INTEGER PROGRAMMING FORMULATIONS AND CUTTING PLANE ALGORITHMS FOR THE MAXIMUM SELECTIVE TREE PROBLEM

by

Ömer Burak Onar

B.S., Industrial Engineering, Boğaziçi University, 2018

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 Industrial Engineering

Boğaziçi University

2022

ACKNOWLEDGEMENTS

I would like to thank to all who helped me, directly and indirectly with the preparation and completion of this study. I really appreciate my supervisors Prof. Tınaz Ekim Aşıcı and Prof. Z. Caner Taşkın to lead me in this journey and to spare their valuable time during the COVID-19 pandemic. Your advice and support during my thesis were very valuable for me and it was a pleasure to work with you.

I thank you Prof. Necati Aras and Assoc. Prof. Didem Gözüpek for participating in my master thesis committee. I appreciate your valuable comments and suggestions.

Last but not least I want to express my gratitude to my beloved wife and our big warm family. You are always there for me. I would not have been where I am now without your unconditioned love, endless support and patience.

ABSTRACT

INTEGER PROGRAMMING FORMULATIONS AND CUTTING PLANE ALGORITHMS FOR THE MAXIMUM SELECTIVE TREE PROBLEM

This thesis considers the Maximum Selective Tree Problem (MSelTP) as a generalization of the Maximum Induced Tree problem. Given an undirected graph with a partition of its vertex set into clusters, MSelTP aims to choose the maximum number of vertices such that at most one vertex per cluster is selected and the graph induced by the selected vertices is a tree. To the best of our knowledge, MSelTP has not been studied before although several related optimization problems have been investigated in the literature. We propose two mixed integer programming formulations for MSelTP; one based on connectivity constraints, the other based on cycle elimination constraints. In addition, we develop two exact cutting plane procedures to solve the problem to optimality. On graphs with up to 25 clusters, up to 250 vertices, and varying densities, we conduct computational experiments to compare the results of two solution procedures with solving a compact integer programming formulation of MSelTP. Our experiments indicate that the algorithm CPAXnY outperforms the other procedures overall except for graphs with low density and large cluster size, and that the algorithm CPAX yields better results in terms of the average time of instances optimally solved and the overall average time.

ÖZET

MAKSİMUM SEÇMELİ AĞAÇ PROBLEMİ İÇİN TAMSAYILI PROGRAMLAMA FORMÜLASYONLARI VE KESME DÜZLEMİ ALGORİTMALARI

Bu tezde Maksimum Tetiklenmiş Ağaç Problemi'nin bir çeşit genelleştirilmesi olarak Maksimum Seçmeli Ağaç Problemi (MSelTP) ele alınıyor. MSelTP, düğümleri kümeleştirilmiş yönsüz bir çizge üzerinde, her bir küme başına en fazla bir tane düğüm seçecek ve bu seçimin tetiklediği çizge bir ağaç olacak şekilde en çok sayıda düğüm seçebilmeyi hedefler. Bildiğimiz kadarıyla, birçok benzer optimizasyon problemi çalışılmış olmasına rağmen, MSelTP'nin daha önce çalışılmadığını söyleyebiliriz. MSelTP için biri bağlantılılık kısıtlarına dayalı, diğeri ise döngü eleme kısıtlarına dayalı olmak üzere iki tane karışık tamsayılı programlama formülasyonu önerdik. İlaveten, problemi optimum olarak çözebilmek için iki tane kesme düzlemi yöntemi geliştirdik. Bu iki çözüm yöntemini ve polinom sayıda kısıt içeren tamsayılı programlama formülasyonunu kıyaslamak amacıyla küme sayısı 25'e kadar olan, düğüm sayısı 250'ye kadar olan ve değişen yoğunluklara sahip çizgelerde hesaplamalı deneyler gerçekleştirdik. Deneylerimiz, CPAXnY algoritmasının diğer yöntemlere göre genel olarak daha iyi çalıştığına işaret etmekte, ancak düşük yoğunluklu ve kümelerindeki ortalama düğüm sayısı çok olan çizgelerde CPAX algoritmasının performansı ortalama süre bakımından daha iyi sonuç vermektedir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

$A(S)$	The set of arcs in A with both endpoints in S
C	A cycle
$D(V, A)$	A directed graph whose vertex set is V and arc set is A
$E(C)$	The edge set of cycle C
$E(S)$	The set of edges in E with both endpoints in S
$E[x]$	The edge set of $G[x]$
$G[x]$	A subgraph induced by the set of selected vertices
$G(V, E)$	An undirected graph whose vertex set is V and edge set is E
$G[V']$	A subgraph of G induced by V'
$\mathcal{K}$	A clustering of vertex set V of graph G
$t_{p,s}$	The running time to solve instance $p \in \mathcal{P}$ by method $s \in \mathcal{S}$
$r_{p,s}$	The performance ratio of method $s \in \mathcal{S}$ on instance $p \in \mathcal{P}$
$V(C)$	The vertex set of cycle C
V_k	Cluster k in the clustering $\mathcal{K}$
$\delta(S)$	The set of edges in E with exactly one endpoint in S
$\delta^-(S)$	The set of arcs in A with head ends in S
$\delta^+(S)$	The set of arcs in A with tail ends in S
$\rho_s(\tau)$	Performance profile of method $s \in \mathcal{S}$ within a factor τ of the performance ratio

LIST OF ACRONYMS/ABBREVIATIONS

GMSTP	Generalized Minimum Spanning Tree Problem
GRASP	Greedy Randomized Adaptive Search Procedures
GTSP	Generalized Travelling Salesman Problem
L-GMST	A Variant of Generalized Minimum Spanning Tree Problem
LB	Lower Bound
MIP	Mixed Integer Programming
MSelTP	Maximum Selective Tree Problem
PCGMSTP	Prize-Collecting Generalized Minimum Spanning Tree Problem
SelCol	Selective Graph Coloring Problem
SelCol+	Maximum variant of Selective Graph Coloring Problem
SelTP	The Decision Variant of MSelTP
UB	Upper Bound

1. INTRODUCTION

In this thesis, we consider the Maximum Selective Tree Problem (MSelTP), which is a generalization of the Maximum Induced Tree problem. Given an undirected graph with a partition of its vertex set into clusters, MSelTP aims to select at most one vertex per cluster such that the graph induced by the selected vertices is a tree and among all possible vertex selections, the order of the induced tree is maximized. The Maximum Induced Tree problem is a special case of MSelTP where the partition of its vertex set is into singletons. Moreover, the graph induced by the selection is a minimal connected graph that connects maximum number of clusters. To the best of our knowledge, MSelTP has not been studied in the literature. For problems where selections of vertices force to consider all edges whose both end vertices are selected, it can be important to obtain a minimal connected graph, which is an induced tree. A water-pipe network, gas-pipe network or a type of circuit having a clustered structure and where the flow uses all pipes or wires among allowed/selected nodes can be application examples of MSelTP.

1.1. Motivation and Literature Review

Graph theory techniques have been widely used and studied to solve combinatorial optimization problems in several fields for years. The travelling salesman problem, the minimum spanning tree problem, the knapsack problem, graph coloring problems etc. are well known problems that have been solved using graph structures.

Induced subgraphs have been theoretically studied and widely used in numerous applications to solve combinatorial optimization problems. In a variety of areas, induced subgraphs with different structures such as induced paths, induced cycles, induced matchings, cliques, induced forests and trees are needed. For some of the problems in these areas, it is aimed to find an induced subgraph of maximum size. Searching for maximum cliques, longest induced paths, induced matchings of maxi-

mum size, and maximum induced trees are included in these kind of problems [1–9].

The maximum induced tree problem is the problem of selecting the maximum number of vertices that induce a tree. Erdös and Palka [1], and Karonski and Palka [2] study maximal induced trees, which are induced trees such that no other tree properly contains them, and they mention and use the largest maximal tree, which is equivalent to the maximum induced tree. Subsequently, Erdös et al. [3] study maximum induced trees in terms of the complexity of the problem, bounds on the order of the maximum induced tree and the relationship of that order to other parameters associated with the graph. Given three vertices, Chudnovsky and Seymour [4] study the problem of deciding whether there is an induced tree including all three of them. Derhy and Picouleau [5] investigate induced trees and induced paths with minimum weight or order including a given set of vertices. Rautenbach [6] studies induced dominating trees and bounds on the order of the maximum induced trees on connected cacti of maximum degree 3 and connected cubic graphs. Hertz et al. [7] compare the order of a maximum induced forest and a maximum induced tree. Melo and Ribeiro [8] present integer programming formulations for finding a maximum weighted induced forest and discuss how to extend it for the maximum weighted induced tree.

The feedback vertex set problem, also named as the decycling set problem, is related to the maximum induced forest problem [10]. Given a graph $G = (V, E)$, the feedback vertex set problem is to find the smallest set $S \subset V$ such that $G - S$ is acyclic. This problem is equivalent to finding a maximum induced forest, since $G - S$ is a forest for every feedback vertex set $S \subset V$.

Classical combinatorial optimization problems can often be generalized in various ways. One possible way, as already investigated in several papers [11–19], is to take as input a graph whose vertex set is partitioned into clusters. Then, the selective version consists of selecting one vertex per cluster so that the graph induced by the selected vertices admits, among all possible selections, the best solution for the original optimization problem. Given a graph whose vertex set is partitioned into clusters,

another way to generalize classical combinatorial optimization problems is to select representative vertices for each cluster and also edges whose both end vertices are selected and satisfy constraints in the original optimization problem. In this way, the subgraph formed by the selection is not necessarily an induced graph. This notion has been studied in several papers [20–37], where the number of these representative vertices is required to be at least, at most or exactly one per cluster for different problem types.

