_{OC}^{(k-1)}$. To obtain the objective function value of the Lagrangean dual (LB), $\sum_{oc\in OC}\lambda_{oc}$ should be subtracted from the result of Prim's algorithm since the right hand side of the OC inequalities are scaled to 1.

It is worth noting that the number of OCs in a graph can be exponential and identifying all of them is impractical. Additionally, not all OCs contribute to improving LB. Therefore, common practice in B&C algorithms is to add the most violated valid inequality as a constraint and re-optimize the LP relaxation using the dual simplex method until no more significant improvement is possible. Regarding OC inequalities in B&C algorithms, an exact seperation method is described in [28]. Due to the limitations of subgradient algorithm mentioned in Section 7.2, we devise a heuristic to identify OC inequalities that are likely to contribute LB significantly.

The idea behind the proposed OC separation heuristic is as follows. At each iteration of the subgradient algorithm, a MST is obtained with respect to a set of Lagrange multipliers. Lagrange multipliers are updated through iterations and eventually converge to optimum values in practice. Note that there is no proof of convergence for the step size rule used in the proposed subgradient algorithm. In the early iterations of the subgradient algorithm, Lagrange multipliers change rapidly, but as the iterations progress, the multipliers start to stabilize around their optimal values and MSTs obtained as the solution to the subproblem becomes similar, that is having more edges in common. Therefore, we can consider odd cycles in conflict graph C that contain edges emerging frequently in the MSTs obtained throughout the subgradient algorithm as more likely to be violated by the optimum solution of LR subproblem corresponding to near-optimum Lagrangean multiplier values of the Lagrangean dual problem.

Based on the aforementioned arguments, the proposed heuristic consists of the following steps:

- Step 1: Identify the edges of G (vertices of the conflict graph C) that appear in at least 40% of the MSTs obtained through iterations of the subgradient algorithm. Put them in a list called $frequent\_vertices$.
- Step 2: For every possible combination of 3 distinct vertices in $frequent\_vertices$, find the shortest paths (SPs) between each pair of vertices on conflict graph C. Note that since C is an unweighted graph, SPs between vertices can be found using BFS. Concatenate the SPs end to end. If the resulting circuit forms an odd cycle consisting only of vertices from $frequent\_vertices$ and has not been found previously, add the resulting OC cut as a valid inequality.

Even though the number of OCs can be exponential, the heuristic aims to find the ones consisting only of the vertices that appear at least 40% of the MSTs obtained through iterations of the subgradient algorithm. Hence limiting the OC valid inequalities to be added to the LR drastically. The heuristic executes instantly on the problem instances proposed in the literature and helps improving LB significantly depending on

the structure of conflict graph C. The LB obtained at the root node of B&B search tree with and without OC valid inequalities are compared in Section 8.4 to demonstrate the effectiveness of the OC valid inequalities.

It is worth noting that the OC valid inequalities remain valid for child nodes as long as none of their vertices are removed from the conflict graph C through the *branch* procedure or the infeasibility test. As a result, the intact inequalities are inherited by the child nodes.

7.2.2. Maximal Clique Valid Inequalities

Maximal clique (MC) inequalities are facet-defining for the SS problem [23] and can be used to formulate SS problem as discussed in Chapter 4. For a reminder, they are given as

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

$\kappa(C)$ denotes set of maximal cliques in C. To enumerate all maximal cliques in conflict graph C, we use the algorithm proposed in [25]. There may be exponential number of maximal cliques in a graph; hence, all general-purpose maximal clique enumeration algorithms take exponential time [25]. However, conflict graphs of problem instances proposed in the literature are sparse. As a result, the algorithm proposed in [25] executes instantly on the test instances. If that was not the case, a heuristic algorithm restricted to execute within a time limit would be necessary.

The subgradient of the corresponding Lagrangean multiplier λ_K at iteration k of the algorithm is defined as

$$s_K^{(k)} = \sum_{e \in K} x_e^*(\boldsymbol{\lambda}^{(k-1)}) - 1. \tag{7.4}$$

Updates in λ_K values also lead to changes in the cost of edges in graph G' of the corresponding subproblem. For each MC inequality, corresponding increase in the cost of edges that are in the maximal clique K of the conflict graph is defined as $\lambda_K^{(k-1)}$ at

iteration k of the subgradient algorithm. To obtain objective value of the Lagrangean dual (LB), just as in OC valid inequalities, $\sum_{K \in \kappa(C)} \lambda_K$ should be subtracted from the result of Prim's algorithm.

In a subproblem P', if at least 2 edges constituting a MC valid inequality is in the set F', then the corresponding MC inequality is still valid and is inherited by the child nodes. If an edge from a MC valid inequality is added to set I', hence forced to be in any feasible solution, then the rest of the edges constituting the MC are added to set X' and are thus deleted from G' in practice. In this case, the corresponding MC valid inequality is no longer inherited by the child nodes.

8. COMPUTATIONAL EXPERIMENTS

In order to expose the performance of the proposed LR scheme and B&B algorithm, we conducted computational experiments on benchmark instances and compared our results with the ones reported in the literature. Lagrangean relaxation scheme (Figure 4.1) with DUBH+LS heuristic is implemented in C++ using Visual Studio 2022 IDE and executed on Windows 10 platform running on Intel Core i7-6700HQ CPU with 4 cores at 2.60GHz and 16 GB of RAM. B&B algorithm is implemented in C++ using Visual Studio 2019 IDE and executed on Windows Server 2016 Standard platform running on Intel Xeon E5-2650 v4 CPU with 18 cores at 2.20GHz and 288GB of RAM. No graph library is used, graph data structures are implemented using Standard Template Library (STL) containers and all algorithms are implemented single threaded.

Note that the existing algorithms in the literature which we are comparing their performances with the ones proposed in this thesis are performed on miscellaneous platforms. Therefore, the reported CPU times may need to be scaled for direct comparison. However, the differences between CPU performances are not significant to affect the conclusion and hence we do not prefer to scale the CPU times reported in this chapter. For interested reader we may refer to Whetstone benchmarks for relative CPU speeds [31].

According to the benchmarks, the performance of CPUs that are used in the papers proposing LR schemes are as follows. CPU time reported in [13] are obtained with a CPU having 4.2 GFLOPS/core score whereas the CPU with which we conduct LR experiments has 3.3 GFLOPS/core score. On the other hand, GFLOPS/core score is 3.23 for the CPU used in [12].

As for the exact solution algorithms, we choose to compare our results with Gurobi 9.5.2 and the B&C algorithm proposed in [19]. The CPU time reported in [19] is obtained with a CPU having a 4.2 GFLOPS/core score. Unfortunately,

the GFLOPS/core score of the CPU used in our B&B experiments is not listed in Whetstone benchmarks but it is worth noting that 2.0GHz version of the same CPU model has a 2.52 GFLOPS/core score. It is worth mentioning that we do not include the algorithm proposed in [18] for comparison since no CPU time is reported in that paper.

In the remaining of this chapter, first we introduce test instances considered and present the effect of preprocessing scheme on the test instances. Then, performance measures are introduced and results of the computational experiments are reported for both LR scheme and B&B algorithms respectively.

8.1. Test Instances

8.1.1. Existing Test Instances

We have performed experiments on standard instances generated in [12] and [19] which is respectively denoted as ZPK and CCPR classes. ZPK class consists of 50 instances, with various vertex, edge and conflict number combinations, are classified in two groups, namely Type 1 and Type 2 instances, according to the instance generation procedure employed.

On the other hand, CCPR class consists of 180 instances with the number of vertices of 25, 50, 75 and 100. For those instances, the graph density is selected 0.2, 0.3 and 0.4 and conflict graph density is chosen 0.01, 0.04 and 0.07. For each combination of these three parameters, 5 instances are generated considering different seed values. The characteristics of test instances from ZPK and CCPR class are reported in Table A.2 and Table A.3 respectively, in Appendix A. Explanations for both tables are provided in Section 8.2. Both ZPK and CCPR instance classes are downloadable from [32].

8.1.2. Newly Generated Test Instances

Existing test instances have low conflict graph densities. The reason might be that excessive number of conflicts drastically reduces the chances of problem instances being feasible. To compare the performances of the exact solution algorithms on test instances with denser conflict graphs, we generated new test instances denoted as SKT class.

For the SKT class instances, we consider number of vertices n to be 40, graph densities p to be 0.3 and 0.5 and conflict graph densities q to be 0.2, 0.3, 0.4 and 0.5. For each combinations of these three parameters, 3 instances are generated considering different seed values. In total, 1 x 2 x 4 x 3 = 24 test instances are generated.

In the generation algorithm, we ensure that there is at least one feasible solution to the problem instances and that ensured feasible solution is the spanning tree with the maximum weight in the graph. Since the conflict graphs of these test instances are dense, the only feasible solution turns out to be the one that is guaranteed to exist. The aim of generating SKT class instances is to compare the performance of exact solution algorithms in detecting infeasible subproblems, hence the number of test instances is kept limited. The steps of generating test instances are described below:

- Step 1: Set $|E(G)| = \left\lfloor p * \frac{n*(n-1)}{2} \right\rfloor$ and $|E(C)| = \left\lfloor q * \frac{|E(G)|*(|E(G)|-1)}{2} \right\rfloor$. Let $V(G) = \{1, 2, ...n\}$.
- Step 2: From all possible edges of G, take a random sample of size $|E(G)|$ and assign them to $E(G)$. Given that $E(G) \equiv V(C)$, from all possible edges between vertices of C, take a random sample of size $|E(C)|$ and assign them to $E(C)$. Let an edge in $E(G)$ that is incident to vertices i and j is denoted as $\{i, j\}$ where $i < j$. Sort the edges in $E(G)$ in increasing order of i and j respectively.

- Step 3: Define an empty set *selected_edges*. Starting from the first edge, insert edges of $E(G)$ to the *selected_edges* respectively if $G[selected\_edges]$ remains acyclic. If addition of an edge creates a cycle in $G[selected\_edges]$, skip to the next edge.
- Step 4: If G is connected, proceed to Step 5. Otherwise, $|selected\_edges|$ is less than $n-1$. Find the connected components of G. Until G becomes connected, randomly select two connected components and connect them by randomly choosing a vertex from each and insert the edge which is incident to them to both $E(G)$ and *selected_edges*. When G becomes connected, to ensure that $|E(G)|$ remains constant, remove as many randomly selected edges as necessary from $E(G)$ which are not in *selected_edges*. Since *selected_edges* is a tree, G is ensured to be connected. Randomly removed edges are also removed from $V(C)$, possibly reducing the number of conflicts $|E(C)|$.
- Step 5: *selected_edges* is employed as a guaranteed feasible solution. Hence, any conflict between the pairs of edges in *selected_edges* are removed from $E(C)$, possibly reducing the number of conflicts $|E(C)|$.
- Step 6: To ensure possible reduction in $|E(C)|$ is compensated, random pairs of edges, such that not both of them are in *selected_edges*, are selected from $E(G)$ and a conflict is defined between. This step is repeated until $|E(C)|$ returns to its initial value.
- Step 7: Cost of edges in *selected_edges* are set to 100 and the cost of rest of the edges are sampled from $[1, 50]$ randomly.

Out of 24 generated test instances, we only employed 18 of them which Gurobi can solve to optimum within 5000 seconds time limit. Rest of the test instances are discarded since they are not useful in comparing exact solution algorithms for their abilities of detecting infeasible subproblems. The characteristics of the SKT class instances are reported in Table A.4, in Appendix A. Explanations for the table is provided in Section 8.2.

8.2. Preprocessing

Preprocessing consists of fixing the values of some decision variables, whenever possible, through logical implications derived from the problem constraints. For MSTC, [18] propose a problem specific preprocessing algorithm taking advantage of the feasibility conditions. It is an iterative algorithm composed of three sequential phases. Among them we confined ourselves employing only the first two phases since the third phase requires excessive computation time and does not yield significant improvement in the performance. Computational time of the preprocessing is included in the reported CPU times for our algorithms.

In our experiments we sequentially apply the following two phases until no further improvements are possible:

- Phase 1: This phase consists of the search for bridges in G. When there is a bridge which is conflicting with at least one edge, remove all edges conflicting with the bridge since all bridges must be in any feasible solution. This process is repeated until there is no bridge conflicting with other edges.
- Phase 2: In this phase probing operation is applied. That is to say, when there is an edge $e \in E(G)$ conflicting with some other edges whose removal makes G disconnected, then we remove e and go back to Phase 1. Otherwise, the preprocessing procedure terminates.

As it can be noticed, when G includes many bridges conflicting with other edges or cut sets with an edge $e \in E(G)$ conflicting with all of them, the preprocessing algorithm may considerably reduce problem size. From Table A.2 in the appendix, it is possible to observe the effect of the preprocessing algorithm on the instances in the ZPK class. First five columns of the table include the characteristics of test instances. "Type" column incorporates the type of the problem which indicates the instance generation procedure by [12], i.e., Type 1 and Type 2. In our experiments, we address only Type 1 instances because when the preprocessing procedure is applied most of the Type 2

instances become ordinary MST instances since for these instances no conflicting edge pairs left. Namely, for Type 2 instances the feasibility is guaranteed however, for Type 1 instances there is no such guarantee. "Opt" column gives the optimum objective value of the instance. A "*" symbol next to the value refers that it is the best known objective value but the optimal solution is unknown. A "-" sign indicates that the feasibility of the problem is unknown whereas "Infeas" denotes the infeasibility of the problem. Columns $|E(G)|$ and $|E(C)|$ respectively report the number of edges and the number of conflicts whereas the numbers in parentheses in these columns represent the same characteristics after the preprocessing algorithm applied if it resulted in any change. As it can be observed, most of the Type 2 instances become ordinary MST instances after the preprocessing is applied. Since for these instances no conflicting edge pairs left. Hence, we will not consider Type 2 instances in our experiments. On the other hand, for none of these instances in CCPR an ordinary MST instance is obtained after the preprocessing. For Table A.3 and Table A.4 in Appendix A, aforementioned descriptions are also valid for common column names. The only difference is that there is no "Type" column but "Seed" column instead which denotes the random seed used to generate the test instances.

8.3. Experimental Results for Lagrangean Relaxation Scheme

To evaluate the performance of LR schemes, we consider upper bound (UB) and lower bound (LB) values, CPU time in seconds. Besides we also address upper and lower bound relative percentage gaps which are respectively denoted with UB Gap(%) and LB Gap(%) and calculated as $\frac{UB-Opt}{Opt} \times 100\%$ and $\frac{Opt-LB}{Opt} \times 100\%$ where Opt stands for the optimum value.

We should note that if the optimum solution of an instance is unknown, we replace the optimum value with the best known objective value while calculating bound gaps. A "-" sign in the gap value columns indicates that the feasibility of the instance is unknown, hence the bound gap cannot be calculated. Likewise, "Infeas" indicates the instance is infeasible.

Since tighter bounds are reported for LB-MI than LB-MST in [12], we consider the results of LB-MI for benchmarking. The authors employed subgradient algorithm to deal with the LRs and set the maximum number of subgradient iterations to 100. We observed that this setting cause to prematurely stop subgradient algorithm, especially for the LB-MI approach. Even though we implement the same relaxation scheme with modifications, we obtained significantly tighter lower bounds when the subgradient algorithm is allowed to repeat longer than 100 iterations.

[13] remarks that their approach yields tighter lower bounds than the ones reported in [12] which may be due to the following two reasons. First of all, [13] sets the maximum number of iterations to 500 which is considerably higher than 100 iterations. Second, the preprocessing algorithm given in [18] is performed to the instances in ZPK class. Recall that these instances contain many bridges so that they become ordinary MST after preprocessing.

In our experiments, we also consider the results reported in [14]. When comparing ours to them, we consider the lower bounds obtained by LD-davol, if provided, since it yields tighter lower bounds than KSTAB. Otherwise, we consider the lower bounds output by KSTAB relaxation.

8.3.1. Results on ZPK Instance Class

We run our subgradient algorithm (SA) whose pseudocode is given as Figure 4.1 considering the STRONG MSTC formulation on Type 1 instances in ZPK class to obtain lower bounds. Unfortunately, in [13] HDA and HDA+ algorithms were not tested on most of the Type 1 instances but they are performed on all Type 2 instances. Therefore, we do not report the performance of both HDA and HDA+ algorithms on the instances in the ZPK class. The performance measures of LB-MI, LD-davol and SA are included with Table 8.1 together with their averages in the last line. It is evident from the results that LB-MI is the least efficient algorithm with the loosest lower bounds. On the other hand, LD-davol yields the tightest lower bounds and its

superiority with respect to bound quality is remarkable. On average, lower bound gap is 14.45% for LB-MI, 9.54% for SA and 4.34% for LD-davol. As it can be observed SA is the most efficient algorithm. Average CPU time requirements of LB-MI and LD-davol are 5130.83 secs. and 1639.92 secs. respectively, whereas SA terminates within 0.49 secs. on average.

Table 8.1. Lower bounds on the ZPK instance class.