Problems of selective nature have been widely studied in the literature. The selective version of each combinatorial optimization problem brings some flexibility into their applications by adding a set of alternatives but also adds in their computational complexity. Generalized network design problems, in general, are obtained by clustered graph instances expressing the feasibility conditions of the classical network design problem in terms of clusters [21]. The selective graph coloring problem (SelCol), introduced by Li and Simha [11] and widely studied since then [12–19], takes as input a clustered graph and aims to select exactly one vertex per cluster so that, among all possible such selections, the chromatic number of the graph induced by the selected vertices is minimized. For instance, the wavelength assignment problem is a problem that can be modelled as a graph coloring problem. In the classical version of the wavelength assignment problem, a route for each connection is given as input, while in the wavelength and routing assignment problem, the selective structure allows us to select a route for each connection from a given set of alternative routes [11]. Different types of real life problems modelled as SelCol and related classes of graphs are also emphasized in [12] such as scheduling and frequency assignment problems. Demange et al. [15] study the complexity of SelCol in several graph classes, such as split graphs, which are graphs partitioned into a clique and a stable set; complete q-partite graphs, which are graphs partitioned into q stable sets such that an edge exists for each vertex pair from different sets; and bipartite graphs, which are graphs partitioned into two stable sets. In [19], the graph classes corresponding to the problems in [12] are studied in terms of complexity, and the maximum variant of SelCol is introduced. This variant (SelCol+) aims to find a selection so that the chromatic number of the induced graph

is maximized; this corresponds to a worst selection which may happen when one does not have full control over the selection process, say for instance because selection decisions are made by a central authority having different criteria. For SelCol, a branch and cut algorithm is applied by Frota et al. [14], a memetic algorithm is proposed by Pop et al. [13] and a branch and price algorithm is presented by Furini et al. [16]. In perfect graphs, Şeker et al. propose a decomposition algorithm [17] and a cutting plane algorithm as its generalization [18].

The generalized spanning tree problem (GMSTP), which is introduced by Myung et al. in [20], is for a given undirected weighted graph and a partition of its vertex set into clusters, to find the minimum-cost tree that spans exactly one vertex from each cluster. GMSTP allows us to model local and global networks together for telecommunication networks, where hubs in local networks have to be connected via transmission links such as optical fibers to create a minimum cost global network [20]. Different types of real life problems represented by GMSTP are also emphasized in the literature, such as energy transportation [30] and agricultural irrigation [31]. As base formulations, Myung et al. in [20] propose generalized subtour elimination, generalized cutset, and flow based formulations. Fereman et al. present several new formulations and their comparison in [26] and the polyhedral analysis in [27]. Also, a new formulation called multi graph formulation is proposed by Sousa et al. [32]. GMSTP has also a variation that relaxes the "exactly one vertex per cluster" constraint as at least one vertex per cluster [24]. It is a particular case of the Generalized Steiner Tree Problem [24] and denoted as L-GMST problem [25]. Haouari et al. [28] propose two stochastic heuristics: probabilistic greedy search method and a genetic algorithm for this problem. Another variation of GMSTP is the prize-collecting generalized minimum spanning tree problem (PCGMSTP) introduced by Golden et al. [29], which also considers different prizes of vertices in the same cluster. When all the prizes of vertices in the same cluster are exactly equal, this problem corresponds to GMST problem. The summary of formulations generated so far and, latest advances on the problem related to both exact and heuristic algorithms are thoroughly overviewed by Pop [22]. GMSTP has similar constraints related to connectivity and absence of cycles as MSelTP. The flow

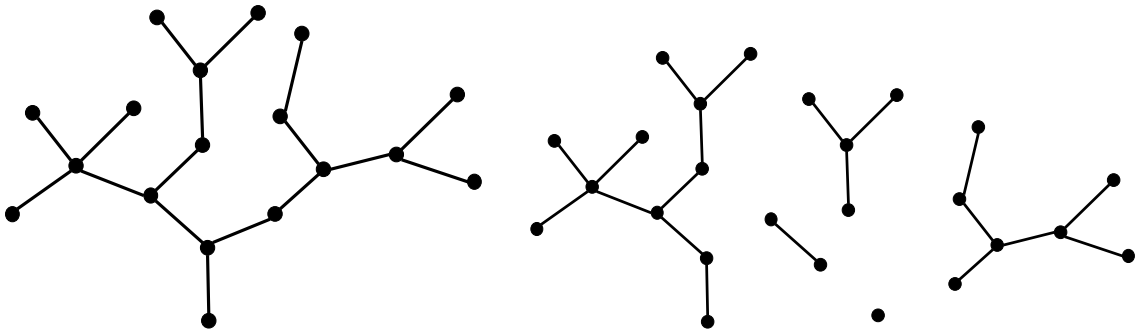
based formulation that we propose for MSelTP is based on the flow mechanism of the directed multicommodity flow model introduced in the same paper. Despite the similarities, MSelTP differentiates from GMSTP in a fundamental way. In GMSTP, a partial subgraph of the input graph is selected so that it forms a tree with minimum weight spanning all clusters. However, MSelTP is about selecting vertices and we require the subgraph induced by the selected vertices to form a tree. Even in an equally weighted graph, the vertex set of an optimum selection for GMSTP may not induce a tree. Besides, an optimum selection for MSelTP, which induces a tree, may not span all the clusters.

The group Steiner tree problem, which is introduced by Reich and Widmayer [23], is a more general version of GMSTP. In this problem, an undirected weighted graph is given, and a subset of its vertex set is partitioned into clusters. The aim is to find the minimum-cost tree that spans a subset having at least one vertex from each cluster. As an example, in the wire routing in VLSI design problem, this generalization allows us to determine also the pin location in each component [23]. Duin et al. transform the group Steiner tree problem to classical Steiner problem [34] and use it as a heuristic to solve the group Steiner tree problem.

A generalized version of the travelling salesman problem (GTSP) has also been considered in the literature. It aims to select exactly one vertex per cluster so that the selected vertices form a cycle with minimum cost [35]. As an example, a welfare client needs to travel with minimum cost through agencies, which are capable of servicing different type of needs of the client. This generalization allows a set of different cost alternatives for each service type, and the client can select one agency per service type so that the total cost is at minimum. Moreover, a variant of GTSP aims to select at least one vertex per cluster forming a cycle with minimum cost [36]. In the travel agent problem, for example, beyond arranging a minimum cost tour through cities, generalization allows the travel agent to provide a tour consisting of at least one city with each type of attractions at minimum cost. Laporte et al. [37] discuss several combinatorial optimization problems that can be modelled as GTSP for both variants.

1.2. Definitions

A graph that contains no cycle is said to be *acyclic* and a *tree* is defined as a connected acyclic graph [38]. An example of a tree is shown in Figure 1.1(a). An acyclic graph is also called a *forest* whose each component is a tree, hence the name forest [38]. An example of a forest is shown in Figure 1.1(b).



(a) A tree with 18 vertices. (b) A forest with 5 components, which are trees.

Figure 1.1. An example for a tree and a forest.

Bondy and Murty [38] state the following theorem:

Theorem 1.1 (Bondy and Murty [38]). *Let $G = (V, E)$ be a graph. If G is a tree, then $|E| = |V| - 1$.*

Thus, as a necessary condition, the number of edges needs to be one less than the number of vertices in a tree. Among the selections with this necessary condition, another theorem stated in [38] gives some conditions to ensure that the selection is a tree:

Theorem 1.2 (Bondy and Murty [38]). *Let G be a graph with n vertices and $n - 1$ edges. The following statements are equivalent:*

(i) *G is connected.*
(ii) *G is acyclic.*
(iii) *G is a tree.*

MSelTP is defined on an undirected graph $G = (V, E)$ with the vertices partitioned into m vertex sets called clusters. Let $|V| = n$ and $\mathcal{K} = \{V_1, \cdots, V_m\}$ be a clustering of V, that is, $V = V_1 \cup V_2 \cdots \cup V_m$ and $V_l \cap V_k = \emptyset$ for all $V_l, V_k \in \mathcal{K}$ such that $l \neq k$. We can assume without loss of generality that edges are defined only between vertices belonging to different clusters since the intracluster edges are irrelevant when at most one vertex is selected from each cluster.

Given a graph $G = (V, E)$ and $V' \subset V$, a graph whose vertex set is V', and whose edges are all the edges of G that have both ends in V' is called a subgraph of G induced by V', and denoted by $G[V']$ [38]. Moreover, for a $|V|$-dimensional binary vector x, let $G[x]$ represent the subgraph induced by the vertex set $\{i \in V \mid x_i = 1\}$, which is the set of selected vertices. Similarly, let $E[x]$ denote the edge set of $G[x]$.

MSelTP is the problem of selecting a maximum number of vertices such that at most one vertex is selected per cluster, and the graph induced by the selected vertices is a tree. Such a tree is called a *maximum selective tree*. Figure 1.2 illustrates a solution for MSelTP in an undirected graph with 15 vertices partitioned into 5 clusters.

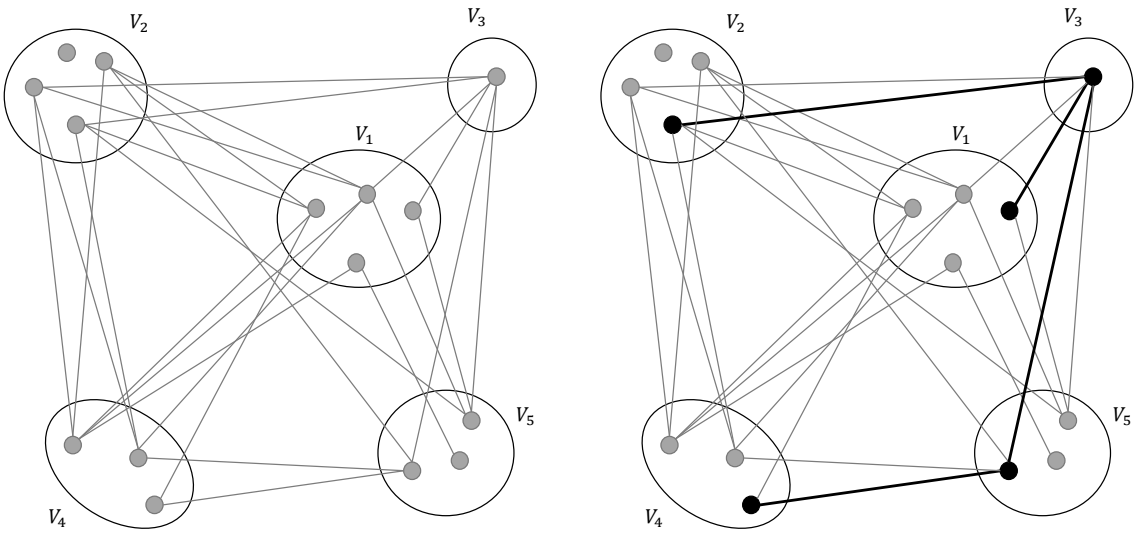


Figure 1.2. An optimal solution for an instance of MSelTP.

Given a graph $G = (V, E)$ and a subset $S \subset V$, the set of edges in E that have both endpoints in S is denoted by $E(S)$, and the set of edges in E that have only one endpoint in S is denoted by $\delta(S)$, which are defined as

$$E(S) = \{(i, j) \in E \mid i \in S,\ j \in S\}$$
$$\delta(S) = \{(i, j) \in E \mid i \in S,\ j \notin S\}.$$

Given a graph $G = (V, E)$, the directed graph associated with G is denoted by $D = (V, A)$, whose arc set A is composed of arcs $[i, j]$ and $[j, i]$ for each edge $(i, j) \in E$. Similarly, for a directed graph $D = (V, A)$ and a subset $S \subset V$, the set of arcs in A that have both endpoints in S is denoted by $A(S)$, and the set of arcs in A that have only tail or head in S is denoted as

$$A(S) = \{[i, j] \in A \mid i \in S,\ j \in S\}$$
$$\delta^+(S) = \{[i, j] \in A \mid i \in S,\ j \notin S\}$$
$$\delta^-(S) = \{[i, j] \in A \mid i \notin S,\ j \in S\}.$$

We use the notations $\delta^+(i)$ and $\delta^-(i)$ instead of $\delta^+(\{i\})$ and $\delta^-(\{i\})$ for brevity. The rest of the thesis is organized as follows: In Chapter 2, two integer programming formulations of MSelTP are presented: we present an exact formulation, namely flow based formulation (Model 1) in Section 2.1, and a formulation with an exponential number of constraints, namely cycle elimination formulation (Model 2) in Section 2.2. In Chapter 3, we develop two cutting plane algorithms, CPAXnY (Figure 3.1) and CPAX (Figure 3.3), based on the cycle elimination formulation. In Chapter 4, the computational results of the two cutting plane algorithms (CPAXnY and CPAX) and the flow based formulation (FLOW) are compared. The results are interpreted in Section 4.1 and as an evaluation technique, they are compared by performance profiles in Section 4.2. Additionally, we discuss on feeding an initial feasible solution to FLOW, CPAXnY and CPAX in Section 4.3. Finally, the thesis is concluded in Chapter 5 with a brief summary and possible future research directions.

2. FORMULATIONS FOR MSELTP

In this chapter, we first discuss the complexity of MSelTP. Then, we suggest two IP formulations for MSelTP.

Let SelTP be the decision variant of MSelTP, formulated as follows: Given a positive integer k and an undirected graph with a partition of its vertex set into clusters, is there a selection of at most one vertex per cluster such that the selected vertices induce a tree of order at least k?

If the vertices are partitioned into singletons, SelTP corresponds to the problem of finding an induced tree of order at least k, which is NP-complete [3]. Since the problem of finding an induced tree of order at least k is a special case of SelTP with $|V_k| = 1$ for all $k \in 1, \cdots, m$, SelTP is NP-complete as well. It follows that MSelTP is NP-hard.

2.1. Flow Based Formulation

We propose a formulation for MSelTP based on connectivity constraints. In the formulation, we use flow mechanism introduced by Myung et al. [20] in the directed multicommodity flow model. Since a connected graph with n vertices and $n-1$ edges is a tree by Theorem 1.2, the model aims to find a connected induced graph with maximum number of vertices, which is one more than the number of its edges.

In this model, each cluster $V_k \in \mathcal{K}$ corresponds to a commodity. For each commodity k, the flow of commodity k should start from the source cluster, which is uniquely designated (for all commodities) to the destination cluster V_k. Only one source is designated among all clusters from which a vertex is selected. In order to indicate the direction of flow, we use arcs $[i, j]$ and $[j, i]$ in A instead of each edge (i, j) in E, and we create a directed graph $D = (V, A)$ as a directed version of graph

$G = (V, E)$. Thus, there may be flow in each direction from i to j and from j to i for each edge $(i, j) \in E$. For each commodity k, we introduce non-integer variables f_{ij}^k to represent the flow of commodity k on arc $[i, j]$ and F_i^k to represent the net flow of commodity k through vertex i.

The following binary variables are introduced:

$$x_i = \begin{cases} 1 & \text{if the vertex } i \text{ is selected in the solution} \\ 0 & \text{otherwise} \end{cases}$$

$$y_{ij} = \begin{cases} 1 & \text{if the edge } (i, j) \in E \text{ is induced by the selected vertices} \\ 0 & \text{otherwise} \end{cases}$$

$$w_{ij} = \begin{cases} 1 & \text{if there is flow from the vertex } i \in V \text{ to the vertex } j \in V \\ 0 & \text{otherwise} \end{cases}$$

$$s_k = \begin{cases} 1 & \text{if the cluster } V_k \in \mathcal{K} \text{ is designated as the source cluster for all the commodities} \\ 0 & \text{otherwise} \end{cases}$$

$$t_k = \begin{cases} 1 & \text{if no vertex is selected from the cluster } V_k \in \mathcal{K} \\ 0 & \text{otherwise} \end{cases}$$

From a flow related point of view, the binary variable x_i indicates whether the vertex i is included in the network so that all flows are sent through, and y_{ij} indicates if the edge (i, j) is able to carry any flow. The variables f_{ij}^k, F_i^k, w_{ij} and s_k are auxiliary flow variables. The flow based model is as follows:

Model 1:

$$\min \quad \sum_{V_k \in \mathcal{K}} t_k \tag{2.1}$$

s.t.

$$\sum_{i \in V_k} x_i + t_k = 1 \qquad \forall V_k \in \mathcal{K} \tag{2.2}$$

$$\sum_{(i,j) \in E} y_{ij} = |\mathcal{K}| - 1 - \sum_{V_k \in \mathcal{K}} t_k \tag{2.3}$$

$$x_i + x_j - 1 \leq y_{ij} \qquad \forall (i,j) \in E \tag{2.4}$$

$$y_{ij} \leq x_i \qquad \forall (i,j) \in E \tag{2.5}$$

$$y_{ij} \leq x_j \qquad \forall (i,j) \in E \tag{2.6}$$

$$\sum_{V_k \in \mathcal{K}} s_k = 1 \tag{2.7}$$

$$s_k \leq 1 - t_k \qquad \forall V_k \in \mathcal{K} \tag{2.8}$$

$$\sum_{j\colon [i,j] \in A} f_{ij}^k - \sum_{j\colon [j,i] \in A} f_{ji}^k = F_i^k \qquad \forall i \in V, \quad V_k \in \mathcal{K} \tag{2.9}$$

$$F_i^k \leq s_k - x_i \qquad \forall i \in V_k, \quad V_k \in \mathcal{K} \tag{2.10}$$

$$F_i^k \geq s_k - 1 \qquad \forall i \in V_k, \quad V_k \in \mathcal{K} \tag{2.11}$$

$$F_i^k \leq s_l \qquad \forall i \in V_l, \quad V_l \in \mathcal{K},\ l \neq k \tag{2.12}$$

$$F_i^k \leq 1 - t_k \qquad \forall i \in V_l, \quad V_l \in \mathcal{K},\ l \neq k \tag{2.13}$$

$$F_i^k \geq x_i + s_l - t_k - 1 \qquad \forall i \in V_l, \quad V_l \in \mathcal{K},\ l \neq k \tag{2.14}$$

$$F_i^k \geq 0 \qquad \forall i \in V_l, \quad V_l \in \mathcal{K},\ l \neq k \tag{2.15}$$

$$f_{ij}^k \leq w_{ij} \qquad \forall [i,j] \in A, \quad V_k \in \mathcal{K} \tag{2.16}$$

$$w_{ij} + w_{ji} = y_{ij} \qquad \forall (i,j) \in E \tag{2.17}$$

$$f_{ij}^k \geq 0 \qquad \forall [i,j] \in A, \quad V_k \in \mathcal{K} \tag{2.18}$$

$$0 \leq y_{ij} \leq 1 \qquad \forall (i,j) \in E \tag{2.19}$$

$$x_i \in \{0,1\} \qquad \forall i \in V \tag{2.20}$$

$$w_{ij} \in \{0,1\} \qquad \forall [i,j] \in A \tag{2.21}$$

$$s_k \in \{0,1\} \qquad \forall V_k \in \mathcal{K} \tag{2.22}$$

$$t_k \in \{0,1\} \qquad \forall V_k \in \mathcal{K}. \tag{2.23}$$

Summing all t-variables, the objective function minimizes the number of clusters having no selected vertex (2.1). By definition, if t_k equals 1, then, $\sum_{i \in V_k} x_i$ needs to be 0. Moreover, if t_k equals 0, then, at least one vertex has to be selected in the cluster V_k, which means $\sum_{i \in V_k} x_i$ needs to be at least 1. Constraint (2.2) ensures that the subgraph defined by any solution has at most one vertex from each cluster and that t_k takes value 1 if no vertex is selected from cluster V_k and 0 otherwise. It also ensures the fact that the number of selected vertices equals the number of clusters containing a selected vertex, which leads to the equation

$$\sum_{i \in V} x_i = \sum_{V_k \in \mathcal{K}} \sum_{i \in V_k} x_i = \sum_{V_k \in \mathcal{K}} (1 - t_k) = |\mathcal{K}| - \sum_{V_k \in \mathcal{K}} t_k. \tag{2.24}$$

By Theorem 1.1, $|E[x]|$, the number of edges induced by the selected vertices, should be exactly one less than the number of selected vertices; this is ensured by constraint (2.3), which is derived from (2.24).