	LB-MI			LD-davol			Subgradient		
ID	LB	LB Gap(%)	CPU(s)	LB	LB Gap(%)	CPU(s)	LB	LB Gap(%)	CPU(s)
1	702.79	0.74	10.45	705.0	0.42	1.2	705.5	0.35	0.06
2	757.82	1.58	10.83	770.0	0.0	1.4	760.31	1.26	0.07
3	807.74	11.91	51.2	900.0	1.85	12.7	866.91	5.46	0.06
4	877.5	33.72	51.64	1251.0	5.51	315.0	962.36	27.31	0.09
8	3991.18	1.23	301.6	4037.0	0.1	5.0	4035.13	0.15	0.1
9	4624.24	18.27	418.61	5371.0	5.07	1402.2	4981.4	11.96	0.1
10	4681.27	Infeas	349.72	6970.0	Infeas	3602.9	5164.46	Infeas	0.17
14	4165.68	2.56	794.08	4275.0	0.0	10.0	4272.61	0.06	0.21
15	4805.4	19.87	753.45	5693.0	5.07	2225.9	5240.4	12.62	0.24
16	4871.27	35.25	781.24	6101.0	18.9	3609.2	5417.05	27.99	0.36
17	4968.99	-	733.25	8506.0	-	1800.0	5543.57	-	0.57
18	5194.67	Infeas	921.39	10506.0	Infeas	1800.0	5565.01	Infeas	0.8
25	11425.8	17.27	4448.64	12993.0	5.92	3603.7	12463.6	9.76	0.3
26	12487.0	Infeas	5456.88	17785.0	Infeas	1800.0	13302.8	Infeas	0.61
31	17922.6	16.56	6640.04	20437.0	4.86	3609.3	19754.1	8.03	0.38
32	19705.7	-	8193.83	27124.0	-	1800.0	20932.1	-	0.57
33	20684.8	-	8429.0	31132.0	-	1800.0	21018.5	-	0.91
34	20226.9	Infeas	9020.98	34648.0	Infeas	1800.0	21023.4	Infeas	1.63
45	40732.7	-	26828.3	51621.0	-	1800.0	44103.3	-	0.9
46	42902.5	-	28421.5	61732.0	-	1800.0	44755.1	-	1.7
Average:	**11326.8**	**14.45**	**5130.83**	**15627.9**	**4.34**	**1639.92**	**12043.4**	**9.54**	**0.49**

8.3.2. Results on CCPR Instance Class

The performances of HDA, LD-davol and SA procedures are reported with Table 8.2 together with their averages at the last line. We present the results as average performance measures of 5 instances having the same vertex, edge and conflict number combinations. Detailed results are provided with Table B.2 in Appendix B.

Table 8.2. Summary of the lower bound results on the CCPR instance class.

	HDA			LD-davol			Subgradient		
$\|V\| - \|E\| - \|E(C)\|$	LB	LB Gap(%)	CPU(s)	LB	LB Gap(%)	CPU(s)	LB	LB Gap(%)	CPU(s)
25-60-18	354.2	0.0	0.04	354.2	0.0	0.12	354.2	0.0	0.01
25-60-71	379.2	0.508	0.05	379.4	0.463	0.44	379.1	0.531	0.01
25-60-124	409.3	2.194	0.07	419.0	0.0	1.72	408.9	2.287	0.02
25-90-41	306.0	0.058	0.05	306.2	0.0	0.18	306.2	0.0	0.01
25-90-161	328.2	0.122	0.08	328.2	0.122	0.52	328.2	0.127	0.02
25-90-281	346.8	2.215	0.1	352.6	0.601	5.8	346.8	2.206	0.04
25-120-72	286.6	0.0	0.06	286.6	0.0	0.3	286.6	0.0	0.01
25-120-286	304.5	0.047	0.12	304.6	0.0	0.68	304.4	0.053	0.02
25-120-500	324.9	2.565	0.16	331.6	0.596	9.28	324.9	2.554	0.05
50-245-299	613.5	0.016	0.34	613.6	0.0	1.08	613.4	0.031	0.06
50-245-1196	659.0	2.704	0.57	671.2	0.928	123.46	659.3	2.67	0.15
50-245-2093	681.7	16.016	0.75	776.2	4.421	3540.92	681.6	16.023	0.21
50-367-672	562.6	0.0	0.85	562.6	0.0	1.82	562.6	0.0	0.05
50-367-2687	603.0	2.463	1.26	613.2	0.805	200.16	603.0	2.462	0.29
50-367-4702	633.9	14.63	1.68	701.4	5.522	3605.22	633.8	14.636	0.45
50-490-1199	541.4	0.0	1.66	541.4	0.0	2.32	541.4	0.001	0.14
50-490-4793	573.1	2.245	2.4	581.8	0.755	250.74	573.2	2.228	0.46
50-490-8387	583.2	12.169	3.08	634.0	4.541	3606.46	583.2	12.177	0.66
75-555-1538	857.8	0.002	2.48	857.8	0.0	4.12	857.8	0.004	0.18
75-555-6150	920.4	9.823	3.52	982.6	3.743	3605.48	920.4	9.82	0.49
75-555-10762	915.1	-	4.43	1078.2	-	1800.0	915.0	-	0.86
75-832-3457	813.6	0.048	6.99	813.6	0.049	8.02	813.7	0.042	0.33
75-832-13828	841.7	6.97	9.04	873.2	3.482	3604.2	841.7	6.964	1.0
75-832-24199	858.3	39.542	12.14	950.4	32.94	1800.0	858.3	39.547	2.09
75-1110-6155	787.1	0.016	14.86	787.2	0.0	11.24	787.1	0.015	0.53
75-1110-24620	807.1	6.104	19.3	835.8	2.764	3605.08	807.1	6.104	2.03
75-1110-43085	810.7	37.463	27.06	871.8	32.75	1800.0	810.7	37.463	3.95
100-990-4896	1112.8	0.09	11.64	1113.4	0.036	19.1	1112.9	0.078	0.62
100-990-19583	1162.6	21.894	15.07	1264.4	15.048	1800.02	1162.6	21.89	1.8
100-990-34269	1151.4	-	21.74	1292.6	-	1800.0	1151.4	-	2.92
100-1485-11019	1061.8	0.056	34.36	1062.0	0.037	40.28	1061.9	0.05	0.82
100-1485-44075	1090.0	16.07	46.01	1147.6	11.636	1800.0	1090.0	16.067	3.64
100-1485-77131	1092.1	-	60.62	1172.6	-	1800.22	1092.0	-	7.28
100-1980-19593	1028.5	0.033	76.55	1028.8	0.0	52.14	1028.5	0.031	1.72
100-1980-78369	1048.4	12.159	105.26	1091.6	8.545	1800.0	1048.6	12.139	8.45
100-1980-137145	1055.8	-	130.42	1114.0	-	1800.0	1055.9	-	15.99
Average:	**719.6**	**5.66**	**17.08**	**752.7**	**3.32**	**1069.48**	**719.6**	**5.66**	**1.59**

One can observe that the lower bound quality of both HDA and SA are identical with approximately 5.66% average lower bound gap. LD-davol again obtains the best lower bound quality with 3.32% average lower bound gap. The tightness of lower bounds obtained by LD-davol stems from the fact that one of the subproblems is strongly NP-hard, which occasionally leads to excessive increase in CPU time requirement. On average, CPU time required by LD-davol is 1069.48 secs. which is far above the average CPU time required by HDA and SA algorithms that are 17.08 and 1.59 secs. respectively. Despite SA outputs identical lower bound quality with the one of HDA, we can state that the former is more efficient than the latter one.

Note that for all three algorithms, among test instances with the same vertex and edge sizes, the ones having more conflicts require more CPU time. [14] sets a 1800 seconds CPU time limit for the solution of KSTAB subproblem. Problem instances for which solving the KSTAB subproblem requires more than 1800 secs., no lower bound results are output by LD-davol; KSTAB relaxation results are reported instead. Also, time limit for LD-davol is 3600 secs. That is why LD-davol does not require less CPU time when executed on problem instances with less conflicts having the same vertex and edge size.

On the other hand as the instance size increases, lower bound gap tends to increase for all three algorithms since the number of dualized constraints also increases and for many of large size instances the optimum solution is substituted by the best known feasible solution which leads to over-estimation of the lower bound gaps.

To evaluate the performance of Lagrangean heuristic DUBH+LS, we consider the results obtained with that of HDA+ on CCPR class. Clearly it would not be fair to expect that a Lagrangean heuristic outperforms a tailor-made and fine-tuned meta-heuristic algorithm. To this end, we compare Lagrangean heuristics with each other.

The performance measures of HDA+ and DUBH+LS are reported with Table B.3 in Appendix B together with their averages in the last line. HDA+ is tested only on test instances with vertex size 25 and 50 in [13]. Therefore, corresponding values are reported with "NA" in the tables, makes averages of performance measures incomparable. We also report the average performance measures of DUBH+LS over test instances on which HDA+ is tested and produces a feasible solution. On such test instances, DUBH+LS has the average upper bound 456.19, average upper bound gap 0.30%, and average CPU time 3.37 secs. On the other hand, HDA+ has the average upper bound 470.31, average upper bound gap 2.64% and average CPU time 1.0 secs. Out of 90 instances on which HDA+ is tested, DUBH+LS found 87 feasible solutions with 62 of them having 0% upper bound gap; whereas HDA+ found 78 feasible solutions with 44 of them having 0% upper bound gap. Results imply that on average DUBH+LS can find tighter upper bounds than HDA+ and it is able to find a feasible solution to more test instances. As for the CPU time, HDA+ is 3.3 times faster on average due to its simple structure.

Note that, DUBH+LS produces a feasible solution in 143 out of 180 instances for which the feasibility of 24 of them are unknown. One of the feasible solutions reached with DUBH+LS belongs to an instance whose feasibility was previously unknown. For 6 of the test instances DUBH+LS improves the best known feasible solution. On average, upper bound gap is 1.22% and CPU time is 24.76 secs. Maximum upper bound gap is 7.67% and maximum CPU time required is 253.32 secs. In 28 out of 180 instances, DUBH+LS requires more than 60 seconds. Finally, we should also remark that DUBH+LS is more efficient on instances for which it cannot find any feasible solution, since local search heuristic is only applied to feasible solutions.

8.4. Experimental Results for B&B Algorithm

To evaluate the performance of the exact solution algorithms, we consider the same set of performance measures as in Section 8.3. The only distinction lies in the representation of infeasibility decisions. If an algorithm decides that an instance is

infeasible, it is denoted by "Infeas" label in UB and LB columns of the tables. A problem instance is solved to optimum if LB and UB values reported in the tables are equal. In the sequel, the branch-and-bound algorithm proposed in Chapter 7 is denoted as B&B, the branch-and-cut algorithm proposed in [19] is denoted as B&C. We also employ Gurobi 9.5.2 to solve problem instances modeled using STRONG MSTC formulation and utilize the results as a benchmark for comparison.

In addition to the performance measures introduced so far, we also utilize *performance profile* approach proposed in [33] to better expose the relative efficiency of the algorithms. The steps for creating a *performance profile* are as follows. For each problem instance p and each algorithm s, *performance ratio* r_{ps} is defined as the ratio of the CPU time taken by algorithm s to solve problem instance p compared to the minimum CPU time required to solve problem instance p among all algorithms. Evidently, the minimum value a *performance ratio* r_{ps} can take is 1, indicating that algorithm s is the most efficient algorithm on problem instance p. If algorithm s could not solve the problem instance in the given time limit, the *performance ratio* r_{ps} is set to a parameter r_M where $r_M \geq r_{ps}$ for any combination of p and s. Also, $r_M = r_{ps}$ if and only if algorithm s does not solve problem instance p.

After calculating *performance ratio* r_{ps} values for all s and p combinations, define the cumulative distribution function p_s for the *performance ratio* as

$$p_s(\tau) = \frac{1}{n_p} |\{p \in P : \log_2 r_{ps} \leq \tau\}|,$$

where P denotes the set of all problem instances and n_p denotes cardinality of set P. The term *performance profile* is used for such cumulative distribution function of performance metrics and illustrated with a graph where x-axis represents τ values and y-axis represent corresponding $p_s(\tau)$ values. The $p_s(\tau)$ values can also be interpreted as the probability that algorithm s requires less than or equal to 2^τ times the best achieved CPU time on a random problem instance.

It is important to emphasize that all of the algorithms compared in this section are limited to execute within 5000 seconds of CPU time. Furthermore, B&C algorithm

is further restricted to execute within a 3 GB memory limit, which leads to termination on some of the problem instances earlier than the time limit of 5000 seconds, without achieving optimality.

In Section 7.2 two classes of valid inequalities are introduced, namely odd cycle (OC) and maximal clique (MC) valid inequalities. To demonstrate the effect of these valid inequalities, LB values for ZPK and CCPR instance classes obtained by LP relaxation of STRONG MSTC formulation, LR with MC valid inequalities and LR with both MC and OC valid inequalities are presented with Table C.2 and Table C.3 in Appendix C.

On the ZPK instance class, the average lower bound gap is 28.38% for LP relaxation of STRONG MSTC formulation, 2.57% for LR with MC valid inequalities and 2.17% for LR with both MC and OC inequalities. On the CCPR instance class, the average lower bound gap is 7.42% for LP relaxation of STRONG MSTC formulation, 4.86% for LR with MC valid inequalities and 4.6% for LR with both MC and OC valid inequalities.

The results suggest that LR schemes provides significantly tighter lower bounds compared to LP relaxation of STRONG MSTC formulation alone. However, it should be noted that B&C algorithms may achieve tighter lower bounds in the root node than the LP relaxation by generating cuts based on the structure of problem instances.

As discussed in Section 8.3, on average, LR of STRONG MSTC yields a 5.66% lower bound gap on CCPR instance class. Consequently, MC valid inequalities lead to 0.8% improvement in the lower bound gap value, while adding OC valid inequalities only results in further 0.26% improvement on CCPR instance class. Both valid inequalities are employed in the B&B algorithm, as the generation of these valid inequalities can be done efficiently.

8.4.1. Results on ZPK Instance Class

The tree aforementioned exact solution algorithms, namely B&B, B&C and Gurobi, are executed on Type 1 instances in ZPK class. The performance measures of the algorithms are presented with Table C.4 in Appendix C together with their averages in the last line. The results reveal a critical weakness of the proposed B&B algorithm: its inability to detect infeasibility. As discussed in Chapter 6, B&B algorithm identifies the infeasiblity of a subproblem in the B&B search tree only if the underlying graph G' is disconnected. On the other hand, B&C and Gurobi detect infeasibility through the infeasibility of the LP relaxation of the underlying subproblem, which is a stronger criterion. In the ZPK class there are 7 instances that have been proven to be infeasible. Gurobi successfully identifies all 7 as infeasible, B&C identifies only 3 of them, while B&B fails to identify any of them within the given time limit.

B&B algorithm prunes a node in the search tree under two conditions. The first condition is when the LB to the visited node is greater than or equal to the UB on the optimal value of the original problem. The second condition when the underlying subproblem is identified as infeasible. Otherwise, *branching* procedure is applied and subproblem is divided into new subproblems. Consequently, inability to detect the infeasibility of a subproblem decreases the efficiency of the algorithm since it may lead to applying *branching* procedure to search nodes representing infeasible subproblems unnecessarily.

On average, lower bound gap is 1.48% for Gurobi, 1.63% for B&C and 3.85% for B&B. Regarding upper bound gaps, Gurobi achieves 0.07%, B&C achieves 0.85%, and B&B achieves 3.6%. Average CPU time requirements is 2219.42 seconds for Gurobi, 2883.12 seconds for B&C and 3535.21 seconds for B&B. Note that infeasible problem instances significantly increases the average CPU time required for B&B algorithm.

Results indicate that B&B is the least efficient algorithm with the largest lower and upper bounds gaps. Gurobi yields tightest lower and upper bounds and is the

most efficient algorithm. No performance profile is provided for the ZPK class since there are only 11 instances for which there exists a known feasible solution. All three algorithms found a feasible solution for those 11 instances.

8.4.2. Results on CCPR Instance Class

Exact solution algorithms are also tested on CCPR instance class. The performance measures of the algorithms are presented with Table C.5 in Appendix C together with their averages in the last line. Out of 180 problem instances, B&B found a feasible solution to 156 problem instances, whereas B&C and Gurobi found a feasible solution to 150 problem instances. Also, Gurobi obtains a proven optimum solution for 114 problem instances whereas B&B and B&C obtain a proven optimum solution for 107 problem instances within the time limit.

The reason B&B found a feasible solution to more problem instances might be the fact that the subgradient algorithm in *bounding* procedure obtains a spanning tree of G' at every iteration contrary to Gurobi or B&C where lower bounds are obtained by LP relaxation, which typically yields fractional solutions. Since Lagrangean subproblem is integer, the only constraints to be satisfied to obtain an upper bound are the conflicting constraints which might be increasing the chances of achieving a feasible solution.

On average, lower bound gap is 2.78% for Gurobi, 3.04% for B&C and 4.12% for B&B. Regarding upper bound gaps, Gurobi achieves 0.05%, B&C achieves 0.4%, and B&B achieves 1.18%. Average CPU time requirements is 1949.68 seconds for Gurobi, 1582.45 seconds for B&C and 2089.29 seconds for B&B. It is worth noting that there are no problem instances proven to be infeasible in CCPR class. Therefore, the average CPU time required for B&B algorithm is not tremendously worse than that of other algorithms, as it is in the ZPK class.

Results again indicate that B&B is the least efficient algorithm with the greatest lower and upper bounds gaps as in Section 8.4.1. Gurobi yields tightest lower and

upper bounds and is the most efficient algorithm. A *performance profile* is provided with Figure8.1 for the CCPR class, based on the results of 133 problem instances where the CPU time required by B&C algorithm is reported to be greater than zero. Unfortunately, 47 problem instances are discarded while calculating the *performance profile*, as the CPU time required by B&C algorithm is reported to be zero due to rounding off, rendering the *performance ratio* r_{ps} indefinite. The *performance profile* depicts the same conclusions regarding comparative performances of the algorithms.