Two selected vertices force the edge between them to be in the (induced) subgraph, with constraint (2.4). For an edge to be in the induced subgraph, its both end vertices have to be selected, as forced by constraints (2.5) and (2.6). Thus, constraints (2.4), (2.5) and (2.6) ensure that the subgraph defined by any solution is the induced subgraph by the selected vertices. It also allows us to relax y-variables as continuous (2.19) since these constraints force y-variables to be integral.

Flow constraints in general include a root vertex/cluster as a source to send all commodities. In MSelTP, the source cluster has to be designated among those clusters including a selected vertex. If the cluster V_k is designated as the source cluster, only then, the variable s_k should be 1. Constraint (2.7) forces that only one of the clusters is selected as the source cluster and constraint (2.8) ensures that only the clusters including a selected vertex can be a source cluster.

Constraints (2.9) are the net flow equations. Let V_s be the source cluster. The LHS of constraint (2.9) is the net flow of the commodity k over the vertex i, which

should be 1 if i is the selected vertex in the source cluster V_s unless $k = s$ or V_k does not include any selected vertices; -1 if i is the selected vertex in V_k unless $k = s$; and 0 otherwise. For a vertex i, there are two cases; it is included in either the cluster V_k, which is associated with the commodity k, or some other cluster V_l.

(i) $i \in V_k$

The cluster V_k is either the source cluster or not.

- If the cluster V_k is the source cluster ($s_k = 1$), the net flow needs to be 0 for each vertex i since the source and the destination for the commodity k is the same.
- Otherwise:

  If the vertex i is included in the selection ($x_i = 1$), then the cluster V_k is the destination cluster for the commodity k. Thus, the net flow needs to be -1. Otherwise ($x_i = 0$), regardless of whether any vertex in the cluster V_k is included in the selection or not, no flow exists on the vertex i, the net flow needs to be 0.

We summarize this case as the net flow of the commodity k over a vertex $i \in V_k$ equals the maximum of $-x_i$ and $s_k - 1$ when the vertex i is included in the cluster V_k, which is expressed as

$$F_i^k = \max(s_k - 1, -x_i) \qquad \forall i \in V_k,\ V_k \in \mathcal{K}.$$

We show that if the vertex i is not selected, then the net flow is implicitly 0 as

$$x_i = 0 \Rightarrow y_{ij} = 0 \Rightarrow w_{ij} = w_{ji} = 0 \Rightarrow f_{ij}^k = f_{ji}^k = 0 \quad \forall j\colon (i,j) \in E,\ \forall V_k \in \mathcal{K}$$
$$\Rightarrow \sum_{j\colon [i,j] \in A} f_{ij}^k - \sum_{j\colon [j,i] \in A} f_{ji}^k \Rightarrow F_i^k = 0 \quad \forall V_k \in \mathcal{K}.$$

This case is ensured by constraints (2.10) and (2.11) as

$$F_i^k = \max(s_k - 1, -x_i) \Longrightarrow \begin{cases} F_i^k \leq s_k - x_i \\ F_i^k \geq s_k - 1 \end{cases} \qquad \forall i \in V_k,\ V_k \in \mathcal{K}.$$

(ii) $i \in V_l$ for $l \neq k$

The cluster V_k either contains a selected vertex or not.

- If no vertex in the cluster V_k is included in the selection ($t_k = 1$), the net flow needs to be 0 for each vertex i since no destination exists for the commodity k.
- Otherwise:

  The cluster V_l is either the source cluster or not.

  If the cluster V_l is the source cluster ($s_l = 1$), the net flow depends on if the vertex is selected.

  If the vertex i is selected ($x_i = 1$), then it is also the source vertex for the commodity k. Thus, the net flow needs to be 1.

  Otherwise ($x_i = 0$), no flow exists on the vertex i, the net flow needs to be 0.

  If the cluster V_l is not the source cluster ($s_l = 0$), the net flow needs to be 0 for each vertex i since the cluster V_l is neither the source nor the destination for the commodity k.

We summarize this case as the net flow of the commodity k over a vertex $i \in V_l$ equals the minimum of x_i, s_l, and $1 - t_k$ when the vertex i is included in the cluster V_l, which is expressed as

$$F_i^k = \min(s_l, 1 - t_k, x_i) \qquad \forall i \in V_l,\ V_l \in \mathcal{K},\ l \neq k.$$

This case is ensured by constraint (2.12)-(2.15) as

$$F_i^k = \min(s_l, 1 - t_k, x_i) \Longrightarrow \begin{cases} F_i^k \leq s_l \\ F_i^k \leq 1 - t_k \\ F_i^k \geq x_i + s_l - t_k - 1 \\ F_i^k \geq 0 \end{cases} \qquad \forall i \in V_l,\ V_l \in \mathcal{K},\ l \neq k.$$

Thus, the net flow constraints (2.9)-(2.15) suggest that the subgraph defined by any solution has to be connected.

Considering the edge (i, j) in any selective tree and the flows f_{ij}^k for all $V_k \in \mathcal{K}$ on it, if we hypothetically eliminate this edge to split the selective tree into two trees, for each cluster V_k included in the same tree with the source cluster, the associated commodity k does not flow through (i, j). However, for each cluster V_k included in the other tree, each associated commodity k flows through (i, j) from the tree including the source cluster to the other tree, which indicates that those flows are in the same direction.

When the edge (i, j) is able to carry flow, w_{ij} is hereby used as an indicator of the flow direction for each commodity. Thus, constraints (2.16) and (2.17) force that the flow of each commodity on arc $[i, j]$ can be sent only if edge (i, j) is contained in $E[x]$, and if multiple commodities flow on edge (i, j), they all flow in the same direction.

Constraints (2.5), (2.6), (2.9), (2.16), (2.17) and (2.18) ensure that if the vertex i is not selected ($x_i = 0$), related y, w, f and F-variables are forced to be 0 as well. Thus, constraints (2.12)-(2.15) allow us to relax F_i^k as $-1 \leq F_i^k \leq 1,\ \forall i \in V,\ V_k \in \mathcal{K}$ since these constraints force F-variables to be integral.

Constraint (2.3) and the connectivity constraints (2.9), (2.16), (2.17) together ensure that the selection is also a tree from Theorem 1.2. Therefore, each feasible solution corresponds to a selection of an induced tree with at most one vertex per

cluster. If the objective function gives the result of $|\mathcal{K}| - 1$, then, the selection is also a generalized spanning tree.

Model 1 has $O((|V| + |E|) \times |\mathcal{K}|)$ constraints, and $O(|V| + |E| + |\mathcal{K}|)$ binary variables out of $O((|V|+|E|)\times|\mathcal{K}|)$ variables. Since the model has polynomial number of variables and constraints, it is a compact formulation for MSelTP.

2.2. Cycle Elimination Formulation

We propose another formulation with constraints eliminating cycles for each vertex subset. This formulation has an exponential number of constraints and $O(|V|+|\mathcal{K}|)$ binary variables. Since an acyclic graph with n vertices and $n-1$ edges is a tree from Theorem 1.2, the model aims to find an acyclic induced graph with maximum number of vertices, which is one more than the number of its edges.

The same set of variables x_i, y_{ij} and t_k as in Model 1 are used in the following integer programming formulation:

Model 2:

$$\min \quad \sum_{V_k \in \mathcal{K}} t_k \tag{2.25}$$

s.t.

$$\sum_{i \in V_k} x_i + t_k = 1 \qquad \forall V_k \in \mathcal{K} \tag{2.26}$$

$$\sum_{(i,j) \in E} y_{ij} = |\mathcal{K}| - 1 - \sum_{V_k \in \mathcal{K}} t_k \tag{2.27}$$

$$x_i + x_j - 1 \leq y_{ij} \qquad \forall (i,j) \in E \tag{2.28}$$

$$y_{ij} \leq x_i \qquad \forall (i,j) \in E \tag{2.29}$$

$$y_{ij} \leq x_j \qquad \forall (i,j) \in E \tag{2.30}$$

$$\sum_{(i,j) \in E(S)} y_{ij} \leq |S| - 1 \qquad \forall S \subset V, \quad 3 \leq |S| \tag{2.31}$$

$$0 \leq y_{ij} \leq 1 \qquad \forall (i,j) \in E \tag{2.32}$$

$$x_i \in \{0,1\} \qquad \forall i \in V \tag{2.33}$$

$$t_k \in \{0,1\} \qquad \forall V_k \in \mathcal{K}. \tag{2.34}$$

The flow formulation in Model 1 focuses on the connectivity of the induced subgraph, whereas Model 2 focuses on the fact that the induced graph has to be acyclic. Thus, only the objective function (2.25), constraint (2.26) ensuring at most one vertex per cluster, constraint (2.27) of order-size relation, and constraints (2.28), (2.29), (2.30) forcing the subgraph to be induced by the selected vertices are revisited in this model.