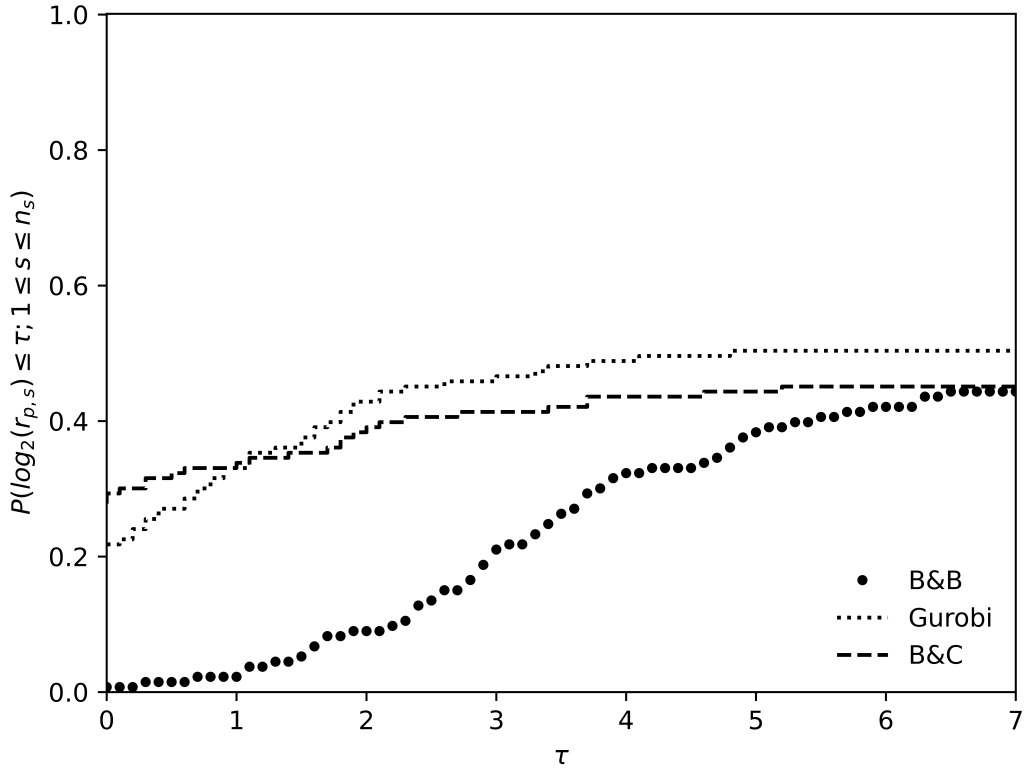


Figure 8.1. Performance comparison of exact solution algorithms.

8.4.3. Results on SKT Instance Class

As stated in Section 8.1.2, the only feasible solution for the SKT instances is the spanning tree with the maximum weight. Therefore, it is not possible to obtain a lower bound greater than the upper bound in any subproblem of these instances. It is only possible to obtain a lower bound equal to the upper bound in a subproblem that holds the feasible solution in its feasible region. Consequently, subproblems that do not hold the feasible solution in their feasible region are never pruned by the bounding procedure but by infeasibility detection.

The SKT instance class is specifically designed to highlight the performance comparison of Gurobi and B&B algorithm in detecting infeasible subproblems. The greater the difference between the infeasible subproblem detection performances of the algorithms, the greater the difference in the efficiency of the algorithms on the SKT instance class. To better expose the difference, the problem instances are selected among which Gurobi can solve to optimum within 5000 seconds time limit. The performance measures of the algorithms are presented with Table 8.3 along with their averages in the last line.

Both algorithms find the optimum solution to all problem instances in the SKT class. However, B&B algorithm solve only 12 problem instances to optimum out of 18 problem instances. Average CPU time requirements is 337.47 seconds for Gurobi and 2476.48 seconds for B&B. B&B algorithm terminating due to time limit hinders the accurate comparison of the relative performances. Therefore, we also report the average CPU time over the 12 test instances which B&B algorithm solved to optimum. On such test instances, average CPU time requirements is 11.94 seconds for Gurobi and 1213.98 seconds for B&B. Results reveals the apparent inefficiency of B&B algorithm in infeasible subproblem detection. In the SKT instance class, lower bound gap values are not of interest in the performance comparison since the cost of the edges in the feasible solution are specifically set to 100, which artificially increases the optimum value of the objective function.

Table 8.3. Performances of the exact solution algorithms on the SKT instance class.

	B&B					Gurobi				
ID	LB	LB Gap(%)	UB	UB Gap(%)	CPU(s)	LB	LB Gap(%)	UB	UB Gap(%)	CPU(s)
231	3900.0	0.0	3900.0	0.0	3400.46	3900.0	0.0	3900.0	0.0	56.35
232	3900.0	0.0	3900.0	0.0	3808.01	3900.0	0.0	3900.0	0.0	48.58
233	3900.0	0.0	3900.0	0.0	3192.69	3900.0	0.0	3900.0	0.0	20.77
234	3900.0	0.0	3900.0	0.0	899.23	3900.0	0.0	3900.0	0.0	5.64
235	3900.0	0.0	3900.0	0.0	746.81	3900.0	0.0	3900.0	0.0	1.48
236	3900.0	0.0	3900.0	0.0	865.86	3900.0	0.0	3900.0	0.0	7.48
237	3900.0	0.0	3900.0	0.0	355.47	3900.0	0.0	3900.0	0.0	0.61
238	3900.0	0.0	3900.0	0.0	360.15	3900.0	0.0	3900.0	0.0	0.47
239	3900.0	0.0	3900.0	0.0	372.04	3900.0	0.0	3900.0	0.0	0.53
240	3900.0	0.0	3900.0	0.0	227.02	3900.0	0.0	3900.0	0.0	0.48
241	3900.0	0.0	3900.0	0.0	176.98	3900.0	0.0	3900.0	0.0	0.48
242	3900.0	0.0	3900.0	0.0	162.98	3900.0	0.0	3900.0	0.0	0.46
243	634.3	83.74	3900.0	0.0	5000.28	3900.0	0.0	3900.0	0.0	2018.6
244	680.3	82.56	3900.0	0.0	5001.04	3900.0	0.0	3900.0	0.0	1970.4
245	715.1	81.66	3900.0	0.0	5002.07	3900.0	0.0	3900.0	0.0	1789.95
246	810.19	79.23	3900.0	0.0	5002.36	3900.0	0.0	3900.0	0.0	48.48
247	826.17	78.82	3900.0	0.0	5001.87	3900.0	0.0	3900.0	0.0	45.64
248	828.11	78.77	3900.0	0.0	5001.29	3900.0	0.0	3900.0	0.0	58.03
Average:	**2849.68**	**26.93**	**3900.0**	**0.0**	**2476.48**	**3900.0**	**0.0**	**3900.0**	**0.0**	**337.47**

9. CONCLUSION

In this thesis, we consider the minimum spanning tree problem with conflicts (MSTC) which can be formulated as a combination of minimum spanning tree (MST) and stable set (SS) problems. We introduce 3 alternative formulation to the problem, namely WEAK, STRONG and CLIQUE formulations and discussed their bound qualities. Then, we propose a Lagrangean relaxation (LR) scheme utilizing STRONG formulation and a Lagrangean Heuristic to obtain upper bounds. The new algorithms outperform the ones in the literature in terms of either efficiency or bound quality. The results have motivated us to integrate these algorithms into a B&B algorithm as an exact solution method.

While integrating the LR scheme into B&B, we have made some modifications and enhancements to increase its efficiency and tighten the LB. We switch the problem formulation from STRONG to CLIQUE and proposed a heuristic algorithm to separate odd cycle (OC) valid inequalities. The LR scheme we propose, in theory, yields the same LB as the LP relaxation as the CLIQUE MSTC formulated with exponential number of subtour elimination constraints and enhanced with the OC valid inequalities we have separated with the heuristic proposed. The advantage of utilizing LR in *bounding* procedure is that the subtour elimination constraints are implicitly separated as the LR subproblem can be solved by greedy algorithms. We also propose an infeasibility test whose function is to reduce problem size whenever possible by exploiting the bridges in the conflict graph. We propose a branching rule so as to maximize the effect of infeasibility test on the subproblems generated by the *branching* procedure.

We compare the performance of the proposed B&B algorithm with that of the branch and cut algorithm (B&C) in the literature and Gurobi. Results indicate that the proposed B&B algorithm is weak in detecting infeasible subproblems as the algorithm decides a subproblem to be infeasible only if the underlying graph is disconnected, which is a weak condition. This situation reveals an important disadvantage of uti-

lizing a LR scheme in a B&B algorithm: relaxing too many constraints hinders the detection of infeasibility of subproblems which leads to inefficiency. Unless there exist an infeasbility test as strong as LP relaxation, substituting LP relaxation with a LR scheme is subject to serious doubt.

While finding maximal cliques, we employ an exact algorithm. Although the algorithm executes in a short period of time for the problem instances considered in this thesis, it may not be the case in denser conflict graphs, necessitating the use of a heuristic or setting a time limit on the algorithm's execution. As further research, developing a stronger and efficient infeasibility test may increase the efficiency of the proposed algorithm drastically.

Despite its apparent inefficiency, the proposed B&B algorithm obtains a feasible solution of 6 problem instances that no other algorithms managed to find. The reason might be the integrality of the optimal solutions of the Lagrangean subproblem. LP relaxation typically produces fractional solutions. On the other hand, subgradient algorithm obtains a spanning tree at each iteration and whenever the obtained spanning tree is conflict free, it is a feasible solution which might increase the chances of obtaining a feasible solution.

REFERENCES

1. 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, Berlin, Heidelberg, 2009.

2. Darmann, A., U. Pferschy, J. Schauer and G. J. Woeginger, "Paths, Trees and Matchings Under Disjunctive Constraints", *Discrete Applied Mathematics*, Vol. 159, No. 16, pp. 1726–1735, 2011.

3. Ahuja, R., J. Magnanti and J. Orlin (Editors), *Network flows: Theory, Algorithms and Applications*, Prentice-Hall, New Jersey, 1993.

4. Prim, R. C., "Shortest Connection Networks and Some Generalizations", *The Bell System Technical Journal*, Vol. 36, No. 6, pp. 1389–1401, 1957.

5. Kruskal, J. B., "On The Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem", *Proceedings of the American Mathematical Society*, Vol. 7, No. 1, pp. 48–50, 1956.

6. Graham, R. and P. Hell, "On The History of The Minimum Spanning Tree Problem", *Annals of the History of Computing*, Vol. 7, No. 1, pp. 43–57, 1985.

7. Jin, R., P. Hou, G. Yang, Y. Qi, C. Chen and Z. Chen, "Cable Routing Optimization for Offshore Wind Power Plants via Wind Scenarios Considering Power Loss Cost Model", *Applied Energy*, Vol. 254, p. 113719, 2019.

8. Klein, A., D. Haugland, J. Bauer and M. Mommer, "An Integer Programming Model for Branching Cable Layouts in Offshore Wind Farms", *Advances in Intelligent Systems and Computing*, Vol. 359, pp. 27–36, 2015.

9. Chan, T.-H. H., L. C. Lau and L. Trevisan (Editors), *Theory and Applications*

of Models of Computation, Lecture Notes in Computer Science, Springer, Berlin, 2013.

10. Held, M. and R. M. Karp, "The Traveling-Salesman Problem and Minimum Spanning Trees", *Operations Research*, Vol. 18, No. 6, pp. 1138–1162, 1970.

11. Kanté, M. M., C. Laforest and B. Momège, "Trees in Graphs with Conflict Edges or Forbidden Transitions", T.-H. H. Chan, L. C. Lau and L. Trevisan (Editors), *Theory and Applications of Models of Computation*, pp. 343–354, Springer, Berlin, Heidelberg, 2013.

12. Zhang, R., S. N. Kabadi and A. P. Punnen, "The Minimum Spanning Tree Problem with Conflict Constraints and Its Variations", *Discrete Optimization*, Vol. 8, No. 2, pp. 191–205, 2011.

13. 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.

14. Samer, P. and D. Haugland, "Polyhedral Results and Stronger Lagrangean Bounds for Stable Spanning Trees", *Optimization Letters*, Vol. 17, No. 6, pp. 1317–1335, 2022.

15. Barahona, F. and R. Anbil, "The Volume Algorithm: Producing Primal Solutions with a Subgradient Method", *Mathematical Programming*, Vol. 87, No. 3, pp. 385–399, 2000.

16. Carrabs, F., C. Cerrone and R. Pentangelo, "A Multiethnic Genetic Approach for The Minimum Conflict Weighted Spanning Tree Problem", *Networks*, Vol. 74, No. 2, pp. 134–147, 2019.

17. da Silva Barros, B. J., R. G. S. Pinheiro, U. S. Souza and L. S. Ochi, "Using Adaptive Memory in GRASP to Find Minimum Conflict-free Spanning Trees",

Soft Computing, Vol. 27, No. 8, pp. 4699–4712, 2022.

18. 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.

19. 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, No. 1, pp. 65–78, 2021.

20. Fisher, M. L., "The Lagrangian Relaxation Method for Solving Integer Programming Problems", *Management Science*, Vol. 50, pp. 1861–1871, 2004.

21. Magnanti, T. L. and L. A. Wolsey, "Chapter 9 Optimal Trees", *Handbooks in Operations Research and Management Science*, Vol. 7, pp. 503–615, Elsevier, Amsterdam, 1995.

22. Öncan, T., İ. K. Altınel and G. Laporte, "A Comparative Analysis of Several Asymmetric Traveling Salesman Problem Formulations", *Computers & Operations Research*, Vol. 36, No. 3, pp. 637–654, 2009.

23. Padberg, M. W., "On The Facial Structure of Set Packing Polyhedra", *Mathematical Programming*, Vol. 5, No. 1, pp. 199–215, 1973.

24. Della Croce, F. and R. Tadei, "A multi-KP Modeling for The Maximum-Clique Problem", *European Journal of Operational Research*, Vol. 73, No. 3, pp. 555–561, 1994.

25. Eppstein, D., M. Löffler and D. Strash, "Listing All Maximal Cliques in Sparse Graphs in Near-Optimal Time", O. Cheong, K.-Y. Chwa and K. Park (Editors), *Algorithms and Computation*, pp. 403–414, Springer, Berlin, Heidelberg, 2010.

26. Guta, B., *Subgradient Optimization Methods in Integer Programming with an Ap-*

plication to a Radiation Therapy Problem, Ph.D. Thesis, Technische Universität Kaiserslautern, 2003.

27. Tarjan, R., "A Note on Finding the Bridges of a Graph", *Information Processing Letters*, Vol. 2, No. 6, pp. 160–161, 1974.

28. Rebennack, S., G. Reinelt and P. M. Pardalos, "A Tutorial on Branch and Cut Algorithms for The Maximum Stable Set Problem", *International Transactions in Operational Research*, Vol. 19, No. 1-2, pp. 161–199, 2012.

29. Balas, E. and J. Xue, "Minimum Weighted Coloring of Triangulated Graphs, with Application to Maximum Weight Vertex Packing and Clique Finding in Arbitrary Graphs", *SIAM Journal on Computing*, Vol. 20, pp. 209–221, 1991.

30. Balas, E. and C. S. Yu, "Finding a Maximum Clique in an Arbitrary Graph", *SIAM Journal on Computing*, Vol. 15, No. 4, pp. 1054–1068, 1986.

31. "Whetstone Benchmarks", `http://setiathome.berkeley.edu/cpu_list.php`, accessed on August 18, 2023.

32. "CCPR Dataset", `http://www.dipmat2.unisa.it/people/carrabs/www/`, accessed on August 18, 2023.

33. 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: PROBLEM INSTANCES

Table A.1. Column expressions.

Column	Description
ID	Unique ID of the problem instance.
$\|V(G)\|$	Number of vertices in G.
$\|E(G)\|$	Number of edges in G. The numbers in parentheses in this column represent the same characteristic after the preprocessing algorithm applied if it resulted in any change.
$\|E(C)\|$	Number of edges in C. Equivalent to number of conflicts in the problem instance. The numbers in parentheses in this column represent the same characteristic after the preprocessing algorithm applied if it resulted in any change.
Type	Type of the problem which indicates the instance generation procedure.
Seed	Random seed used to generate the test instance.
Opt	Optimum objective value of the test instance. A "*" symbol next to the value refers that it is the best known objective value but the optimal solution is unknown. A "-" sign indicates that the feasibility of the problem is unknown whereas "Infeas" denotes the infeasibility of the problem.

Table A.2. Instances in the ZPK class and the effect of preprocessing.

ID	$\|V(G)\|$	$\|E(G)\|$	$\|E(C)\|$	Type	Opt
1	50	200	199	1	708
2	50	200	398	1	770
3	50	200	597	1	917
4	50	200	995	1	1324
5	50	200(197)	3903(3754)	2	1636
6	50	200(189)	4877(4165)	2	2043
7	50	200(184)	5864(4706)	2	2338
8	100	300	448	1	4041
9	100	300(299)	897(891)	1	5658
10	100	300(299)	1344(1284)	1	Infeas
11	100	300(101)	8609(0)	2	7434
12	100	300(100)	10686(0)	2	7968
13	100	300(100)	12761(0)	2	8166
14	100	500	1247	1	4275
15	100	500	2495	1	5997
16	100	500	3741	1	7523*
17	100	500	6237	1	-
18	100	500	12474	1	Infeas
19	100	500(495)	24740(24060)	2	12652
20	100	500(485)	30886(28573)	2	11232
21	100	500(461)	36827(29603)	2	11481
22	200	400(202)	13660(1)	2	17728
23	200	400(201)	17088(1)	2	18617
24	200	400(200)	20469(0)	2	19140
25	200	600	1797	1	13811*

ID	$\|V(G)\|$	$\|E(G)\|$	$\|E(C)\|$	Type	Opt
26	200	600	3594	1	Infeas
27	200	600	5391	1	Infeas
28	200	600(201)	34504(0)	2	20716
29	200	600(200)	42930(0)	2	18025
30	200	600(200)	50984(0)	2	20864
31	200	800	3196	1	21480*
32	200	800	6392	1	-
33	200	800	9588	1	-
34	200	800	15980	1	Infeas
35	200	800(744)	62625(51804)	2	39895
36	200	800(203)	78387(0)	2	37671
37	200	800(200)	93978(0)	2	38798
38	300	600(299)	31001(0)	2	43721
39	300	600(301)	38216(1)	2	44267
40	300	600(302)	45310(0)	2	43071
41	300	800(768)	3196(2810)	1	Infeas
42	300	800(300)	59600(0)	2	43125
43	300	800(300)	74500(0)	2	42292
44	300	800(300)	89300(0)	2	44114
45	300	1000	4995	1	-
46	300	1000(999)	9990(9816)	1	-
47	300	1000(998)	14985(14924)	1	Infeas
48	300	1000(300)	96590(0)	2	71562
49	300	1000(299)	120500(0)	2	76345
50	300	1000(299)	144090(0)	2	78880

Table A.3. Instances in the CCPR class and the effect of preprocessing.

ID	$\|V(G)\|$	$\|E(G)\|$	$\|E(C)\|$	Seed	Opt	ID	$\|V(G)\|$	$\|E(G)\|$	$\|E(C)\|$	Seed	Opt	ID	$\|V(G)\|$	$\|E(G)\|$	$\|E(C)\|$	Seed	Opt
51	25	60	18	1	347	71	25	90	161	121	305	91	25	120	500	241	329
52	25	60(58)	18(16)	7	389	72	25	90(89)	161(155)	127	339	92	25	120	500	247	339
53	25	60	18	13	353	73	25	90	161	133	344	93	25	120	500	253	368
54	25	60	18	19	346	74	25	90	161	139	329	94	25	120	500	259	311
55	25	60	18	25	336	75	25	90	161	145	326	95	25	120	500	265	321
56	25	60	71	31	381	76	25	90	281	151	349	96	50	245	299	271	619
57	25	60	71	37	390	77	25	90	281	157	385	97	50	245	299	277	604
58	25	60(59)	71(67)	43	372	78	25	90	281	163	335	98	50	245	299	283	634
59	25	60(58)	71(60)	49	357	79	25	90	281	169	348	99	50	245	299	289	616
60	25	60	71	55	406	80	25	90	281	175	357	100	50	245	299	295	595
61	25	60(52)	124(77)	61	385	81	25	120	72	181	282	101	50	245	1196	301	678
62	25	60(52)	124(88)	67	432	82	25	120	72	187	294	102	50	245	1196	307	681
63	25	60	124	73	458	83	25	120	72	193	284	103	50	245	1196	313	709
64	25	60	124	79	400	84	25	120	72	199	281	104	50	245	1196	319	639
65	25	60	124	85	420	85	25	120	72	205	292	105	50	245	1196	325	681
66	25	90	41	91	311	86	25	120	286	211	321	106	50	245	2093	331	816
67	25	90(89)	41(39)	97	306	87	25	120	286	217	317	107	50	245	2093	337	835
68	25	90	41	103	299	88	25	120	286	223	284	108	50	245	2093	343	810
69	25	90	41	109	297	89	25	120	286	229	311	109	50	245	2093	349	832
70	25	90	41	115	318	90	25	120	286	235	290	110	50	245	2093	355	769

Table A.3. Instances in the CCPR class and the effect of preprocessing (cont.).

ID	$\|V(G)\|$	$\|E(G)\|$	$\|E(C)\|$	Seed	Opt	ID	$\|V(G)\|$	$\|E(G)\|$	$\|E(C)\|$	Seed	Opt	ID	$\|V(G)\|$	$\|E(G)\|$	$\|E(C)\|$	Seed	Opt
111	50	367	672	361	570	131	50	490	4793	481	594	151	75	555	10762	601	-
112	50	367	672	367	561	132	50	490	4793	487	579	152	75	555	10762	607	-
113	50	367	672	373	573	133	50	490	4793	493	589	153	75	555	10762	613	-
114	50	367	672	379	560	134	50	490	4793	499	577	154	75	555	10762	619	-
115	50	367	672	385	549	135	50	490	4793	505	592	155	75	555	10762	625	-
116	50	367	2687	391	612	136	50	490	8387	511	666*	156	75	832	3457	631	798
117	50	367	2687	397	615	137	50	490	8387	517	638*	157	75	832	3457	637	821
118	50	367	2687	403	587	138	50	490	8387	523	678*	158	75	832	3457	643	816
119	50	367	2687	409	634	139	50	490	8387	529	682*	159	75	832	3457	649	820
120	50	367	2687	415	643	140	50	490	8387	535	657	160	75	832	3457	655	815
121	50	367	4702	421	726*	141	75	555	1538	541	868	161	75	832	13828	661	896*
122	50	367	4702	427	744*	142	75	555	1538	547	871	162	75	832	13828	667	934*
123	50	367	4702	433	778*	143	75	555	1538	553	838	163	75	832	13828	673	891*
124	50	367	4702	439	701*	144	75	555	1538	559	855	164	75	832	13828	679	907*
125	50	367	4702	445	764*	145	75	555	1538	565	857	165	75	832	13828	685	896*
126	50	490	1199	451	548	146	75	555	6150	571	1047*	166	75	832	24199	691	1439*
127	50	490	1199	457	530	147	75	555	6150	577	1036*	167	75	832	24199	697	-
128	50	490	1199	463	549	148	75	555	6150	583	1035*	168	75	832	24199	703	-
129	50	490	1199	469	540	149	75	555	6150	589	992	169	75	832	24199	709	-
130	50	490	1199	475	540	150	75	555	6150	595	994*	170	75	832	24199	715	-

Table A.3. Instances in the CCPR class and the effect of preprocessing (cont.).

ID	$\|V(G)\|$	$\|E(G)\|$	$\|E(C)\|$	Seed	Opt	ID	$\|V(G)\|$	$\|E(G)\|$	$\|E(C)\|$	Seed	Opt	ID	$\|V(G)\|$	$\|E(G)\|$	$\|E(C)\|$	Seed	Opt
171	75	1110	6155	721	787	191	100	990	19583	841	1467*	211	100	1485	77131	961	-
172	75	1110	6155	727	785	192	100	990	19583	847	1491*	212	100	1485	77131	967	-
173	75	1110	6155	733	783	193	100	990	19583	853	1496*	213	100	1485	77131	973	-
174	75	1110	6155	739	784	194	100	990	19583	859	1441*	214	100	1485	77131	979	-
175	75	1110	6155	745	797	195	100	990	19583	865	1552*	215	100	1485	77131	985	-
176	75	1110	24620	751	854*	196	100	990	34269	871	-	216	100	1980	19593	991	1031
177	75	1110	24620	757	848*	197	100	990	34269	877	-	217	100	1980	19593	997	1036
178	75	1110	24620	763	876*	198	100	990	34269	883	-	218	100	1980	19593	1003	1024
179	75	1110	24620	769	862*	199	100	990	34269	889	-	219	100	1980	19593	1009	1025
180	75	1110	24620	775	858*	200	100	990	34269	895	-	220	100	1980	19593	1015	1028
181	75	1110	43085	781	1360*	201	100	1485	11019	901	1079	221	100	1980	78369	1021	1209*
182	75	1110	43085	787	1259*	202	100	1485	11019	907	1056	222	100	1980	78369	1027	1157*
183	75	1110	43085	793	1194*	203	100	1485	11019	913	1059	223	100	1980	78369	1033	1202*
184	75	1110	43085	799	1345*	204	100	1485	11019	919	1046	224	100	1980	78369	1039	1219*
185	75	1110	43085	805	1342*	205	100	1485	11019	925	1072	225	100	1980	78369	1045	1182*
186	100	990	4896	811	1119	206	100	1485	44075	931	1304*	226	100	1980	137145	1051	-
187	100	990	4896	817	1137	207	100	1485	44075	937	1291*	227	100	1980	137145	1057	-
188	100	990	4896	823	1113	208	100	1485	44075	943	1316*	228	100	1980	137145	1063	-
189	100	990	4896	829	1110	209	100	1485	44075	949	1286*	229	100	1980	137145	1069	-
190	100	990	4896	835	1090	210	100	1485	44075	955	1297*	230	100	1980	137145	1075	-

Table A.4. Instances in the SKT class and the effect of preprocessing.

ID	$\|V(G)\|$	$\|E(G)\|$	$\|E(C)\|$	Seed	Opt
231	40	234	5452	175	3900
232	40	234	5452	1649	3900
233	40	234	5452	6625	3900
234	40	234	8178	1769	3900
235	40	234	8178	7299	3900
236	40	234	8178	7723	3900
237	40	234(232)	10904(10715)	846	3900
238	40	234(231)	10904(10579)	4351	3900
239	40	234(232)	10904(10713)	5487	3900
240	40	234(221)	13630(12074)	2171	3900
241	40	234(221)	13630(12040)	4848	3900
242	40	234(217)	13630(11628)	6091	3900
243	40	390	22756	4371	3900
244	40	390	22756	3935	3900
245	40	390	22756	4262	3900
246	40	390	30342	1017	3900
247	40	390	30342	6630	3900
248	40	390	30342	6730	3900

APPENDIX B: LAGRANGEAN RELAXATION RESULTS

Table B.1. Column expressions.

Column	Description
ID	Unique ID of the problem instance.
LB	Lower bound value.
LB Gap(%)	Lower bound relative percentage gap. A "-" sign indicates that the feasibility of the problem is unknown whereas "Infeas" denotes the infeasibility of the problem.
UB	Upper bound value. A "-" sign indicates that no feasible solution is found by the algorithm and NA denotes data is not available.
UB Gap(%)	Upper bound relative percentage gap. A "-" sign indicates that no feasible solution is found by the algorithm and NA denotes data is not available.
CPU(s)	CPU time in seconds. NA denotes data is not available.

Table B.2. Lower bounds on the CCPR instance class.

	HDA			LD-davol			Subgradient		
ID	LB	LB Gap(%)	CPU(s)	LB	LB Gap(%)	CPU(s)	LB	LB Gap(%)	CPU(s)
51	347.0	0.0	0.04	347.0	0.0	0.2	347.0	0.0	0.03
52	389.0	0.0	0.04	389.0	0.0	0.1	389.0	0.0	0.01
53	353.0	0.0	0.04	353.0	0.0	0.1	353.0	0.0	0.01
54	346.0	0.0	0.04	346.0	0.0	0.1	346.0	0.0	0.01
55	336.0	0.0	0.04	336.0	0.0	0.1	336.0	0.0	0.01
56	379.63	0.36	0.06	380.0	0.26	0.5	379.66	0.35	0.02
57	381.5	2.18	0.05	382.0	2.05	0.8	381.44	2.19	0.02
58	372.0	0.0	0.05	372.0	0.0	0.3	371.8	0.05	0.01
59	357.0	0.0	0.05	357.0	0.0	0.3	356.79	0.06	0.01
60	406.0	0.0	0.05	406.0	0.0	0.3	406.0	0.0	0.01
61	385.0	0.0	0.06	385.0	0.0	0.8	384.64	0.09	0.01
62	432.0	0.0	0.06	432.0	0.0	0.5	432.0	0.0	0.01
63	424.48	7.32	0.07	458.0	0.0	3.4	423.52	7.53	0.03
64	398.0	0.5	0.07	400.0	0.0	2.4	397.93	0.52	0.03
65	406.77	3.15	0.08	420.0	0.0	1.5	406.16	3.29	0.03
66	310.1	0.29	0.05	311.0	0.0	0.3	311.0	0.0	0.01
67	306.0	0.0	0.05	306.0	0.0	0.1	306.0	0.0	0.01
68	299.0	0.0	0.05	299.0	0.0	0.2	299.0	0.0	0.02
69	297.0	0.0	0.05	297.0	0.0	0.2	297.0	0.0	0.01
70	318.0	0.0	0.05	318.0	0.0	0.1	318.0	0.0	0.01
71	305.0	0.0	0.08	305.0	0.0	0.4	305.0	0.0	0.01
72	339.0	0.0	0.09	339.0	0.0	0.4	339.0	0.0	0.02
73	344.0	0.0	0.08	344.0	0.0	0.4	344.0	0.0	0.02
74	328.0	0.3	0.09	328.0	0.3	0.6	327.93	0.32	0.02
75	325.0	0.31	0.08	325.0	0.31	0.8	324.98	0.31	0.03
76	347.93	0.31	0.1	349.0	0.0	2.1	347.95	0.3	0.04
77	370.63	3.73	0.1	379.0	1.56	14.5	370.56	3.75	0.04
78	330.78	1.26	0.11	334.0	0.3	3.7	330.76	1.27	0.04
79	334.67	3.83	0.1	344.0	1.15	6.2	334.6	3.85	0.03
80	350.05	1.95	0.1	357.0	0.0	2.5	350.34	1.87	0.04
81	282.0	0.0	0.07	282.0	0.0	0.3	282.0	0.0	0.01
82	294.0	0.0	0.06	294.0	0.0	0.3	294.0	0.0	0.01
83	284.0	0.0	0.06	284.0	0.0	0.3	284.0	0.0	0.01
84	281.0	0.0	0.06	281.0	0.0	0.3	281.0	0.0	0.02
85	292.0	0.0	0.06	292.0	0.0	0.3	292.0	0.0	0.01
86	320.25	0.23	0.12	321.0	0.0	0.7	320.17	0.26	0.03

Table B.2. Lower bounds on the CCPR instance class (cont.).

	HDA			LD-davol			Subgradient		
ID	LB	LB Gap(%)	CPU(s)	LB	LB Gap(%)	CPU(s)	LB	LB Gap(%)	CPU(s)
87	317.0	0.0	0.12	317.0	0.0	1.1	316.98	0.01	0.03
88	284.0	0.0	0.12	284.0	0.0	0.5	284.0	0.0	0.01
89	311.0	0.0	0.12	311.0	0.0	0.7	311.0	0.0	0.02
90	290.0	0.0	0.13	290.0	0.0	0.4	290.0	0.0	0.03
91	319.24	2.97	0.16	326.0	0.91	4.3	319.19	2.98	0.04
92	326.05	3.82	0.16	334.0	1.47	6.3	326.03	3.83	0.05
93	354.49	3.67	0.16	367.0	0.27	24.7	354.44	3.69	0.04
94	306.54	1.43	0.16	310.0	0.32	8.5	306.57	1.43	0.05
95	318.01	0.93	0.16	321.0	0.0	2.6	318.27	0.85	0.06
96	619.0	0.0	0.34	619.0	0.0	1.3	618.57	0.07	0.04
97	604.0	0.0	0.36	604.0	0.0	0.9	604.0	0.0	0.02
98	634.0	0.0	0.34	634.0	0.0	0.7	634.0	0.0	0.02
99	615.5	0.08	0.34	616.0	0.0	1.2	615.5	0.08	0.07
100	595.0	0.0	0.34	595.0	0.0	1.3	594.97	0.0	0.12
101	668.5	1.4	0.57	674.0	0.59	124.9	668.73	1.37	0.12
102	655.49	3.75	0.56	678.0	0.44	134.2	655.94	3.68	0.16
103	678.22	4.34	0.59	695.0	1.97	184.4	678.53	4.3	0.18
104	634.04	0.78	0.56	637.0	0.31	47.9	634.14	0.76	0.17
105	658.82	3.26	0.56	672.0	1.32	125.9	658.9	3.25	0.12
106	661.27	18.96	0.76	774.0	5.15	3607.0	661.32	18.96	0.31
107	705.98	15.45	0.75	803.0	3.83	3601.6	705.74	15.48	0.16
108	666.45	17.72	0.74	762.0	5.93	3609.3	666.26	17.75	0.19
109	680.81	18.17	0.76	784.0	5.77	3603.9	680.72	18.18	0.19
110	693.86	9.77	0.74	758.0	1.43	3282.8	694.02	9.75	0.2
111	570.0	0.0	0.82	570.0	0.0	1.9	570.0	0.0	0.02
112	561.0	0.0	0.86	561.0	0.0	1.8	560.99	0.0	0.04
113	573.0	0.0	0.86	573.0	0.0	1.7	573.0	0.0	0.1
114	560.0	0.0	0.86	560.0	0.0	1.9	560.0	0.0	0.02
115	549.0	0.0	0.83	549.0	0.0	1.8	549.0	0.0	0.07
116	596.44	2.54	1.26	607.0	0.82	228.9	596.44	2.54	0.32
117	596.67	2.98	1.27	608.0	1.14	254.4	596.72	2.97	0.31
118	577.09	1.69	1.26	585.0	0.34	129.7	577.12	1.68	0.24
119	608.47	4.03	1.26	626.0	1.26	279.4	608.38	4.04	0.22
120	636.06	1.08	1.26	640.0	0.47	108.4	636.12	1.07	0.34
121	624.43	13.99	1.66	690.0	4.96	3601.3	624.37	14.0	0.65
122	635.39	14.6	1.65	703.0	5.51	3608.4	635.38	14.6	0.54

Table B.2. Lower bounds on the CCPR instance class (cont.).

	HDA			LD-davol			Subgradient		
ID	LB	LB Gap(%)	CPU(s)	LB	LB Gap(%)	CPU(s)	LB	LB Gap(%)	CPU(s)
123	646.89	16.85	1.64	721.0	7.33	3606.4	646.86	16.86	0.41
124	597.47	14.77	1.72	668.0	4.71	3601.0	597.4	14.78	0.33
125	665.14	12.94	1.71	725.0	5.1	3609.0	665.09	12.95	0.33
126	548.0	0.0	1.68	548.0	0.0	2.2	547.97	0.01	0.05
127	530.0	0.0	1.66	530.0	0.0	2.1	530.0	0.0	0.27
128	549.0	0.0	1.65	549.0	0.0	2.7	549.0	0.0	0.11
129	540.0	0.0	1.67	540.0	0.0	2.2	540.0	0.0	0.24
130	540.0	0.0	1.65	540.0	0.0	2.4	540.0	0.0	0.03
131	584.26	1.64	2.38	592.0	0.34	294.8	584.55	1.59	0.5
132	559.85	3.31	2.4	570.0	1.55	323.7	559.94	3.29	0.44
133	578.34	1.81	2.43	587.0	0.34	235.0	578.44	1.79	0.44
134	565.41	2.01	2.39	571.0	1.04	137.7	565.42	2.01	0.41
135	577.44	2.46	2.4	589.0	0.51	262.5	577.45	2.46	0.5
136	574.99	13.67	3.08	620.0	6.91	3604.6	574.92	13.67	0.57
137	569.0	10.82	3.07	613.0	3.92	3609.9	568.93	10.83	0.62
138	592.2	12.65	3.08	647.0	4.57	3600.8	592.18	12.66	0.73
139	592.5	13.12	3.07	655.0	3.96	3609.3	592.42	13.14	0.57
140	587.45	10.59	3.09	635.0	3.35	3607.7	587.41	10.59	0.83
141	868.0	0.0	2.51	868.0	0.0	5.0	868.0	0.0	0.36
142	870.93	0.01	2.49	871.0	0.0	3.8	871.0	0.0	0.19
143	838.0	0.0	2.45	838.0	0.0	4.7	837.98	0.0	0.14
144	855.0	0.0	2.47	855.0	0.0	3.6	854.85	0.02	0.06
145	857.0	0.0	2.47	857.0	0.0	3.5	857.0	0.0	0.16
146	942.34	10.0	3.49	1018.0	2.77	3609.1	942.28	10.0	0.55
147	937.61	9.5	3.54	997.0	3.76	3606.2	937.75	9.48	0.42
148	910.9	11.99	3.48	985.0	4.83	3603.2	910.81	12.0	0.46
149	913.03	7.96	3.53	960.0	3.23	3608.5	913.3	7.93	0.5
150	897.88	9.67	3.54	953.0	4.12	3600.4	897.79	9.68	0.5
151	922.49	-	4.44	1098.0	-	1800.0	922.32	-	0.89
152	947.99	-	4.4	1107.0	-	1800.0	947.8	-	1.0
153	901.94	-	4.44	1069.0	-	1800.0	901.81	-	0.66
154	888.35	-	4.44	1036.0	-	1800.0	888.29	-	0.82
155	914.72	-	4.42	1081.0	-	1800.0	914.83	-	0.95
156	797.77	0.03	6.86	798.0	0.0	6.6	797.99	0.0	0.25
157	820.0	0.12	6.83	820.0	0.12	8.6	819.98	0.12	0.27
158	815.32	0.08	6.87	815.0	0.12	7.7	815.3	0.09	0.32

Table B.2. Lower bounds on the CCPR instance class (cont.).