Constraint (2.31) suggests that the subgraph defined by any solution has to be acylic. For each vertex subset $S \subset V$ with at least 3 vertices, it eliminates all solutions containing a cycle of size $|S|$. Constraints (2.27) and (2.31) together ensure that the selection induces a tree from Theorem 1.2. Therefore, each feasible solution yields a selection of an induced tree with at most one vertex per cluster. If the optimal objective function value is $|\mathcal{K}| - 1$, then the selection is also a generalized spanning tree.

3. CUTTING PLANE METHODS

Model 2 has an exponential number of constraints, implying that only relatively small instances can be solved with this formulation. Therefore, we develop cutting plane algorithms that add cycle elimination constraints as needed.

Constraint (2.31) in Model 2 is of exponential cardinality and it ensures that the subgraph defined by any solution is acylic. Since there is an exponential number of constraints (2.31), we remove them from the formulation and generate them as needed in a cutting plane fashion. In particular, given a selection of vertices, the feasibility of constraint (2.31) is checked by running a Depth-First Search algorithm to detect cycles. If there is no cycle, then the selection is feasible. Otherwise, let C denote a detected cycle having vertex set $V(C)$ and edge set $E(C)$. We add a cut as

$$\sum_{(i,j)\in E(C)} y_{ij} \leq |V(C)| - 1. \tag{3.1}$$

The cutting plane algorithm CPAXnY summarizes the procedure above in Figure 3.1.

Model 2 has $O(|V| + |E| + |\mathcal{K}|)$ variables, which is significantly fewer than in Model 1. Furthermore, we observe that Model 2 can be reformulated in terms of only x and t-variables, hence eliminating y-variables. To do that, all constraints involving y-variables have to be re-expressed. Hence, the related formulation is:

Model 3:

$$\min \quad \sum_{V_k\in\mathcal{K}} t_k \tag{3.2}$$

s.t.

$$\sum_{i\in V_k} x_i + t_k = 1 \qquad \forall V_k \in \mathcal{K} \tag{3.3}$$

$$|E[x]| = |\mathcal{K}| - 1 - \sum_{V_k \in \mathcal{K}} t_k \tag{3.4}$$

$$G[x] \text{ admits no cycle} \tag{3.5}$$

$$x_i \in \{0,1\} \qquad \forall i \in V \tag{3.6}$$

$$t_k \in \{0,1\} \qquad \forall V_k \in \mathcal{K}. \tag{3.7}$$

Require: A graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ with clustering $\mathcal{K}$
Ensure: An optimal selection $\mathbf{x}^*$ of vertices inducing a maximum selective tree
while true do
Solve Model 2 with constraint (2.31) relaxed. Let $\hat{\mathbf{x}}$ be an optimal selection of vertices, and $\mathbf{G}[\hat{\mathbf{x}}]$ be the subgraph induced by $\hat{\mathbf{x}}$.
Find cycles in the induced graph, if any, by depth-first search.
if $\mathbf{G}[\hat{\mathbf{x}}]$ does not contain any cycles **then**
The selection $\hat{\mathbf{x}}$ is optimal.
break
else
For some cycle $\mathbf{C}$, generate cut (3.1) and add it to Model 2.
end if
end while
$\mathbf{x}^* \leftarrow \hat{\mathbf{x}}$

Figure 3.1. CPAXnY Algorithm.

Model 3 has $O(|V| + |\mathcal{K}|)$ variables, which is fewer than in Model 1 and Model 2. Since the sum of y-variables in constraint (2.27) indicates the number of edges induced by the selected vertices, the size of $E[x]$, which denotes the edge set of $G[x]$, is used in constraint (3.4). Since $E[x]$ in (3.4) depends on the selection and since there is an exponential number of (3.5), we relax constraints (3.4) and (3.5). At any

point, if constraint (3.4) is not satisfied, we need to add a cut to eliminate the current selection from the solution set. Laporte and Louveaux define a binary optimality cut to exclude the current solution [39]. Given binary solution v^n with $S = \{i \mid v_i^n = 1\}$ and $S' = \{i \mid v_i^n = 0\}$, the binary optimality cut is defined as

$$\sum_{i \in S}(1 - v_i) + \sum_{i \in S'} v_i \geq 1.$$

In our case, the binary variables are x and t-variables. Let $\hat{V}$ be the set of selected vertices $\hat{x}$, and $\hat{\mathcal{K}}$ be $\{V_k \in \mathcal{K} \mid \hat{t}_k = 1\}$, indicating the set consisting of clusters, in which no vertex is selected. Thus, in Model 3, the corresponding binary optimality cut is

$$\sum_{i \in \hat{V}}(1 - x_i) + \sum_{i \in V \setminus \hat{V}} x_i + \sum_{V_k \in \hat{\mathcal{K}}}(1 - t_k) + \sum_{V_k \in \mathcal{K} \setminus \hat{\mathcal{K}}} t_k \geq 1. \tag{3.8}$$

By constraint (3.3), changing a 0-valued x or t variable to 1 forces to change a 1-valued variable to 0, and vice versa. Thus, Equation (3.8) can be improved as

$$\sum_{i \in \hat{V}}(1 - x_i) + \sum_{V_k \in \hat{\mathcal{K}}}(1 - t_k) \geq 1. \tag{3.9}$$

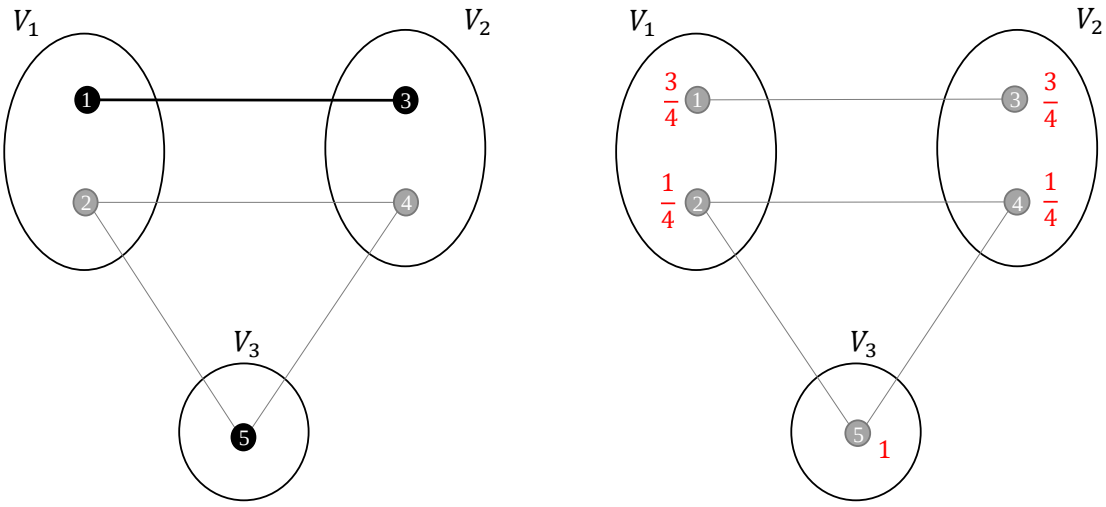
Since at most one vertex can be selected from each cluster by constraint (3.3), the sum of the number of clusters having no selected vertex and the number of selected vertices is equal to the total number of clusters, which leads $|\hat{\mathcal{K}}| + |\hat{V}| = |\mathcal{K}|$. By that equation, an equivalent of Equation (3.9) is as

$$\sum_{i \in \hat{V}} x_i + \sum_{V_k \in \hat{\mathcal{K}}} t_k \leq |\mathcal{K}| - 1. \tag{3.10}$$

Proposition 3.1. *Cut* (3.10) *is stronger than cut* (3.8).

Proof. Any nonnegative pair (x, t) satisfying (3.9) and hence (3.10) also satisfies (3.8).

To show that cut (3.10) is stronger than cut (3.8), we consider a graph $G = (V, E)$ with clustering $\mathcal{K} = \{V_1, V_2, V_3\}$ where $V = \{1, 2, 3, 4, 5\}$, $V_1 = \{1, 2\}$, $V_2 = \{3, 4\}$, $V_3 = \{5\}$, and $E = \{\{1, 3\}, \{2, 4\}, \{2, 5\}, \{4, 5\}\}$. One possible selection for this graph is $\hat{x} = \{1, 0, 1, 0, 1\}$, hence $\hat{t} = \{0, 0, 0\}$ as in Figure 3.2(a). The induced graph $G[\hat{x}]$ is, then, composed of vertices $\{1, 3, 5\}$ and edge $(1, 3)$, and constraint (3.4) is not satisfied. This is a case where a binary optimality cut is generated. Cut (3.8) associated with selection $\hat{x}$ is $(1 - x_1) + (1 - x_3) + (1 - x_5) + x_2 + x_4 + t_1 + t_2 + t_3 \geq 1$, whereas cut (3.10) associated with selection $\hat{x}$ is $x_1 + x_3 + x_5 \leq 2$. The point (x, t) with $x = (3/4, 1/4, 3/4, 1/4, 1)$ and $t = (0, 0, 0)$, as in Figure 3.2(b), satisfies constraint (3.3) and cut (3.8); however, cut (3.10) is violated. Hence, cut (3.10) is stronger than cut (3.8). □



(a) A selection that violates (3.4).