	HDA			LD-davol			Subgradient		
ID	LB	LB Gap(%)	CPU(s)	LB	LB Gap(%)	CPU(s)	LB	LB Gap(%)	CPU(s)
159	819.97	0.0	7.16	820.0	0.0	8.6	820.0	0.0	0.43
160	814.99	0.0	7.23	815.0	0.0	8.6	814.99	0.0	0.38
161	830.85	7.27	9.08	865.0	3.46	3601.6	830.91	7.26	0.85
162	859.49	7.98	9.05	889.0	4.82	3601.5	859.42	7.99	0.97
163	829.0	6.96	9.04	858.0	3.7	3607.2	828.92	6.97	1.03
164	842.24	7.14	8.98	879.0	3.09	3605.6	842.36	7.13	0.79
165	846.7	5.5	9.05	875.0	2.34	3605.1	846.92	5.48	1.36
166	869.99	39.54	12.24	965.0	32.94	1800.0	869.92	39.55	2.77
167	837.71	-	12.33	921.0	-	1800.0	837.91	-	2.48
168	842.0	-	12.01	925.0	-	1800.0	841.91	-	1.66
169	862.99	-	11.98	967.0	-	1800.0	862.92	-	1.88
170	878.66	-	12.14	974.0	-	1800.0	878.91	-	1.68
171	787.0	0.0	14.88	787.0	0.0	12.6	787.0	0.0	0.44
172	785.0	0.0	14.82	785.0	0.0	13.7	784.91	0.01	0.17
173	783.0	0.0	14.78	783.0	0.0	8.9	782.99	0.0	0.64
174	783.88	0.02	14.94	784.0	0.0	7.8	784.0	0.0	0.69
175	796.5	0.06	14.86	797.0	0.0	13.2	796.5	0.06	0.73
176	816.88	4.35	19.16	838.0	1.87	3602.9	816.96	4.34	2.33
177	797.98	5.9	19.28	828.0	2.36	3606.9	797.95	5.9	1.98
178	808.43	7.71	19.44	847.0	3.31	3605.9	808.45	7.71	2.01
179	807.97	6.27	19.31	836.0	3.02	3603.4	807.95	6.27	1.52
180	803.99	6.29	19.32	830.0	3.26	3606.3	803.95	6.3	2.29
181	816.97	39.93	27.33	882.0	35.15	1800.0	816.95	39.93	4.36
182	804.0	36.14	26.74	861.0	31.61	1800.0	803.95	36.14	3.4
183	827.29	30.71	26.9	890.0	25.46	1800.0	827.45	30.7	3.81
184	799.0	40.59	27.17	864.0	35.76	1800.0	798.95	40.6	4.03
185	806.0	39.94	27.14	862.0	35.77	1800.0	805.94	39.94	4.15
186	1118.48	0.05	11.59	1118.0	0.09	15.1	1118.47	0.05	0.5
187	1134.74	0.2	11.43	1137.0	0.0	20.4	1134.92	0.18	0.6
188	1112.45	0.05	11.71	1113.0	0.0	25.1	1112.47	0.05	0.74
189	1109.63	0.03	11.48	1110.0	0.0	17.4	1109.92	0.01	0.64
190	1088.67	0.12	11.99	1089.0	0.09	17.5	1088.88	0.1	0.64
191	1175.62	19.86	15.37	1282.0	12.61	1800.0	1175.62	19.86	1.5
192	1141.98	23.41	14.98	1242.0	16.7	1800.0	1141.85	23.42	1.97
193	1140.74	23.75	14.8	1236.0	17.38	1800.0	1140.84	23.74	1.8
194	1178.06	18.25	14.82	1284.0	10.9	1800.0	1178.35	18.23	1.52

Table B.2. Lower bounds on the CCPR instance class (cont.).

	HDA			LD-davol			Subgradient		
ID	LB	LB Gap(%)	CPU(s)	LB	LB Gap(%)	CPU(s)	LB	LB Gap(%)	CPU(s)
195	1176.36	24.2	15.36	1278.0	17.65	1800.1	1176.37	24.2	2.23
196	1126.42	-	22.2	1266.0	-	1800.0	1126.35	-	3.02
197	1148.91	-	24.59	1293.0	-	1800.0	1148.83	-	2.98
198	1179.91	-	21.17	1318.0	-	1800.0	1179.85	-	3.13
199	1146.74	-	20.3	1275.0	-	1800.0	1146.86	-	3.13
200	1154.99	-	20.42	1311.0	-	1800.0	1154.84	-	2.35
201	1077.94	0.1	34.81	1078.0	0.09	41.5	1077.98	0.09	0.85
202	1054.88	0.11	34.46	1055.0	0.09	39.6	1054.98	0.1	0.76
203	1059.0	0.0	34.29	1059.0	0.0	28.8	1058.99	0.0	0.88
204	1045.99	0.0	34.09	1046.0	0.0	32.8	1046.0	0.0	0.83
205	1071.22	0.07	34.17	1072.0	0.0	58.7	1071.38	0.06	0.79
206	1096.86	15.88	46.07	1152.0	11.66	1800.0	1096.94	15.88	3.72
207	1090.97	15.49	45.6	1155.0	10.53	1800.0	1090.94	15.5	3.24
208	1084.44	17.6	46.54	1144.0	13.07	1800.0	1084.43	17.6	3.42
209	1089.38	15.29	46.01	1142.0	11.2	1800.0	1089.43	15.29	3.42
210	1088.39	16.08	45.84	1145.0	11.72	1800.0	1088.44	16.08	4.4
211	1083.49	-	60.68	1167.0	-	1800.0	1083.43	-	6.74
212	1085.99	-	60.29	1170.0	-	1800.0	1085.93	-	8.48
213	1093.98	-	60.69	1184.0	-	1800.0	1093.94	-	5.86
214	1105.91	-	60.83	1185.0	-	1800.0	1105.94	-	8.42
215	1090.9	-	60.63	1157.0	-	1801.1	1090.94	-	6.92
216	1030.91	0.01	76.15	1031.0	0.0	56.4	1030.94	0.01	1.58
217	1034.88	0.11	76.02	1036.0	0.0	53.0	1034.99	0.1	1.68
218	1024.0	0.0	76.43	1024.0	0.0	60.5	1023.99	0.0	2.09
219	1024.98	0.0	76.67	1025.0	0.0	45.0	1025.0	0.0	1.63
220	1027.5	0.05	77.49	1028.0	0.0	45.8	1027.49	0.05	1.63
221	1060.5	12.28	106.68	1107.0	8.44	1800.0	1060.67	12.27	6.59
222	1033.99	10.63	106.82	1069.0	7.61	1800.1	1033.96	10.63	8.2
223	1050.54	12.6	104.96	1096.0	8.82	1800.0	1050.82	12.58	9.79
224	1048.1	14.02	103.83	1092.0	10.42	1800.1	1048.37	14.0	8.48
225	1048.88	11.26	103.99	1094.0	7.45	1800.0	1049.39	11.22	9.2
226	1041.79	-	129.47	1101.0	-	1800.0	1041.96	-	14.42
227	1066.49	-	129.26	1127.0	-	1800.0	1066.46	-	13.44
228	1052.36	-	130.38	1112.0	-	1800.0	1052.47	-	17.37
229	1062.85	-	130.01	1116.0	-	1800.0	1062.97	-	17.34
230	1055.48	-	132.96	1114.0	-	1800.0	1055.46	-	17.37
Average:	**719.6**	**5.66**	**17.08**	**752.65**	**3.32**	**1069.48**	**719.61**	**5.66**	**1.59**

Table B.3. Upper bounds on the CCPR instance class.

	HDA+			DUBH+LS				HDA+			DUBH+LS		
ID	UB	UB Gap(%)	CPU(s)	UB	UB Gap(%)	CPU(s)	ID	UB	UB Gap(%)	CPU(s)	UB	UB Gap(%)	CPU(s)
51	347.0	0.0	0.05	347.0	0.0	0.04	74	331.0	0.61	0.11	329.0	0.0	1.04
52	389.0	0.0	0.04	389.0	0.0	0.01	75	327.0	0.31	0.11	326.0	0.0	1.17
53	353.0	0.0	0.04	353.0	0.0	0.01	76	349.0	0.0	0.13	349.0	0.0	1.03
54	346.0	0.0	0.05	346.0	0.0	0.01	77	385.0	0.0	0.16	385.0	0.0	0.92
55	336.0	0.0	0.04	336.0	0.0	0.01	78	335.0	0.0	0.14	335.0	0.0	1.47
56	381.0	0.0	0.07	381.0	0.0	0.45	79	358.0	2.87	0.18	348.0	0.0	0.7
57	390.0	0.0	0.07	390.0	0.0	0.54	80	359.0	0.56	0.17	357.0	0.0	1.2
58	372.0	0.0	0.06	372.0	0.0	0.02	81	282.0	0.0	0.06	282.0	0.0	0.12
59	357.0	0.0	0.06	357.0	0.0	0.04	82	294.0	0.0	0.06	294.0	0.0	0.02
60	406.0	0.0	0.06	406.0	0.0	0.03	83	284.0	0.0	0.08	284.0	0.0	0.02
61	385.0	0.0	0.07	385.0	0.0	0.14	84	281.0	0.0	0.07	281.0	0.0	0.02
62	432.0	0.0	0.07	432.0	0.0	0.04	85	292.0	0.0	0.06	292.0	0.0	0.01
63	474.0	3.49	0.1	458.0	0.0	0.32	86	321.0	0.0	0.16	321.0	0.0	0.83
64	400.0	0.0	0.08	400.0	0.0	0.46	87	317.0	0.0	0.17	317.0	0.0	0.21
65	421.0	0.24	0.12	420.0	0.0	0.57	88	284.0	0.0	0.12	284.0	0.0	0.52
66	311.0	0.0	0.05	311.0	0.0	0.02	89	312.0	0.32	0.17	311.0	0.0	0.12
67	306.0	0.0	0.06	306.0	0.0	0.01	90	290.0	0.0	0.13	290.0	0.0	0.02
68	299.0	0.0	0.05	299.0	0.0	0.13	91	341.0	3.65	0.26	331.0	0.61	1.17
69	297.0	0.0	0.05	297.0	0.0	0.01	92	347.0	2.36	0.25	339.0	0.0	1.12
70	318.0	0.0	0.05	318.0	0.0	0.01	93	383.0	4.08	0.3	371.0	0.82	0.86
71	305.0	0.0	0.09	305.0	0.0	0.03	94	314.0	0.96	0.22	311.0	0.0	1.88
72	339.0	0.0	0.11	339.0	0.0	0.04	95	325.0	1.25	0.25	321.0	0.0	1.38
73	344.0	0.0	0.09	344.0	0.0	0.04	96	619.0	0.0	0.37	619.0	0.0	0.36

Table B.3. Upper bounds on the CCPR instance class (cont.).

	HDA+			DUBH+LS				HDA+			DUBH+LS		
ID	UB	UB Gap(%)	CPU(s)	UB	UB Gap(%)	CPU(s)	ID	UB	UB Gap(%)	CPU(s)	UB	UB Gap(%)	CPU(s)
97	604.0	0.0	0.38	604.0	0.0	0.11	120	688.0	7.0	2.45	645.0	0.31	22.55
98	634.0	0.0	0.35	634.0	0.0	0.1	121	-	-	-	759.0	4.55	1.21
99	616.0	0.0	0.4	616.0	0.0	0.74	122	-	-	-	753.0	1.21	1.77
100	595.0	0.0	0.44	595.0	0.0	0.12	123	-	-	-	819.0	5.27	1.4
101	698.0	2.95	0.96	678.0	0.0	13.16	124	-	-	-	744.0	6.13	1.41
102	721.0	5.87	0.99	691.0	1.47	9.51	125	868.0	13.61	3.36	798.0	4.45	1.31
103	725.0	2.26	1.0	713.0	0.56	7.05	126	552.0	0.73	2.03	548.0	0.0	1.08
104	656.0	2.66	0.86	639.0	0.0	14.62	127	531.0	0.19	1.88	530.0	0.0	0.58
105	748.0	9.84	1.01	687.0	0.88	10.22	128	549.0	0.0	1.76	549.0	0.0	0.58
106	-	-	-	861.0	5.51	0.79	129	541.0	0.19	2.01	540.0	0.0	0.38
107	-	-	-	-	-	0.99	130	540.0	0.0	1.71	540.0	0.0	0.5
108	-	-	-	-	-	1.2	131	629.0	5.89	4.63	594.0	0.0	18.64
109	-	-	-	-	-	0.94	132	650.0	12.26	4.9	582.0	0.52	14.82
110	-	-	-	783.0	1.82	1.38	133	657.0	11.54	4.75	593.0	0.68	23.24
111	570.0	0.0	0.88	570.0	0.0	0.15	134	643.0	11.44	4.74	579.0	0.35	14.68
112	561.0	0.0	0.98	561.0	0.0	0.34	135	670.0	13.18	4.71	592.0	0.0	17.54
113	573.0	0.0	1.06	573.0	0.0	0.82	136	812.0	21.92	6.54	703.0	5.56	3.22
114	560.0	0.0	0.87	560.0	0.0	0.17	137	-	-	-	679.0	6.43	3.6
115	551.0	0.36	0.99	549.0	0.0	0.62	138	-	-	-	712.0	5.01	5.98
116	657.0	7.35	2.37	617.0	0.82	14.45	139	-	-	-	717.0	5.13	4.91
117	663.0	7.8	2.24	620.0	0.81	17.1	140	828.0	26.03	6.95	691.0	5.18	4.11
118	635.0	8.18	2.33	589.0	0.34	16.05	141	NA	NA	NA	868.0	0.0	5.14
119	721.0	13.72	2.4	635.0	0.16	14.99	142	NA	NA	NA	871.0	0.0	5.09

Table B.3. Upper bounds on the CCPR instance class (cont.).

	HDA+			DUBH+LS				HDA+			DUBH+LS		
ID	UB	UB Gap(%)	CPU(s)	UB	UB Gap(%)	CPU(s)	ID	UB	UB Gap(%)	CPU(s)	UB	UB Gap(%)	CPU(s)
143	NA	NA	NA	838.0	0.0	4.17	166	NA	NA	NA	-	-	4.43
144	NA	NA	NA	855.0	0.0	2.32	167	NA	NA	NA	-	-	3.83
145	NA	NA	NA	857.0	0.0	4.59	168	NA	NA	NA	-	-	2.61
146	NA	NA	NA	1087.0	3.82	15.63	169	NA	NA	NA	-	-	3.14
147	NA	NA	NA	1078.0	4.05	17.03	170	NA	NA	NA	-	-	2.78
148	NA	NA	NA	1080.0	4.35	8.26	171	NA	NA	NA	787.0	0.0	5.92
149	NA	NA	NA	1025.0	3.33	7.89	172	NA	NA	NA	785.0	0.0	5.94
150	NA	NA	NA	1040.0	4.63	12.46	173	NA	NA	NA	783.0	0.0	5.87
151	NA	NA	NA	-	-	1.6	174	NA	NA	NA	784.0	0.0	5.81
152	NA	NA	NA	-	-	2.21	175	NA	NA	NA	797.0	0.0	22.33
153	NA	NA	NA	-	-	1.53	176	NA	NA	NA	874.0	2.34	162.91
154	NA	NA	NA	-	-	2.08	177	NA	NA	NA	862.0	1.65	146.9
155	NA	NA	NA	-	-	2.19	178	NA	NA	NA	907.0	3.54	111.02
156	NA	NA	NA	798.0	0.0	10.23	179	NA	NA	NA	899.0	4.29	117.12
157	NA	NA	NA	821.0	0.0	61.79	180	NA	NA	NA	887.0	3.38	165.3
158	NA	NA	NA	816.0	0.0	10.93	181	NA	NA	NA	-	-	5.15
159	NA	NA	NA	820.0	0.0	4.46	182	NA	NA	NA	-	-	3.92
160	NA	NA	NA	815.0	0.0	12.04	183	NA	NA	NA	-	-	5.33
161	NA	NA	NA	925.0	3.24	69.83	184	NA	NA	NA	-	-	4.96
162	NA	NA	NA	940.0	0.64	79.94	185	NA	NA	NA	-	-	4.78
163	NA	NA	NA	923.0	3.59	64.62	186	NA	NA	NA	1119.0	0.0	69.17
164	NA	NA	NA	935.0	3.09	79.29	187	NA	NA	NA	1137.0	0.0	134.03
165	NA	NA	NA	912.0	1.79	80.1	188	NA	NA	NA	1113.0	0.0	42.72

Table B.3. Upper bounds on the CCPR instance class (cont.).

	HDA+			DUBH+LS				HDA+			DUBH+LS		
ID	UB	UB Gap(%)	CPU(s)	UB	UB Gap(%)	CPU(s)	ID	UB	UB Gap(%)	CPU(s)	UB	UB Gap(%)	CPU(s)
189	NA	NA	NA	1110.0	0.0	87.93	210	NA	NA	NA	1343.0	3.55	30.63
190	NA	NA	NA	1090.0	0.0	156.87	211	NA	NA	NA	-	-	8.46
191	NA	NA	NA	1512.0	3.07	4.09	212	NA	NA	NA	-	-	11.52
192	NA	NA	NA	-	-	4.08	213	NA	NA	NA	-	-	7.66
193	NA	NA	NA	-	-	3.63	214	NA	NA	NA	-	-	12.3
194	NA	NA	NA	-	-	3.64	215	NA	NA	NA	-	-	10.39
195	NA	NA	NA	-	-	4.25	216	NA	NA	NA	1031.0	0.0	124.86
196	NA	NA	NA	-	-	4.71	217	NA	NA	NA	1036.0	0.0	253.32
197	NA	NA	NA	-	-	4.48	218	NA	NA	NA	1024.0	0.0	97.54
198	NA	NA	NA	-	-	4.6	219	NA	NA	NA	1025.0	0.0	65.56
199	NA	NA	NA	-	-	4.97	220	NA	NA	NA	1028.0	0.0	144.01
200	NA	NA	NA	-	-	3.3	221	NA	NA	NA	1301.0	7.61	123.88
201	NA	NA	NA	1079.0	0.0	219.77	222	NA	NA	NA	1240.0	7.17	129.81
202	NA	NA	NA	1056.0	0.0	188.43	223	NA	NA	NA	1259.0	4.74	184.38
203	NA	NA	NA	1059.0	0.0	27.98	224	NA	NA	NA	1265.0	3.77	183.5
204	NA	NA	NA	1046.0	0.0	61.84	225	NA	NA	NA	1267.0	7.19	161.28
205	NA	NA	NA	1072.0	0.0	53.51	226	NA	NA	NA	-	-	18.13
206	NA	NA	NA	1359.0	4.22	23.48	227	NA	NA	NA	-	-	16.09
207	NA	NA	NA	1390.0	7.67	22.42	228	NA	NA	NA	-	-	21.69
208	NA	NA	NA	1411.0	7.22	15.31	229	NA	NA	NA	-	-	20.13
209	NA	NA	NA	1362.0	5.91	22.94	230	NA	NA	NA	-	-	19.1
							Average:	**470.31**	**2.64**	**1.0**	**697.99**	**1.22**	**24.76**

APPENDIX C: BRANCH-AND-BOUND RESULTS

Table C.1. Column expressions.

Column	Description
ID	Unique ID of the problem instance.
LB	Lower bound value. “Infeas” denotes that the algorithm detected the infeasibility of the problem.
LB Gap(%)	Lower bound relative percentage gap. A “-” sign indicates that the feasibility of the problem is unknown whereas “Infeas” denotes the infeasibility of the problem.
UB	Upper bound value. A “-” sign indicates that no feasible solution is found by the algorithm. “Infeas” denotes that the algorithm detected the infeasibility of the problem.
UB Gap(%)	Upper bound relative percentage gap. A “-” sign indicates that the feasibility of the problem is unknown whereas “Infeas” denotes the infeasibility of the problem.
CPU(s)	CPU time in seconds.

Table C.2. Effect of valid inequalities on the ZPK instance class.

	STRONG LP		CLIQUE LR		CLIQUE + OC LR			STRONG LP		CLIQUE LR		CLIQUE + OC LR	
ID	LB	LB Gap(%)	LB	LB Gap(%)	LB	LB Gap(%)	ID	LB	LB Gap(%)	LB	LB Gap(%)	LB	LB Gap(%)
1	613.9	13.29	705.5	0.35	705.5	0.35	26	13186.8	Infeas	14464.1	Infeas	15741.3	Infeas
2	658.57	14.47	769.09	0.12	769.09	0.12	27	13222.3	Infeas	16809.6	Infeas	17587.3	Infeas
3	726.15	20.81	873.69	4.72	873.69	4.72	28	13196.3	36.3	20716.0	0.0	20716.0	0.0
4	943.0	28.78	1035.18	21.81	1122.79	15.2	29	13181.3	26.87	18025.0	0.0	18025.0	0.0
5	997.0	39.06	1635.01	0.06	1635.01	0.06	30	13222.3	36.63	20864.0	0.0	20864.0	0.0
6	997.0	51.2	2042.21	0.04	2042.21	0.04	31	18417.3	14.26	19970.5	7.03	20044.6	6.68
7	997.0	57.36	2337.01	0.04	2337.01	0.04	32	20404.1	-	22878.9	-	24909.9	-
8	3445.59	14.73	4037.03	0.1	4037.03	0.1	33	20536.6	-	27479.1	-	28444.0	-
9	4637.52	18.04	5030.34	11.09	5119.39	9.52	34	20536.6	Infeas	31571.0	Infeas	31561.1	Infeas
10	4951.2	Infeas	5365.17	Infeas	5772.41	Infeas	35	20536.6	48.52	39894.0	0.0	39894.0	0.0
11	4996.23	32.79	7434.0	0.0	7434.0	0.0	36	20536.6	45.48	37671.0	0.0	37671.0	0.0
12	4996.23	37.3	7968.0	0.0	7968.0	0.0	37	20440.1	47.32	38798.0	0.0	38798.0	0.0
13	4996.23	38.82	8166.0	0.0	8166.0	0.0	38	43721.0	0.0	43721.0	0.0	43721.0	0.0
14	3461.6	19.03	4260.7	0.33	4260.7	0.33	39	44259.0	0.02	44266.0	0.0	44266.0	0.0
15	4845.66	19.2	5356.06	10.69	5480.33	8.62	40	42910.1	0.37	43071.0	0.0	43071.0	0.0
16	5012.66	33.37	5828.2	22.53	6113.69	18.73	41	31400.9	Infeas	34443.8	Infeas	36471.4	Infeas
17	5204.57	-	6752.78	-	6885.55	-	42	31672.3	26.56	43125.0	0.0	43125.0	0.0
18	5234.69	Infeas	8776.84	Infeas	8776.84	Infeas	43	31757.8	24.91	42292.0	0.0	42292.0	0.0
19	5234.69	58.63	11442.1	9.56	11442.1	9.56	44	31698.8	28.14	44114.0	0.0	44114.0	0.0
20	5234.69	53.39	11226.9	0.05	11226.9	0.05	45	43317.4	-	45505.5	-	48645.6	-
21	5234.69	54.41	11480.0	0.01	11480.0	0.01	46	43878.4	-	53909.0	-	56581.1	-
22	17652.1	0.43	17727.2	0.0	17727.2	0.0	47	43976.9	Infeas	63561.1	Infeas	64454.9	Infeas
23	18595.0	0.12	18616.1	0.0	18616.1	0.0	48	43976.9	38.55	71562.0	0.0	71562.0	0.0
24	19033.5	0.56	19140.0	0.0	19140.0	0.0	49	43976.9	42.4	76345.0	0.0	76345.0	0.0
25	12149.5	12.03	12544.2	9.17	12660.5	8.33	50	43976.9	44.25	78880.0	0.0	78880.0	0.0
							Average:	**17375.78**	**28.38**	**23489.72**	**2.57**	**23791.54**	**2.17**

Table C.3. Effect of valid inequalities on the CCPR instance class.

	STRONG LP		CLIQUE LR		CLIQUE + OC LR			STRONG LP		CLIQUE LR		CLIQUE + OC LR	
ID	LB	LB Gap(%)	LB	LB Gap(%)	LB	LB Gap(%)	ID	LB	LB Gap(%)	LB	LB Gap(%)	LB	LB Gap(%)
51	332.35	4.22	346.1	0.26	346.1	0.26	74	308.67	6.18	328.0	0.31	328.0	0.31
52	365.75	5.98	389.0	0.0	389.0	0.0	75	307.83	5.57	325.0	0.31	325.0	0.31
53	337.58	4.37	353.0	0.0	353.0	0.0	76	330.92	5.18	348.05	0.27	348.05	0.27
54	341.21	1.38	346.0	0.0	346.0	0.0	77	354.77	7.85	372.65	3.21	372.65	3.21
55	328.33	2.28	336.0	0.0	336.0	0.0	78	325.0	2.99	332.16	0.85	332.16	0.85
56	366.08	3.92	379.66	0.35	379.66	0.35	79	326.79	6.09	338.0	2.87	343.43	1.31
57	361.42	7.33	381.49	2.18	381.49	2.18	80	347.0	2.8	356.03	0.27	356.03	0.27
58	353.42	5.0	371.52	0.13	371.52	0.13	81	281.0	0.35	282.0	0.0	282.0	0.0
59	346.5	2.94	356.31	0.19	356.31	0.19	82	284.17	3.34	293.98	0.01	293.98	0.01
60	387.58	4.54	405.56	0.11	405.56	0.11	83	283.04	0.34	283.1	0.32	283.1	0.32
61	374.21	2.8	384.19	0.21	384.19	0.21	84	267.42	4.83	280.17	0.29	280.17	0.29
62	412.21	4.58	431.08	0.21	431.08	0.21	85	289.04	1.01	291.1	0.31	291.1	0.31
63	422.5	7.75	431.93	5.69	439.8	3.97	86	315.04	1.86	320.05	0.29	320.05	0.29
64	383.38	4.16	398.34	0.41	398.34	0.41	87	310.21	2.14	316.02	0.31	316.02	0.31
65	400.62	4.61	408.98	2.62	408.98	2.62	88	283.04	0.34	283.3	0.25	283.3	0.25
66	300.33	3.43	310.21	0.26	310.21	0.26	89	304.25	2.17	310.05	0.3	310.05	0.3
67	288.54	5.71	306.0	0.0	306.0	0.0	90	283.38	2.28	289.3	0.24	289.3	0.24
68	286.58	4.15	299.0	0.0	299.0	0.0	91	308.17	6.33	322.0	2.13	322.85	1.87
69	288.29	2.93	296.91	0.03	296.91	0.03	92	323.0	4.72	329.06	2.93	330.09	2.63
70	314.96	0.96	317.82	0.06	317.82	0.06	93	353.9	3.83	362.24	1.57	363.77	1.15
71	293.38	3.81	304.52	0.16	304.52	0.16	94	304.09	2.22	307.71	1.06	307.71	1.06
72	332.25	1.99	338.3	0.21	338.3	0.21	95	314.5	2.02	320.01	0.31	320.07	0.29
73	333.42	3.08	343.13	0.25	343.13	0.25	96	573.92	7.28	618.26	0.12	618.26	0.12

Table C.3. Effect of valid inequalities on the CCPR instance class (cont.).

	STRONG LP		CLIQUE LR		CLIQUE + OC LR			STRONG LP		CLIQUE LR		CLIQUE + OC LR	
ID	LB	LB Gap(%)	LB	LB Gap(%)	LB	LB Gap(%)	ID	LB	LB Gap(%)	LB	LB Gap(%)	LB	LB Gap(%)
97	593.43	1.75	603.71	0.05	603.71	0.05	120	619.59	3.64	636.75	0.97	637.38	0.87
98	630.04	0.62	633.18	0.13	633.18	0.13	121	617.02	15.01	646.41	10.96	654.77	9.81
99	600.22	2.56	615.07	0.15	615.07	0.15	122	627.5	15.66	662.6	10.94	663.18	10.86
100	577.22	2.99	594.33	0.11	594.33	0.11	123	640.26	17.7	674.78	13.27	678.34	12.81
101	661.86	2.38	669.44	1.26	671.08	1.02	124	592.5	15.48	621.87	11.29	629.28	10.23
102	647.82	4.87	667.56	1.97	670.91	1.48	125	654.81	14.29	685.82	10.23	685.82	10.23
103	662.68	6.53	683.68	3.57	686.09	3.23	126	532.29	2.87	547.04	0.18	547.04	0.18
104	624.62	2.25	635.2	0.59	635.2	0.59	127	514.26	2.97	529.01	0.19	529.01	0.19
105	645.58	5.2	661.66	2.84	663.86	2.52	128	541.2	1.42	548.04	0.18	548.04	0.18
106	656.52	19.54	712.8	12.65	719.62	11.81	129	528.66	2.1	539.1	0.17	539.1	0.17
107	697.5	16.47	738.02	11.61	747.57	10.47	130	527.22	2.37	539.08	0.17	539.08	0.17
108	659.0	18.64	697.36	13.91	702.96	13.21	131	570.9	3.89	584.94	1.52	584.94	1.52
109	671.52	19.29	717.72	13.74	725.47	12.8	132	548.25	5.31	562.95	2.77	566.28	2.2
110	690.56	10.2	713.9	7.16	714.77	7.05	133	571.46	2.98	583.96	0.85	583.96	0.85
111	545.45	4.31	569.28	0.13	569.28	0.13	134	559.54	3.03	566.35	1.85	568.48	1.48
112	540.37	3.68	560.01	0.18	560.01	0.18	135	572.43	3.31	582.5	1.61	583.52	1.43
113	565.18	1.36	572.04	0.17	572.04	0.17	136	569.5	14.49	595.66	10.56	595.66	10.56
114	551.14	1.58	560.0	0.0	560.0	0.0	137	564.5	11.52	587.54	7.91	588.07	7.83
115	540.14	1.61	548.0	0.18	548.0	0.18	138	587.75	13.31	613.46	9.52	614.47	9.37
116	585.27	4.37	599.98	1.96	599.98	1.96	139	589.02	13.63	616.47	9.61	616.94	9.54
117	587.54	4.46	598.45	2.69	600.99	2.28	140	586.0	10.81	599.9	8.69	605.61	7.82
118	565.63	3.64	578.39	1.47	579.2	1.33	141	838.31	3.42	867.87	0.01	867.87	0.01
119	603.58	4.8	611.49	3.55	617.68	2.57	142	858.12	1.48	870.06	0.11	870.06	0.11

Table C.3. Effect of valid inequalities on the CCPR instance class (cont.).

	STRONG LP		CLIQUE LR		CLIQUE + OC LR			STRONG LP		CLIQUE LR		CLIQUE + OC LR	
ID	LB	LB Gap(%)	LB	LB Gap(%)	LB	LB Gap(%)	ID	LB	LB Gap(%)	LB	LB Gap(%)	LB	LB Gap(%)
143	828.1	1.18	837.03	0.12	837.03	0.12	166	866.5	39.78	920.36	36.04	922.92	35.86
144	830.22	2.9	854.77	0.03	854.77	0.03	167	834.0	-	882.12	-	882.0	-
145	833.27	2.77	856.01	0.12	856.01	0.12	168	839.01	-	884.82	-	886.06	-
146	939.5	10.27	964.88	7.84	984.74	5.95	169	855.5	-	919.5	-	919.69	-
147	933.53	9.89	956.03	7.72	968.39	6.53	170	876.51	-	930.18	-	931.34	-
148	904.0	12.66	930.6	10.09	945.31	8.67	171	776.03	1.39	786.49	0.06	786.49	0.06
149	908.0	8.47	935.54	5.69	942.1	5.03	172	771.07	1.77	784.04	0.12	784.04	0.12
150	891.53	10.31	918.87	7.56	921.54	7.29	173	773.11	1.26	782.02	0.12	782.02	0.12
151	911.05	-	1001.18	-	1005.73	-	174	772.11	1.52	783.01	0.13	783.01	0.13
152	940.04	-	1020.82	-	1022.35	-	175	782.15	1.86	796.01	0.12	796.01	0.12
153	895.0	-	983.14	-	988.39	-	176	811.03	5.03	823.92	3.52	828.16	3.03
154	885.0	-	956.98	-	958.62	-	177	796.0	6.13	804.55	5.12	808.68	4.64
155	908.03	-	994.68	-	994.68	-	178	806.5	7.93	816.37	6.81	823.49	5.99
156	779.15	2.36	797.06	0.12	797.06	0.12	179	803.01	6.84	813.19	5.66	818.99	4.99
157	801.14	2.42	819.9	0.13	819.9	0.13	180	795.55	7.28	811.46	5.42	815.18	4.99
158	798.19	2.18	815.04	0.12	815.04	0.12	181	812.5	40.26	853.88	37.21	855.8	37.07
159	805.19	1.81	819.0	0.12	819.0	0.12	182	800.03	36.46	837.46	33.48	838.14	33.43
160	800.15	1.82	814.49	0.06	814.49	0.06	183	822.5	31.11	865.0	27.55	865.79	27.49
161	824.55	7.97	838.5	6.42	849.63	5.18	184	797.0	40.74	837.85	37.71	839.12	37.61
162	852.01	8.78	871.22	6.72	874.54	6.37	185	800.03	40.39	841.86	37.27	841.78	37.27
163	825.01	7.41	841.44	5.56	847.91	4.84	186	1096.7	1.99	1118.02	0.09	1118.02	0.09
164	837.0	7.72	850.82	6.19	855.62	5.67	187	1114.64	1.97	1136.0	0.09	1136.0	0.09
165	843.53	5.86	854.33	4.65	858.8	4.15	188	1076.26	3.3	1112.26	0.07	1112.26	0.07

Table C.3. Effect of valid inequalities on the CCPR instance class (cont.).