(b) A fractional point that satisfies cut (3.8), and violates cut (3.10).

Figure 3.2. A graph example to illustrate that cut (3.10) is stronger than (3.8).

Thus, when the number of edges does not satisfy the condition (3.4), cut (3.10) is generated. The feasibility of constraint (3.5) is checked by running a Depth-First Search algorithm to detect cycles. If there is no cycle, then the selection (x, t) is optimum. Otherwise, let C denote a detected cycle having vertex set $V(C)$. We add a cut as

$$\sum_{i \in V(C)} x_i \leq |V(C)| - 1. \tag{3.11}$$

The cutting plane algorithm CPAX summarizes the procedure above in Figure 3.3.

Require: A graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ with clustering $\mathcal{K}$
Ensure: An optimal selection $\mathbf{x}^*$ of vertices inducing a maximum selective tree
while true do
Solve Model 3 with constraints (3.4) and (3.5) relaxed. Let $\hat{\mathbf{x}}$ be an optimal selection of vertices, and $\mathbf{G}[\hat{\mathbf{x}}]$ be the subgraph induced by $\hat{\mathbf{x}}$.
Find the subgraph $\mathbf{G}[\hat{\mathbf{x}}]$ induced by $\hat{\mathbf{x}}$.
Find cycles in $\mathbf{G}[\hat{\mathbf{x}}]$, if any, by depth-first search.
if $\mathbf{G}[\hat{\mathbf{x}}]$ does not contain any cycles **then**
if $|E[\hat{x}]| = |\mathcal{K}| - 1 - \sum_{V_k \in \mathcal{K}} \hat{t}_k$ **then**
The selection is optimal.
break
else
Generate cut (3.10) and add it to Model 3.
end if
else
For some cycle $\mathbf{C}$, generate cut (3.11) and add it to Model 3.
end if
end while
$\mathbf{x}^* \leftarrow \hat{\mathbf{x}}$

Figure 3.3. CPAX Algorithm.

4. EXPERIMENTAL RESULTS

In this chapter, we discuss the computational experiments that we conduct to compare the results of two solution procedure CPAXnY and CPAX with FLOW, which is solving the compact integer programming formulation Model 1 of MSelTP, in terms of their performances and advantages.

4.1. Interpretation of the Results

We implement the algorithms in C++ and conduct experiments on a computer with an Intel Core i5-7200U @ 2.50 GHz processor and 8GB RAM with a time limit of 1500 seconds for each experiment. We use CPLEX 20.1 as a solver. We use lazy callback mechanism in CPLEX to apply the cutting plane algorithms.

We randomly generate test instances as graphs with clusters varying from 5 to 25. We call the number of vertices in a cluster the *cluster size*, and we use three different average cluster size values, which are 3, 6 and 10. As an average density value, we use 0.1, 0.3, 0.5, and 0.7. For each number of clusters, average cluster size and edge density, 10 random instances are generated.

The graphs are randomly generated so that a vertex can be in any cluster with the same probability and for each pair of vertices, the probability of forming an edge between them is the same, which is the edge density value. However, we assume that there is no empty cluster and edges are defined only between vertices belonging to different clusters since the intracluster edges are irrelevant when at most one vertex is selected from each cluster. Additionally, we only choose connected graphs as test instances. In CPAXnY and CPAX, in iterations where multiple cycles are found, we limit the number of cuts to be added in an iteration to 75.

Table 4.1. Summary table for all graph instances.

cluster size	density	FLOW # opt	FLOW avg gap nonopt	FLOW avg time	CPAXnY # opt	CPAXnY avg gap nonopt	CPAXnY avg time	CPAX # opt	CPAX avg gap nonopt	CPAX avg time
small	low	80	83.46	331.88	100		5.20	98	61.90	63.33
	high	42	76.71	902.35	100		31.96	81	31.48	372.40
	all	122	78.44	617.11	200		18.58	179	34.38	217.86
medium	low	57	99.82	718.64	92	63.35	147.82	89	80.28	190.67
	high	31	97.63	1088.51	76	61.59	485.60	58	74.35	709.72
	all	88	98.47	903.57	168	62.03	316.71	147	75.58	450.19
large	low	46	100.00	903.01	90	91.64	189.26	90	92.33	178.32
	high	23	99.93	1187.03	57	85.63	736.31	42	88.41	896.16
	all	69	99.96	1045.02	147	86.77	462.78	132	88.99	537.24
overall		279	94.21	855.24	515	77.45	266.02	458	75.91	401.77

We conduct experiments on 600 test instances of MSelTP to compare the three methods. The number of clusters in these graphs are 5, 10, 15, 20, and 25. Each of them has 3 different sizes as "small", "medium", and "large", corresponding to the average number of vertices per cluster, which are 3, 6, and 10. There are 4 different density options for each case as 0.1 and 0.3 being "low" densities and 0.5 and 0.7 being "high" densities. For each tuple of types, 10 random cases are generated. We first present the general analysis of the results in Table 4.1. The cases are grouped by their cluster sizes and densities, in which each combination represents 100 instances. The first two columns in this table identify these combinations. Next columns are split into groups for each method (FLOW, CPAXnY, CPAX). In each column group, the first column shows the number of optimally solved instances out of 100. The second column shows the average of the relative optimality gap for nonoptimal cases in percent within the given time limit of 1500 seconds. The relative optimality gap is calculated as $\left(\frac{UB-LB}{UB} \times 100\right)$, where LB is the best known lower bound on the optimal solution value and UB is the upper bound, which is the best integer objective found. The

third column shows the average overall time in seconds within the time limit. For each cluster size group, the last row corresponds to aggregate values for all density groups.

The values at the last row of each group and the final row indicate that CPAXnY method outperforms the other two methods in terms of each criteria in the table, except the total average relative gap. Moreover, CPAX method yields better results than FLOW. CPAXnY and CPAX optimally solves 86% and 76% of the instances, respectively. FLOW solves only 47% of the instances optimally. In terms of the average relative gap and the average time, the ranking between the methods is the same. For only large-size low-density group of instances, CPAX method seems better than CPAXnY in terms of the average time, despite yielding the average relative gap a bit worse, while their number of optimally solved instances are the same. Moreover, CPAX method seems a bit better than CPAXnY in terms of the total average relative gap although CPAXnY method yields better average relative gaps for each group. The reason is that CPAXnY solves all small-size group optimally, therefore it has no nonoptimal case, which affects the total average relative gap.

We observe that the performance of each method deteriorates as the average cluster size increases. Similarly, as the graph gets denser, the performance of each method becomes worse. While CPAXnY method spends a few seconds on average for small-size low-density group and it solves each one of them to optimality, it rises to 736 seconds for large-size high-density group, and it solves only 57% of the instances optimally.

In order to analyze the results better, we compare the methods separately for different cluster sizes and densities. In our next set of experiments, we compare the results of three methods on graphs of which, the average number of vertices per cluster is 3, in other words graphs with “small” cluster size. The summary of the comparison is in Table 4.2.

Table 4.2. Computational results for graphs with small cluster size.

			FLOW					**CPAXnY**						**CPAX**					
# clust	# vert	avg density	# opt	avg % gap in nonopt	avg % gap overall	avg time in opt	avg time overall	# opt	avg % gap in nonopt	avg % gap overall	avg time in opt	avg time overall	avg time in subpr	# opt	avg % gap in nonopt	avg % gap overall	avg time in opt	avg time overall	avg time in subpr
5	15	0.1	10		0.00	0.22	0.22	10		0.00	0.02	0.02	0.00	10		0.00	0.11	0.11	0.01
5	15	0.3	10		0.00	0.25	0.25	10		0.00	0.03	0.03	0.00	10		0.00	0.05	0.05	0.00
10	30	0.1	10		0.00	0.71	0.71	10		0.00	0.03	0.03	0.00	10		0.00	3.95	3.95	0.09
10	30	0.3	10		0.00	2.94	2.94	10		0.00	0.05	0.05	0.00	10		0.00	0.02	0.02	0.00
15	45	0.1	10		0.00	1.74	1.74	10		0.00	0.05	0.05	0.00	10		0.00	26.60	26.60	0.40
15	45	0.3	10		0.00	134.85	134.85	10		0.00	0.52	0.52	0.00	10		0.00	0.06	0.06	0.01
20	60	0.1	10		0.00	31.11	31.11	10		0.00	0.26	0.26	0.01	10		0.00	36.27	36.27	0.44
20	60	0.3	0	77.03	77.03		1500.00	10		0.00	3.93	3.93	0.01	10		0.00	12.59	12.59	0.14
25	75	0.1	10		0.00	146.96	146.96	10		0.00	6.52	6.52	0.04	9	100.00	10.00	6.31	155.68	0.26
25	75	0.3	0	89.90	89.90		1500.00	10		0.00	40.64	40.64	0.02	9	23.81	2.38	275.48	397.93	0.52
			80	83.46	16.69	39.85	331.88	100	0.00	0.00	5.20	5.20	0.01	98	61.90	1.24	34.01	63.33	0.18
5	15	0.5	10		0.00	0.21	0.21	10		0.00	0.03	0.03	0.00	10		0.00	0.02	0.02	0.00
5	15	0.7	10		0.00	0.26	0.26	10		0.00	0.03	0.03	0.00	10		0.00	0.02	0.02	0.00
10	30	0.5	10		0.00	39.64	39.64	10		0.00	0.40	0.40	0.00	10		0.00	0.05	0.05	0.01
10	30	0.7	10		0.00	53.75	53.75	10		0.00	0.86	0.86	0.00	10		0.00	0.24	0.24	0.01
15	45	0.5	2	52.31	41.84	1148.08	1429.62	10		0.00	2.65	2.65	0.00	10		0.00	5.65	5.65	0.07
15	45	0.7	0	64.88	64.88		1500.00	10		0.00	5.21	5.21	0.00	10		0.00	19.54	19.54	0.12
20	60	0.5	0	80.89	80.89		1500.00	10		0.00	27.18	27.18	0.00	10		0.00	322.85	322.85	0.43
20	60	0.7	0	79.16	79.16		1500.00	10		0.00	24.42	24.42	0.00	10		0.00	428.15	428.15	0.57
25	75	0.5	0	88.40	88.40		1500.00	10		0.00	154.78	154.78	0.00	1	32.97	29.67	974.91	1447.49	0.74
25	75	0.7	0	89.75	89.75		1500.00	10		0.00	104.01	104.01	0.00	0	30.14	30.14		1500.00	0.00
			42	76.71	44.49	77.02	902.35	100	0.00	0.00	31.96	31.96	0.00	81	31.48	5.98	107.90	372.40	0.16

Table 4.3. Computational results for graphs with medium cluster size.