	STRONG LP		CLIQUE LR		CLIQUE + OC LR			STRONG LP		CLIQUE LR		CLIQUE + OC LR	
ID	LB	LB Gap(%)	LB	LB Gap(%)	LB	LB Gap(%)	ID	LB	LB Gap(%)	LB	LB Gap(%)	LB	LB Gap(%)
189	1086.27	2.14	1109.31	0.06	1109.31	0.06	210	1084.51	16.38	1104.03	14.88	1114.39	14.08
190	1062.69	2.51	1088.86	0.1	1088.86	0.1	211	1077.5	-	1150.18	-	1148.9	-
191	1166.02	20.52	1205.4	17.83	1224.99	16.5	212	1084.0	-	1152.52	-	1151.28	-
192	1136.51	23.78	1169.76	21.55	1192.62	20.01	213	1084.53	-	1165.2	-	1165.03	-
193	1131.0	24.4	1169.58	21.82	1188.11	20.58	214	1099.5	-	1169.19	-	1169.19	-
194	1174.01	18.53	1218.34	15.45	1234.13	14.36	215	1085.5	-	1145.99	-	1145.98	-
195	1167.0	24.81	1213.61	21.8	1226.95	20.94	216	1023.07	0.77	1030.02	0.1	1030.02	0.1
196	1122.02	-	1229.36	-	1229.36	-	217	1028.05	0.77	1034.72	0.12	1034.72	0.12
197	1144.5	-	1261.39	-	1261.39	-	218	1016.12	0.77	1023.03	0.09	1023.03	0.09
198	1167.03	-	1288.78	-	1288.78	-	219	1018.05	0.68	1024.01	0.1	1024.01	0.1
199	1134.57	-	1244.74	-	1245.0	-	220	1018.09	0.96	1027.02	0.1	1027.02	0.1
200	1153.5	-	1268.16	-	1268.16	-	221	1058.5	12.45	1079.86	10.68	1081.69	10.53
201	1054.12	2.31	1077.8	0.11	1077.8	0.11	222	1031.0	10.89	1044.13	9.76	1049.64	9.28
202	1038.09	1.7	1054.82	0.11	1054.82	0.11	223	1047.5	12.85	1067.81	11.16	1073.36	10.7
203	1042.1	1.6	1058.03	0.09	1058.03	0.09	224	1041.88	14.53	1064.37	12.68	1068.92	12.31
204	1030.09	1.52	1045.0	0.1	1045.0	0.1	225	1044.0	11.68	1066.25	9.79	1069.5	9.52
205	1040.24	2.96	1071.01	0.09	1071.01	0.09	226	1037.0	-	1086.72	-	1086.72	-
206	1090.51	16.37	1113.02	14.65	1123.31	13.86	227	1064.5	-	1115.75	-	1116.74	-
207	1085.13	15.95	1108.32	14.15	1121.77	13.11	228	1049.5	-	1099.88	-	1099.88	-
208	1077.04	18.16	1107.3	15.86	1117.94	15.05	229	1048.52	-	1104.32	-	1104.32	-
209	1086.0	15.55	1110.94	13.61	1119.63	12.94	230	1053.0	-	1105.4	-	1104.72	-
							Average:	**709.88**	**7.42**	**735.52**	**4.86**	**737.73**	**4.6**

Table C.4. Performances of the exact solution algorithms on the ZPK instance class.

	B&B					Gurobi					B&C				
ID	LB	LB Gap(%)	UB	UB Gap(%)	CPU(s)	LB	LB Gap(%)	UB	UB Gap(%)	CPU(s)	LB	LB Gap(%)	UB	UB Gap(%)	CPU(s)
1	708.0	0.0	708.0	0.0	30.79	708.0	0.0	708.0	0.0	1.65	708.0	0.0	708.0	0.0	0.2
2	770.0	0.0	770.0	0.0	5.04	770.0	0.0	770.0	0.0	1.76	770.0	0.0	770.0	0.0	0.5
3	917.0	0.0	917.0	0.0	12.43	917.0	0.0	917.0	0.0	6.22	917.0	0.0	917.0	0.0	1.8
4	1324.0	0.0	1324.0	0.0	58.49	1324.0	0.0	1324.0	0.0	13.4	1324.0	0.0	1324.0	0.0	7.4
8	4041.0	0.0	4041.0	0.0	90.95	4041.0	0.0	4041.0	0.0	7.73	4041.0	0.0	4041.0	0.0	4.6
9	5658.0	0.0	5658.0	0.0	840.87	5658.0	0.0	5658.0	0.0	229.15	5658.0	0.0	5658.0	0.0	178.5
10	5772.41	Infeas	-	Infeas	5006.34	Infeas	Infeas	Infeas	Infeas	420.96	6635.4	Infeas	-	Infeas	5010.0
14	4275.0	0.0	4275.0	0.0	164.3	4275.0	0.0	4275.0	0.0	5.72	4275.0	0.0	4275.0	0.0	11.5
15	5480.33	8.62	6173.0	2.93	5003.86	5976.0	0.35	6000.0	0.05	5000.05	5997.0	0.0	5997.0	0.0	1239.4
16	6113.69	18.73	9507.0	26.37	5007.77	6812.0	9.45	7570.0	0.62	5000.06	6707.8	10.84	8049.0	6.99	5010.1
17	6885.55	-	-	-	5003.44	8153.0	-	-	-	5000.1	7729.3	-	-	-	5010.0
18	8776.84	Infeas	-	Infeas	5005.06	Infeas	Infeas	Infeas	Infeas	356.65	10560.2	Infeas	-	Infeas	5010.0
25	12660.5	8.33	14488.0	4.9	5012.02	13361.0	3.26	13811.0	0.0	5000.05	13171.2	4.63	14086.0	1.99	5010.0
26	15741.3	Infeas	-	Infeas	5007.61	Infeas	Infeas	Infeas	Infeas	3118.0	17595.0	Infeas	-	Infeas	5010.0
27	17587.3	Infeas	-	Infeas	5002.03	Infeas	Infeas	Infeas	Infeas	18.46	Infeas	Infeas	Infeas	Infeas	16.4
31	20044.6	6.68	22632.0	5.36	5013.95	20784.0	3.24	21503.0	0.11	5000.07	20941.5	2.51	21553.0	0.34	5010.1
32	24909.9	-	-	-	5008.23	26651.0	-	-	-	5000.09	26526.7	-	-	-	5010.1
33	28444.0	-	-	-	5006.59	32406.0	-	-	-	5000.1	30634.2	-	-	-	5010.0
34	31561.1	Infeas	-	Infeas	5010.39	Infeas	Infeas	Infeas	Infeas	1614.17	36900.2	Infeas	-	Infeas	5010.0
41	36471.4	Infeas	-	Infeas	5007.31	Infeas	Infeas	Infeas	Infeas	68.66	Infeas	Infeas	Infeas	Infeas	2911.1
45	48645.6	-	-	-	5003.54	51617.0	-	-	-	5000.08	51398.4	-	-	-	5010.0
46	56581.1	-	-	-	5001.72	62690.0	-	-	-	5000.06	61878.9	-	-	-	5010.0
47	64454.9	Infeas	-	Infeas	5006.58	Infeas	Infeas	Infeas	Infeas	183.56	Infeas	Infeas	Infeas	Infeas	1820.0
Average:	**17731.46**	**3.85**	**6408.45**	**3.6**	**3535.21**	**15383.94**	**1.48**	**6052.45**	**0.07**	**2219.42**	**15718.44**	**1.63**	**6125.27**	**0.85**	**2883.12**

Table C.5. Performances of the exact solution algorithms on the CCPR instance class.

	B&B					Gurobi					B&C				
ID	LB	LB Gap(%)	UB	UB Gap(%)	CPU(s)	LB	LB Gap(%)	UB	UB Gap(%)	CPU(s)	LB	LB Gap(%)	UB	UB Gap(%)	CPU(s)
51	347.0	0.0	347.0	0.0	0.03	347.0	0.0	347.0	0.0	0.07	347.0	0.0	347.0	0.0	0.0
52	389.0	0.0	389.0	0.0	0.02	389.0	0.0	389.0	0.0	0.05	389.0	0.0	389.0	0.0	0.0
53	353.0	0.0	353.0	0.0	0.02	353.0	0.0	353.0	0.0	0.06	353.0	0.0	353.0	0.0	0.0
54	346.0	0.0	346.0	0.0	0.02	346.0	0.0	346.0	0.0	0.03	346.0	0.0	346.0	0.0	0.0
55	336.0	0.0	336.0	0.0	0.02	336.0	0.0	336.0	0.0	0.05	336.0	0.0	336.0	0.0	0.0
56	381.0	0.0	381.0	0.0	0.69	381.0	0.0	381.0	0.0	0.29	381.0	0.0	381.0	0.0	0.0
57	390.0	0.0	390.0	0.0	0.8	390.0	0.0	390.0	0.0	0.22	390.0	0.0	390.0	0.0	0.1
58	372.0	0.0	372.0	0.0	0.04	372.0	0.0	372.0	0.0	0.05	372.0	0.0	372.0	0.0	0.0
59	357.0	0.0	357.0	0.0	0.07	357.0	0.0	357.0	0.0	0.07	357.0	0.0	357.0	0.0	0.0
60	406.0	0.0	406.0	0.0	0.06	406.0	0.0	406.0	0.0	0.14	406.0	0.0	406.0	0.0	0.0
61	385.0	0.0	385.0	0.0	0.15	385.0	0.0	385.0	0.0	0.04	385.0	0.0	385.0	0.0	0.0
62	432.0	0.0	432.0	0.0	0.06	432.0	0.0	432.0	0.0	0.04	432.0	0.0	432.0	0.0	0.0
63	458.0	0.0	458.0	0.0	0.62	458.0	0.0	458.0	0.0	0.35	458.0	0.0	458.0	0.0	0.3
64	400.0	0.0	400.0	0.0	0.63	400.0	0.0	400.0	0.0	0.08	400.0	0.0	400.0	0.0	0.0
65	420.0	0.0	420.0	0.0	0.56	420.0	0.0	420.0	0.0	0.23	420.0	0.0	420.0	0.0	0.0
66	311.0	0.0	311.0	0.0	0.04	311.0	0.0	311.0	0.0	0.17	311.0	0.0	311.0	0.0	0.0
67	306.0	0.0	306.0	0.0	0.02	306.0	0.0	306.0	0.0	0.03	306.0	0.0	306.0	0.0	0.0
68	299.0	0.0	299.0	0.0	0.03	299.0	0.0	299.0	0.0	0.14	299.0	0.0	299.0	0.0	0.0
69	297.0	0.0	297.0	0.0	0.03	297.0	0.0	297.0	0.0	0.12	297.0	0.0	297.0	0.0	0.0
70	318.0	0.0	318.0	0.0	0.04	318.0	0.0	318.0	0.0	0.11	318.0	0.0	318.0	0.0	0.0
71	305.0	0.0	305.0	0.0	0.07	305.0	0.0	305.0	0.0	0.22	305.0	0.0	305.0	0.0	0.0
72	339.0	0.0	339.0	0.0	0.07	339.0	0.0	339.0	0.0	0.05	339.0	0.0	339.0	0.0	0.0
73	344.0	0.0	344.0	0.0	0.08	344.0	0.0	344.0	0.0	0.06	344.0	0.0	344.0	0.0	0.0

Table C.5. Performances of the exact solution algorithms on the CCPR instance class (cont.).

	B&B					Gurobi					B&C				
ID	LB	LB Gap(%)	UB	UB Gap(%)	CPU(s)	LB	LB Gap(%)	UB	UB Gap(%)	CPU(s)	LB	LB Gap(%)	UB	UB Gap(%)	CPU(s)
74	329.0	0.0	329.0	0.0	1.22	329.0	0.0	329.0	0.0	0.17	329.0	0.0	329.0	0.0	0.0
75	326.0	0.0	326.0	0.0	0.79	326.0	0.0	326.0	0.0	0.34	326.0	0.0	326.0	0.0	0.0
76	349.0	0.0	349.0	0.0	0.28	349.0	0.0	349.0	0.0	0.34	349.0	0.0	349.0	0.0	0.0
77	385.0	0.0	385.0	0.0	1.42	385.0	0.0	385.0	0.0	0.59	385.0	0.0	385.0	0.0	0.5
78	335.0	0.0	335.0	0.0	1.28	335.0	0.0	335.0	0.0	0.07	335.0	0.0	335.0	0.0	0.0
79	348.0	0.0	348.0	0.0	1.23	348.0	0.0	348.0	0.0	0.51	348.0	0.0	348.0	0.0	0.1
80	357.0	0.0	357.0	0.0	0.25	357.0	0.0	357.0	0.0	0.15	357.0	0.0	357.0	0.0	0.0
81	282.0	0.0	282.0	0.0	0.02	282.0	0.0	282.0	0.0	0.04	282.0	0.0	282.0	0.0	0.0
82	294.0	0.0	294.0	0.0	0.03	294.0	0.0	294.0	0.0	0.18	294.0	0.0	294.0	0.0	0.0
83	284.0	0.0	284.0	0.0	0.05	284.0	0.0	284.0	0.0	0.06	284.0	0.0	284.0	0.0	0.0
84	281.0	0.0	281.0	0.0	0.03	281.0	0.0	281.0	0.0	0.32	281.0	0.0	281.0	0.0	0.0
85	292.0	0.0	292.0	0.0	0.03	292.0	0.0	292.0	0.0	0.07	292.0	0.0	292.0	0.0	0.0
86	321.0	0.0	321.0	0.0	0.23	321.0	0.0	321.0	0.0	0.38	321.0	0.0	321.0	0.0	0.0
87	317.0	0.0	317.0	0.0	0.24	317.0	0.0	317.0	0.0	0.12	317.0	0.0	317.0	0.0	0.0
88	284.0	0.0	284.0	0.0	0.08	284.0	0.0	284.0	0.0	0.06	284.0	0.0	284.0	0.0	0.0
89	311.0	0.0	311.0	0.0	0.15	311.0	0.0	311.0	0.0	0.27	311.0	0.0	311.0	0.0	0.0
90	290.0	0.0	290.0	0.0	0.05	290.0	0.0	290.0	0.0	0.11	290.0	0.0	290.0	0.0	0.0
91	329.0	0.0	329.0	0.0	1.73	329.0	0.0	329.0	0.0	1.34	329.0	0.0	329.0	0.0	0.1
92	339.0	0.0	339.0	0.0	1.75	339.0	0.0	339.0	0.0	0.9	339.0	0.0	339.0	0.0	0.5
93	368.0	0.0	368.0	0.0	1.28	368.0	0.0	368.0	0.0	0.87	368.0	0.0	368.0	0.0	0.4
94	311.0	0.0	311.0	0.0	2.13	311.0	0.0	311.0	0.0	0.36	311.0	0.0	311.0	0.0	0.0
95	321.0	0.0	321.0	0.0	0.46	321.0	0.0	321.0	0.0	0.58	321.0	0.0	321.0	0.0	0.0
96	619.0	0.0	619.0	0.0	0.75	619.0	0.0	619.0	0.0	0.77	619.0	0.0	619.0	0.0	0.0

Table C.5. Performances of the exact solution algorithms on the CCPR instance class (cont.).

	B&B					Gurobi					B&C				
ID	LB	LB Gap(%)	UB	UB Gap(%)	CPU(s)	LB	LB Gap(%)	UB	UB Gap(%)	CPU(s)	LB	LB Gap(%)	UB	UB Gap(%)	CPU(s)
97	604.0	0.0	604.0	0.0	0.55	604.0	0.0	604.0	0.0	1.08	604.0	0.0	604.0	0.0	0.0
98	634.0	0.0	634.0	0.0	0.42	634.0	0.0	634.0	0.0	0.12	634.0	0.0	634.0	0.0	0.0
99	616.0	0.0	616.0	0.0	0.57	616.0	0.0	616.0	0.0	0.64	616.0	0.0	616.0	0.0	0.1
100	595.0	0.0	595.0	0.0	0.34	595.0	0.0	595.0	0.0	1.1	595.0	0.0	595.0	0.0	0.0
101	678.0	0.0	678.0	0.0	17.33	678.0	0.0	678.0	0.0	5.0	678.0	0.0	678.0	0.0	1.4
102	681.0	0.0	681.0	0.0	17.61	681.0	0.0	681.0	0.0	9.21	681.0	0.0	681.0	0.0	3.2
103	709.0	0.0	709.0	0.0	50.4	709.0	0.0	709.0	0.0	18.54	709.0	0.0	709.0	0.0	6.3
104	639.0	0.0	639.0	0.0	19.62	639.0	0.0	639.0	0.0	1.42	639.0	0.0	639.0	0.0	1.5
105	681.0	0.0	681.0	0.0	29.3	681.0	0.0	681.0	0.0	12.16	681.0	0.0	681.0	0.0	3.8
106	719.62	11.81	816.0	0.0	5000.72	816.0	0.0	816.0	0.0	2447.63	791.2	3.04	833.0	2.08	5010.1
107	835.0	0.0	835.0	0.0	3973.03	835.0	0.0	835.0	0.0	587.52	835.0	0.0	835.0	0.0	1938.7
108	702.96	13.21	810.0	0.0	5000.68	810.0	0.0	810.0	0.0	2766.4	773.23	4.54	840.0	3.7	5010.1
109	725.47	12.8	837.0	0.6	5000.43	832.0	0.0	832.0	0.0	360.24	820.02	1.44	836.0	0.48	5010.1
110	769.0	0.0	769.0	0.0	169.57	769.0	0.0	769.0	0.0	34.64	769.0	0.0	769.0	0.0	25.7
111	570.0	0.0	570.0	0.0	1.23	570.0	0.0	570.0	0.0	1.01	570.0	0.0	570.0	0.0	0.1
112	561.0	0.0	561.0	0.0	1.35	561.0	0.0	561.0	0.0	2.27	561.0	0.0	561.0	0.0	1.4
113	573.0	0.0	573.0	0.0	1.41	573.0	0.0	573.0	0.0	0.21	573.0	0.0	573.0	0.0	0.0
114	560.0	0.0	560.0	0.0	0.8	560.0	0.0	560.0	0.0	0.79	560.0	0.0	560.0	0.0	0.0
115	549.0	0.0	549.0	0.0	0.76	549.0	0.0	549.0	0.0	1.88	549.0	0.0	549.0	0.0	0.5
116	613.0	0.16	613.0	0.16	52.41	612.0	0.0	612.0	0.0	19.5	612.0	0.0	612.0	0.0	7.5
117	615.0	0.0	615.0	0.0	68.41	615.0	0.0	615.0	0.0	12.56	615.0	0.0	615.0	0.0	6.6
118	587.0	0.0	587.0	0.0	21.06	587.0	0.0	587.0	0.0	11.2	587.0	0.0	587.0	0.0	3.0
119	634.0	0.0	634.0	0.0	376.78	634.0	0.0	634.0	0.0	26.8	634.0	0.0	634.0	0.0	7.3

Table C.5. Performances of the exact solution algorithms on the CCPR instance class (cont.).

	B&B					Gurobi					B&C				
ID	LB	LB Gap(%)	UB	UB Gap(%)	CPU(s)	LB	LB Gap(%)	UB	UB Gap(%)	CPU(s)	LB	LB Gap(%)	UB	UB Gap(%)	CPU(s)
120	643.0	0.0	643.0	0.0	24.28	643.0	0.0	643.0	0.0	4.35	643.0	0.0	643.0	0.0	3.2
121	654.77	9.81	757.0	4.27	5001.39	707.0	2.62	726.0	0.0	5000.03	701.26	3.41	726.0	0.0	5010.1
122	663.18	10.86	765.0	2.82	5001.66	744.0	0.0	744.0	0.0	4683.39	719.45	3.3	770.0	3.49	5010.0
123	678.34	12.81	800.0	2.83	5001.58	732.0	5.91	778.0	0.0	5000.07	723.89	6.96	786.0	1.03	5010.0
124	629.28	10.23	710.0	1.28	5001.41	680.0	3.0	701.0	0.0	5000.22	669.84	4.45	711.0	1.43	5010.0
125	685.82	10.23	777.0	1.7	5001.81	745.0	2.49	766.0	0.26	5000.18	737.31	3.49	764.0	0.0	5010.0
126	548.0	0.0	548.0	0.0	3.09	548.0	0.0	548.0	0.0	0.85	548.0	0.0	548.0	0.0	0.1
127	530.0	0.0	530.0	0.0	2.39	530.0	0.0	530.0	0.0	0.78	530.0	0.0	530.0	0.0	0.5
128	549.0	0.0	549.0	0.0	1.7	549.0	0.0	549.0	0.0	0.41	549.0	0.0	549.0	0.0	0.0
129	540.0	0.0	540.0	0.0	2.12	540.0	0.0	540.0	0.0	3.63	540.0	0.0	540.0	0.0	0.2
130	540.0	0.0	540.0	0.0	1.56	540.0	0.0	540.0	0.0	0.65	540.0	0.0	540.0	0.0	0.0
131	594.0	0.0	594.0	0.0	86.93	594.0	0.0	594.0	0.0	17.42	594.0	0.0	594.0	0.0	7.8
132	579.0	0.0	579.0	0.0	391.89	579.0	0.0	579.0	0.0	61.76	579.0	0.0	579.0	0.0	13.8
133	589.0	0.0	589.0	0.0	42.04	589.0	0.0	589.0	0.0	3.85	589.0	0.0	589.0	0.0	3.0
134	577.0	0.0	577.0	0.0	72.32	577.0	0.0	577.0	0.0	13.93	577.0	0.0	577.0	0.0	7.5
135	592.0	0.0	592.0	0.0	172.06	592.0	0.0	592.0	0.0	10.26	592.0	0.0	592.0	0.0	6.0
136	595.66	10.56	693.0	4.05	5001.22	639.0	4.05	666.0	0.0	5000.09	631.43	5.19	678.0	1.8	5010.0
137	588.07	7.83	667.0	4.55	5002.72	631.0	1.1	638.0	0.0	5000.14	626.72	1.77	651.0	2.04	5010.0
138	614.47	9.37	689.0	1.62	5001.75	662.0	2.36	678.0	0.0	5000.19	658.38	2.89	689.0	1.62	5010.0
139	616.94	9.54	710.0	4.11	5001.39	671.0	1.61	687.0	0.73	5000.2	662.22	2.9	682.0	0.0	5010.1
140	605.61	7.82	686.0	4.41	5001.13	657.0	0.0	657.0	0.0	3109.13	641.31	2.39	674.0	2.59	5010.0
141	868.0	0.0	868.0	0.0	23.56	868.0	0.0	868.0	0.0	2.17	868.0	0.0	868.0	0.0	0.7
142	871.0	0.0	871.0	0.0	8.66	871.0	0.0	871.0	0.0	5.76	871.0	0.0	871.0	0.0	3.0

Table C.5. Performances of the exact solution algorithms on the CCPR instance class (cont.).

	B&B					Gurobi					B&C				
ID	LB	LB Gap(%)	UB	UB Gap(%)	CPU(s)	LB	LB Gap(%)	UB	UB Gap(%)	CPU(s)	LB	LB Gap(%)	UB	UB Gap(%)	CPU(s)
143	838.0	0.0	838.0	0.0	6.84	838.0	0.0	838.0	0.0	3.32	838.0	0.0	838.0	0.0	0.3
144	855.0	0.0	855.0	0.0	5.19	855.0	0.0	855.0	0.0	4.91	855.0	0.0	855.0	0.0	4.4
145	857.0	0.0	857.0	0.0	8.66	857.0	0.0	857.0	0.0	3.66	857.0	0.0	857.0	0.0	4.1
146	984.74	5.95	1072.0	2.39	5003.7	1030.0	1.62	1048.0	0.1	5000.07	1023.72	2.22	1047.0	0.0	5010.0
147	968.39	6.53	1084.0	4.63	5004.32	1016.0	1.93	1036.0	0.0	5000.18	1008.82	2.62	1069.0	3.19	5010.2
148	945.31	8.67	1080.0	4.35	5004.34	994.0	3.96	1035.0	0.0	5000.18	987.31	4.61	1040.0	0.48	5010.1
149	942.1	5.03	1012.0	2.02	5003.98	992.0	0.0	992.0	0.0	2682.85	985.64	0.64	998.0	0.6	5010.1
150	921.54	7.29	1022.0	2.82	5003.8	972.0	2.21	1000.0	0.6	5000.09	962.55	3.16	994.0	0.0	5010.1
151	1005.73	-	-	-	5001.56	1065.0	-	-	-	5000.1	1054.25	-	-	-	4647.3
152	1022.35	-	-	-	5001.19	1079.0	-	-	-	5000.22	1069.51	-	-	-	3483.6
153	988.39	-	-	-	5001.88	1056.0	-	-	-	5000.27	1040.97	-	-	-	5010.0
154	958.62	-	-	-	5001.22	1026.0	-	-	-	5000.14	1006.3	-	-	-	5010.0
155	994.68	-	-	-	5001.11	1074.0	-	-	-	5000.2	1046.43	-	-	-	3208.1
156	798.0	0.0	798.0	0.0	56.03	798.0	0.0	798.0	0.0	3.8	798.0	0.0	798.0	0.0	4.8
157	821.0	0.0	821.0	0.0	84.19	821.0	0.0	821.0	0.0	31.74	821.0	0.0	821.0	0.0	1.1
158	816.0	0.0	816.0	0.0	20.89	816.0	0.0	816.0	0.0	6.46	816.0	0.0	816.0	0.0	4.0
159	820.0	0.0	820.0	0.0	12.84	820.0	0.0	820.0	0.0	6.22	820.0	0.0	820.0	0.0	23.9
160	815.0	0.0	815.0	0.0	21.05	815.0	0.0	815.0	0.0	4.66	815.0	0.0	815.0	0.0	1.4
161	849.63	5.18	917.0	2.34	5001.3	877.0	2.12	896.0	0.0	5000.3	873.83	2.47	903.0	0.78	3463.3
162	874.54	6.37	944.0	1.07	5001.76	904.0	3.21	934.0	0.0	5000.25	901.81	3.45	953.0	2.03	4443.7
163	847.91	4.84	910.0	2.13	5001.16	876.0	1.68	891.0	0.0	5000.21	873.67	1.95	892.0	0.11	5010.0
164	855.62	5.67	937.0	3.31	5001.17	890.0	1.87	907.0	0.0	5000.21	885.57	2.36	915.0	0.88	4570.8
165	858.8	4.15	927.0	3.46	5001.64	889.0	0.78	900.0	0.45	5000.24	886.87	1.02	896.0	0.0	5010.0

Table C.5. Performances of the exact solution algorithms on the CCPR instance class (cont.).

	B&B					Gurobi					B&C				
ID	LB	LB Gap(%)	UB	UB Gap(%)	CPU(s)	LB	LB Gap(%)	UB	UB Gap(%)	CPU(s)	LB	LB Gap(%)	UB	UB Gap(%)	CPU(s)
166	922.92	35.86	1439.0	0.0	5001.76	966.0	32.87	-	-	5000.31	949.55	34.01	-	-	1305.3
167	882.0	-	-	-	5001.09	922.0	-	-	-	5000.43	907.8	-	-	-	1161.2
168	886.06	-	-	-	5001.99	921.0	-	-	-	5000.46	910.0	-	-	-	953.0
169	919.69	-	-	-	5001.36	963.0	-	-	-	5000.37	943.98	-	-	-	1224.2
170	931.34	-	-	-	5001.78	973.0	-	-	-	5000.4	956.31	-	-	-	962.3
171	787.0	0.0	787.0	0.0	80.94	787.0	0.0	787.0	0.0	2.64	787.0	0.0	787.0	0.0	2.1
172	785.0	0.0	785.0	0.0	24.9	785.0	0.0	785.0	0.0	2.67	785.0	0.0	785.0	0.0	5.4
173	783.0	0.0	783.0	0.0	25.31	783.0	0.0	783.0	0.0	1.97	783.0	0.0	783.0	0.0	0.0
174	784.0	0.0	784.0	0.0	25.38	784.0	0.0	784.0	0.0	2.15	784.0	0.0	784.0	0.0	3.3
175	797.0	0.0	797.0	0.0	44.4	797.0	0.0	797.0	0.0	4.33	797.0	0.0	797.0	0.0	5.7
176	828.16	3.03	879.0	2.93	5002.02	854.0	0.0	854.0	0.0	3552.27	846.69	0.86	867.0	1.52	4067.5
177	808.68	4.64	871.0	2.71	5001.77	832.0	1.89	848.0	0.0	5000.32	829.23	2.21	851.0	0.35	4573.5
178	823.49	5.99	903.0	3.08	5001.49	846.0	3.42	876.0	0.0	5000.35	841.54	3.93	892.0	1.83	2189.8
179	818.99	4.99	902.0	4.64	5001.0	844.0	2.09	862.0	0.0	5000.22	841.62	2.36	864.0	0.23	2866.0
180	815.18	4.99	892.0	3.96	5002.64	840.0	2.1	858.0	0.0	5000.35	835.04	2.68	882.0	2.8	1783.3
181	855.8	37.07	1360.0	0.0	5002.08	879.0	35.37	-	-	5000.46	868.72	36.12	-	-	1049.9
182	838.14	33.43	1259.0	0.0	5002.69	867.0	31.14	-	-	5000.61	853.45	32.21	-	-	1350.7
183	865.79	27.49	1385.0	16.0	5001.93	895.0	25.04	-	-	5000.57	884.67	25.91	1194.0	0.0	1522.3
184	839.12	37.61	1345.0	0.0	5001.06	867.0	35.54	-	-	5000.53	853.0	36.58	-	-	1466.3
185	841.78	37.27	1342.0	0.0	5002.69	865.0	35.54	-	-	5000.35	853.98	36.37	-	-	1461.4
186	1119.0	0.0	1119.0	0.0	137.01	1119.0	0.0	1119.0	0.0	30.68	1119.0	0.0	1119.0	0.0	43.7
187	1137.0	0.0	1137.0	0.0	36.88	1137.0	0.0	1137.0	0.0	53.91	1137.0	0.0	1137.0	0.0	11.8
188	1113.0	0.0	1113.0	0.0	86.38	1113.0	0.0	1113.0	0.0	17.51	1113.0	0.0	1113.0	0.0	71.8

Table C.5. Performances of the exact solution algorithms on the CCPR instance class (cont.).

	B&B					Gurobi					B&C				
ID	LB	LB Gap(%)	UB	UB Gap(%)	CPU(s)	LB	LB Gap(%)	UB	UB Gap(%)	CPU(s)	LB	LB Gap(%)	UB	UB Gap(%)	CPU(s)
189	1110.0	0.0	1110.0	0.0	95.26	1110.0	0.0	1110.0	0.0	18.35	1110.0	0.0	1110.0	0.0	48.6
190	1090.0	0.0	1090.0	0.0	243.43	1090.0	0.0	1090.0	0.0	9.77	1090.0	0.0	1090.0	0.0	35.8
191	1224.99	16.5	1521.0	3.68	5002.63	1257.0	14.31	1467.0	0.0	5000.39	1249.38	14.83	-	-	5010.0
192	1192.62	20.01	1534.0	2.88	5001.46	1223.0	17.97	1513.0	1.48	5000.3	1225.76	17.79	1491.0	0.0	5010.0
193	1188.11	20.58	1651.0	10.36	5001.94	1221.0	18.38	1496.0	0.0	5000.35	1215.0	18.78	1510.0	0.94	5010.0
194	1234.13	14.36	1551.0	7.63	5001.13	1267.0	12.07	1467.0	1.8	5000.2	1264.17	12.27	1441.0	0.0	5010.2
195	1226.95	20.94	1787.0	15.14	5001.72	1261.0	18.75	1552.0	0.0	5000.29	1257.27	18.99	1560.0	0.52	5010.1
196	1229.36	-	-	-	5001.69	1265.0	-	-	-	5000.39	1262.0	-	-	-	3006.9
197	1261.39	-	-	-	5001.59	1292.0	-	-	-	5000.35	1290.68	-	-	-	3371.9
198	1288.78	-	-	-	5001.14	1319.0	-	-	-	5000.35	1318.54	-	-	-	3684.2
199	1245.0	-	-	-	5001.27	1277.0	-	-	-	5000.36	1282.38	-	-	-	3939.1
200	1268.16	-	-	-	5002.98	1305.0	-	-	-	5000.32	1304.45	-	-	-	3103.7
201	1079.0	0.0	1079.0	0.0	475.7	1079.0	0.0	1079.0	0.0	67.58	1079.0	0.0	1079.0	0.0	248.6
202	1056.0	0.0	1056.0	0.0	493.14	1056.0	0.0	1056.0	0.0	8.58	1056.0	0.0	1056.0	0.0	113.4
203	1059.0	0.0	1059.0	0.0	97.91	1059.0	0.0	1059.0	0.0	3.54	1059.0	0.0	1059.0	0.0	47.9
204	1046.0	0.0	1046.0	0.0	140.56	1046.0	0.0	1046.0	0.0	5.09	1046.0	0.0	1046.0	0.0	195.2
205	1072.0	0.0	1072.0	0.0	134.19	1072.0	0.0	1072.0	0.0	48.73	1072.0	0.0	1072.0	0.0	249.6
206	1123.31	13.86	1350.0	3.53	5002.73	1150.0	11.81	1304.0	0.0	5000.35	1143.95	12.27	1374.0	5.37	3018.3
207	1121.77	13.11	1370.0	6.12	5001.56	1150.0	10.92	1313.0	1.7	5000.29	1143.61	11.42	1291.0	0.0	2144.6
208	1117.94	15.05	1426.0	8.36	5002.0	1138.0	13.53	1316.0	0.0	5000.56	1137.62	13.55	1344.0	2.13	3075.6
209	1119.63	12.94	1334.0	3.73	5001.65	1138.0	11.51	1287.0	0.08	5000.5	1136.9	11.59	1286.0	0.0	3523.3
210	1114.39	14.08	1350.0	4.09	5001.73	1141.0	12.03	1297.0	0.0	5000.29	1134.63	12.52	1370.0	5.63	2954.2
211	1148.9	-	-	-	5002.14	1168.0	-	-	-	5000.7	1164.44	-	-	-	4773.8

Table C.5. Performances of the exact solution algorithms on the CCPR instance class (cont.).

	B&B					Gurobi					B&C				
ID	LB	LB Gap(%)	UB	UB Gap(%)	CPU(s)	LB	LB Gap(%)	UB	UB Gap(%)	CPU(s)	LB	LB Gap(%)	UB	UB Gap(%)	CPU(s)
212	1151.28	-	-	-	5001.99	1171.0	-	-	-	5000.64	1168.2	-	-	-	5010.0
213	1165.03	-	-	-	5001.66	1185.0	-	-	-	5000.47	1180.02	-	-	-	5010.0
214	1169.19	-	-	-	5001.76	1188.0	-	-	-	5000.91	1183.53	-	-	-	5010.0
215	1145.98	-	-	-	5001.25	1166.0	-	-	-	5000.61	1159.25	-	-	-	5010.0
216	1031.0	0.0	1031.0	0.0	366.53	1031.0	0.0	1031.0	0.0	8.3	1031.0	0.0	1031.0	0.0	214.4
217	1036.0	0.0	1036.0	0.0	1466.64	1036.0	0.0	1036.0	0.0	3.95	1036.0	0.0	1036.0	0.0	42.8
218	1024.0	0.0	1024.0	0.0	406.8	1024.0	0.0	1024.0	0.0	4.78	1024.0	0.0	1024.0	0.0	21.8
219	1025.0	0.0	1025.0	0.0	298.45	1025.0	0.0	1025.0	0.0	4.05	1025.0	0.0	1025.0	0.0	27.4
220	1028.0	0.0	1028.0	0.0	388.07	1028.0	0.0	1028.0	0.0	65.76	1028.0	0.0	1028.0	0.0	151.0
221	1081.69	10.53	1272.0	5.21	5001.76	1100.0	9.02	1209.0	0.0	5000.54	1096.83	9.28	1234.0	2.07	1938.3
222	1049.64	9.28	1233.0	6.57	5002.29	1068.0	7.69	1157.0	0.0	5000.76	1065.64	7.9	1187.0	2.59	2160.3
223	1073.36	10.7	1252.0	4.16	5001.82	1091.0	9.23	1202.0	0.0	5000.73	1087.39	9.53	1213.0	0.92	3595.6
224	1068.92	12.31	1261.0	3.45	5002.05	1086.0	10.91	1219.0	0.0	5000.69	1081.26	11.3	1221.0	0.16	2411.8
225	1069.5	9.52	1248.0	5.58	5001.94	1090.0	7.78	1182.0	0.0	5000.69	1084.09	8.28	1245.0	5.33	2385.6
226	1086.72	-	-	-	5001.51	1103.0	-	-	-	5000.98	1098.61	-	-	-	5010.1
227	1116.74	-	-	-	5001.75	1129.0	-	-	-	5001.05	1126.27	-	-	-	5010.1
228	1099.88	-	-	-	5001.66	1115.0	-	-	-	5001.05	1111.27	-	-	-	5010.1
229	1104.32	-	-	-	5002.04	1122.0	-	-	-	5000.97	1114.58	-	-	-	5010.1
230	1104.72	-	-	-	5001.2	1118.0	-	-	-	5001.04	1114.07	-	-	-	5010.2
Average:	**740.19**	**4.12**	**747.47**	**1.18**	**2089.29**	**755.39**	**2.78**	**711.22**	**0.05**	**1949.68**	**752.26**	**3.04**	**712.87**	**0.4**	**1582.45**