			FLOW					CPAXnY						CPAX					
# clust	# vert	avg density	# opt	avg % gap in nonopt	avg % gap overall	avg time in opt	avg time overall	# opt	avg % gap in nonopt	avg % gap overall	avg time in opt	avg time overall	avg time in subpr	# opt	avg % gap in nonopt	avg % gap overall	avg time in opt	avg time overall	avg time in subpr
5	30	0.1	10		0.00	0.20	0.20	10		0.00	0.03	0.03	0.00	10		0.00	0.09	0.09	0.01
5	30	0.3	10		0.00	0.44	0.44	10		0.00	0.05	0.05	0.00	10		0.00	0.02	0.02	0.00
10	60	0.1	10		0.00	2.53	2.53	10		0.00	0.07	0.07	0.00	10		0.00	49.45	49.45	0.56
10	60	0.3	10		0.00	35.46	35.46	10		0.00	0.13	0.13	0.00	10		0.00	0.02	0.02	0.00
15	90	0.1	10		0.00	52.15	52.15	10		0.00	0.22	0.22	0.00	9	100.00	10.00	8.66	157.80	0.20
15	90	0.3	2	99.46	79.57	875.46	1375.09	10		0.00	1.72	1.72	0.00	10		0.00	0.25	0.25	0.03
20	120	0.1	5	100.00	50.00	941.10	1220.55	10		0.00	0.67	0.67	0.01	10		0.00	6.37	6.37	0.20
20	120	0.3	0	99.75	99.75		1500.00	10		0.00	99.67	99.67	0.03	10		0.00	192.18	192.18	0.89
25	150	0.1	0	100.00	100.00		1500.00	10		0.00	2.81	2.81	0.03	10		0.00	0.48	0.48	0.09
25	150	0.3	0	99.89	99.89		1500.00	2	63.35	50.68	864.13	1372.83	0.04	0	78.30	78.30		1500.00	0
			57	99.82	42.92	129.19	718.64	92	63.35	5.07	30.24	147.82	0.01	89	80.28	8.83	28.84	190.67	0.22
5	30	0.5	10		0.00	0.74	0.74	10		0.00	0.03	0.03	0.00	10		0.00	0.02	0.02	0.00
5	30	0.7	10		0.00	1.85	1.85	10		0.00	0.05	0.05	0.00	10		0.00	0.02	0.02	0.00
10	60	0.5	10		0.00	506.90	506.90	10		0.00	0.98	0.98	0.00	10		0.00	0.35	0.35	0.02
10	60	0.7	1	90.26	81.23	256.02	1375.60	10		0.00	4.19	4.19	0.00	10		0.00	7.12	7.12	0.11
15	90	0.5	0	97.41	97.41		1500.00	10		0.00	34.52	34.52	0.00	10		0.00	222.16	222.16	0.60
15	90	0.7	0	97.73	97.73		1500.00	10		0.00	66.62	66.62	0.00	8	45.00	9.00	709.40	867.52	1.30
20	120	0.5	0	98.95	98.95		1500.00	8	37.18	7.44	699.24	832.71	0.01	0	64.00	64.00		1500.00	0.00
20	120	0.7	0	98.33	98.33		1500.00	8	24.62	4.92	771.07	916.86	0.01	0	63.57	63.57		1500.00	0.00
25	150	0.5	0	100.00	100.00		1500.00	0	73.36	73.36		1500.00	0.00	0	87.22	87.22		1500.00	0.00
25	150	0.7	0	100.00	100.00		1500.00	0	62.08	62.08		1500.00	0.00	0	88.47	88.47		1500.00	0.00
			31	97.63	67.37	172.61	1088.51	76	61.59	14.78	168.77	485.60	0.00	58	74.35	31.23	137.45	709.72	0.31

Table 4.4. Computational results for graphs with large cluster size.

			FLOW					**CPAXnY**						**CPAX**					
# clust	# vert	avg density	# opt	avg % gap in nonopt	avg % gap overall	avg time in opt	avg time overall	# opt	avg % gap in nonopt	avg % gap overall	avg time in opt	avg time overall	avg time in subpr	# opt	avg % gap in nonopt	avg % gap overall	avg time in opt	avg time overall	avg time in subpr
5	50	0.1	10		0.00	0.78	0.78	10		0.00	0.03	0.03	0.00	10		0.00	0.39	0.39	0.03
5	50	0.3	10		0.00	1.81	1.81	10		0.00	0.05	0.05	0.00	10		0.00	0.02	0.02	0.00
10	100	0.1	10		0.00	8.10	8.10	10		0.00	0.17	0.17	0.00	10		0.00	107.83	107.83	0.82
10	100	0.3	10		0.00	356.09	356.09	10		0.00	0.51	0.51	0.00	10		0.00	0.02	0.02	0.00
15	150	0.1	6	100.00	40.00	938.88	1163.33	10		0.00	0.86	0.86	0.00	10		0.00	4.99	4.99	0.15
15	150	0.3	0	100.00	100.00		1500.00	10		0.00	10.90	10.90	0.01	10		0.00	0.16	0.16	0.03
20	200	0.1	0	100.00	100.00		1500.00	10		0.00	5.59	5.59	0.01	10		0.00	1.73	1.73	0.15
20	200	0.3	0	100.00	100.00		1500.00	10		0.00	345.68	345.68	0.06	10		0.00	164.77	164.77	1.32
25	250	0.1	0	100.00	100.00		1500.00	10		0.00	28.77	28.77	0.03	10		0.00	3.31	3.31	0.26
25	250	0.3	0	100.00	100.00		1500.00	0	91.64	91.64		1500.00	0.00	0	92.33	92.33		1500.00	0.00
			46	100.00	54.00	202.20	903.01	90	91.64	9.16	43.62	189.26	0.01	90	92.33	9.23	31.47	178.32	0.31
5	50	0.5	10		0.00	2.84	2.84	10		0.00	0.06	0.06	0.00	10		0.00	0.02	0.02	0.00
5	50	0.7	10		0.00	6.65	6.65	10		0.00	0.07	0.07	0.00	10		0.00	0.03	0.03	0.00
10	100	0.5	3	100.00	70.00	1036.14	1360.84	10		0.00	2.53	2.53	0.00	10		0.00	0.27	0.27	0.02
10	100	0.7	0	99.93	99.93		1500.00	10		0.00	26.70	26.70	0.00	9	83.33	8.33	159.36	293.43	0.58
15	150	0.5	0	100.00	100.00		1500.00	10		0.00	465.66	465.66	0.01	3	72.86	51.00	392.90	1167.87	1.47
15	150	0.7	0	99.52	99.52		1500.00	7	44.44	13.33	701.94	868.05	0.00	0	73.24	73.24		1500.00	0.00
20	200	0.5	0	100.00	100.00		1500.00	0	87.30	87.30		1500.00	0.00	0	90.57	90.57		1500.00	0.00
20	200	0.7	0	100.00	100.00		1500.00	0	81.55	81.55		1500.00	0.00	0	92.46	92.46		1500.00	0.00
25	250	0.5	0	100.00	100.00		1500.00	0	94.44	94.44		1500.00	0.00	0	99.82	99.82		1500.00	0.00
25	250	0.7	0	100.00	100.00		1500.00	0	91.60	91.60		1500.00	0.00	0	97.35	97.35		1500.00	0.00
			23	99.93	76.94	139.28	1187.03	57	85.63	36.82	173.05	736.31	0.00	42	88.41	51.28	62.29	896.16	0.23

In each table hereafter, the first three columns in the table identify the number of clusters, the number of vertices, and average edge density across ten random instances. Next columns are split into groups for each method (FLOW, CPAXnY, CPAX). In each column group, the first column shows the number of optimally solved instances out of 10. The second column shows the average of the relative optimality gap in percent of the instances that are not solved to optimality within the given time limit of 1500 seconds, while the third column shows that of all the instances. The relative optimality gap is calculated as $\left(\frac{UB-LB}{UB} \times 100\right)$, where LB is the best known lower bound on the optimal solution value and UB is the upper bound, which is the best integer objective found. The fourth and the fifth columns show the average time in seconds of the instances that are not solved to optimality within the time limit and that of all the instances, respectively. The procedures CPAXnY and CPAX have one last column that shows average time spent in their subproblem in seconds of all the instances. As we can see from Tables 4.2, 4.3 and 4.4, in each row, the average time in subproblems is negligible. All the values in each row are the average result of the 10 instances for that case.

As we can see the results in Table 4.2, the algorithm CPAXnY outperforms the other procedures in terms of the number of optimally solved instances, the average of the relative optimality gap for both nonoptimal cases and overall, and the average time of instances optimally solved and the overall average time. For graphs with lower density and a smaller number of cluster, its results are similar with that of CPAX in terms of the average time of instances optimally solved and the overall average time, yet, as the number of clusters increases, it gave better results. Furthermore, for graphs with higher density, CPAXnY yields significantly better results than others. It still optimally solves all the cases whereas CPAX is not able to solve to optimality except one instance for graphs with 25 clusters. Moreover, for graphs with 20 and 25 clusters, the overall average time of CPAX is longer more than 10 times that of CPAXnY. Both outperform FLOW significantly, which is able to solve very few instances optimally as the number of clusters increases to 15 for graphs with higher density.

We conduct two additional sets of experiments in which we compare the methods on graphs with "medium" and "large" cluster sizes. The average number of vertices per cluster is 6 in graphs with "medium" cluster size, whereas it is 10 in the "large" one. The results of the comparison for graphs with medium and large cluster size are in Tables 4.3 and 4.4, respectively.

As we can see the results in Table 4.3, the algorithm CPAXnY still outperforms the other procedures in terms of each criteria. However, the performance of each method deteriorates as the cluster size increases. For graphs with lower density and fewer number of clusters, the results of CPAXnY are similar with that of CPAX in terms of the average time of instances optimally solved and the overall average time, yet, as the number of clusters increases, it gave better results. This time, instead of all cases, CPAXnY and CPAX optimally solve 92% and 89% of the instances, respectively. Furthermore, for graphs with higher density, CPAXnY yields much better results than others. However, it optimally solves 76% of the instances this time and 80% of the instances for graphs with 20 cluster, whereas CPAX is not able to solve any instances to optimality for graphs with 20 and 25 clusters. Moreover, CPAXnY has better average of the relative optimality gap for both nonoptimal cases and overall. Both outperform FLOW significantly. Among graphs with higher density, it is able to solve the instances optimally for only those with 5 clusters and half of those with 10 clusters.

As we can see the results in Table 4.4, for graphs with lower density, the algorithm CPAXnY outperforms the other procedures in terms of the average of the relative optimality gap for both nonoptimal cases and overall; however, the algorithm CPAX yields better results than the other procedures in terms of the average time of instances optimally solved and the overall average time. CPAXnY and CPAX solve the same number of instances to optimality. For graphs with lower density, both CPAXnY and CPAX optimally solve 90% of the instances. Nevertheless, for graphs with higher density, CPAXnY yields much better results than others in terms of each criteria. It optimally solves 57% of the instances this time and 85% of the instances for graphs with 15 cluster, whereas CPAX optimally solves 42% of the instances and it is not

able to solve any instances to optimality for graphs with 15, 20 and 25 clusters except 3 instances for those with 15. Their average times are closer than before. Moreover, CPAXnY has better average of the relative optimality gap for both nonoptimal cases and overall. Both outperform FLOW significantly. Among graphs with higher density, it is able to solve the instances optimally for only those with 5 clusters and 15% of those with 10 clusters. As a limitation, for the large-size high-density graph instances, CPAXnY is not able to solve after those with 20 clusters in 1500 seconds, whereas CPAX is not almost able to solve after those with 15 clusters in 1500 seconds and FLOW is not after those with 10 clusters. As the number of clusters or density increases, the performance of all methods deteriorates in general, especially FLOW method worsens faster. Intuitively, one can think that finding a tree gets easier as density increases; however, finding an induced tree gets harder since the possibility of including a cycle in an induced subgraph increases. Comparing Tables 4.2, 4.3 and 4.4, we observe also that for graphs with high density, the performance of all three methods gets worse in general as the cluster size increases. However, for graphs with low density, the deterioration in performance is less than high density graphs.

4.2. Comparison by Performance Profiles

To compare methods, using average of metrics can be misleading in some circumstances. The results of solving a few difficult cases, maybe unsolved within a specified time limit, can dominate the overall results. Only considering either optimally solved cases or non-solved cases, robust methods can seem worse on average times or gaps even though they solve more cases. To compare the performance of methods effectively as avoiding the concerns aforementioned, we use an evaluation technique, named performance profiles, that Dolan and More [40] propose for optimization methods, which has been widely used for years [41–47]. We use running time as a performance measure in performance profiles.

Let $\mathcal{P}$ be the set of all 600 instances (n_p problems) and $\mathcal{S}$ be the set of all methods, which is $\mathcal{S} = \{\text{FLOW}, \text{CPAXnY}, \text{CPAX}\}$. We denote the running time to

solve instance $p \in \mathcal{P}$ by method $s \in \mathcal{S}$ as $t_{p,s}$. For each instance, we compare each method by its relative performance to the best performance on the same instance, namely the *performance ratio*

$$r_{p,s} = \begin{cases} \dfrac{t_{p,s}}{\min\{t_{p,s} \colon s \in \mathcal{S}\}} & \text{if } s \text{ can optimally solve } p \\ r_M & \text{otherwise} \end{cases} \qquad \forall s \in \mathcal{S},\ \forall p \in \mathcal{P}.$$

r_M is chosen as large enough to satisfy $r_M \geq r_{p,s},\ \forall s \in \mathcal{S},\ \forall p \in \mathcal{P}$. Performance profile of a method is defined as the fraction of times that the performance ratio of the method is less than or equal to τ

$$\rho_s(\tau) = \frac{1}{n_p}\text{size}\{p \in \mathcal{P} \colon r_{p,s} \leq \tau,\ \tau \in \mathbb{R}\} \quad \forall s \in \mathcal{S},$$

which indicates the ratio of instances which this method can optimally solve within factor τ of the minimum time by any method. Thus, $1 - \rho_s(\tau)$ indicates the ratio of instances which this method cannot solve within factor τ of the minimum time by any method, and $\rho_s(1)$ represents the ratio of instances which this method can solve faster than others.

Performance profile of FLOW, CPAXnY and CPAX for $\tau \in [1, 10]$ is in Figure 4.1. From the figure, as a percentage of times in being the fastest on solving an instance optimally, CPAX seems to have the highest ratio of 46% and CPAXnY seems to have the ratio of 40%. Since FLOW seems to have no instance that it solves the fastest, the remaining fraction of 14% is the ratio of instances that no method can solve optimally. FLOW seems to have the worst performance. It solves no instance optimally within a factor of $\tau \leq 3.58$. For $\tau \geq 2$, CPAXnY solves more instance optimally within a factor of τ of the best method. If it is important to obtain the highest number of cases that a method solves faster than others, then, CPAX can suffice. However, instead of number of wins, a method that can solve 52% or a higher portion of instances with better efficiency is needed, CPAXnY seems to be the best method.

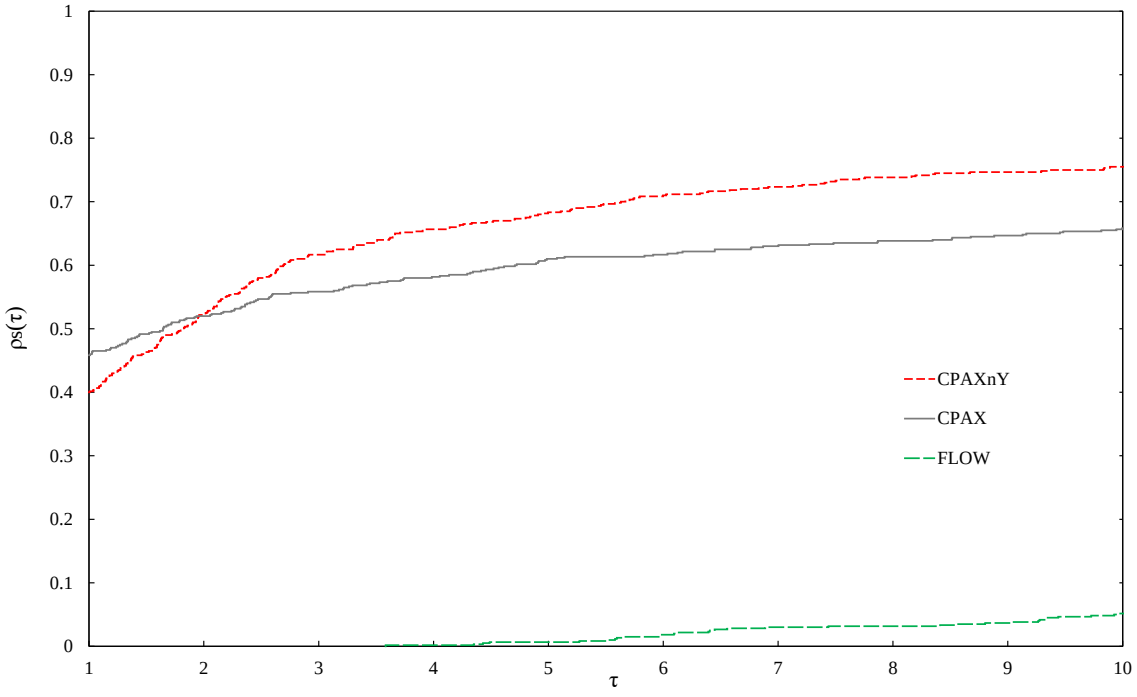


Figure 4.1. Performance profile on $[1, 10]$.

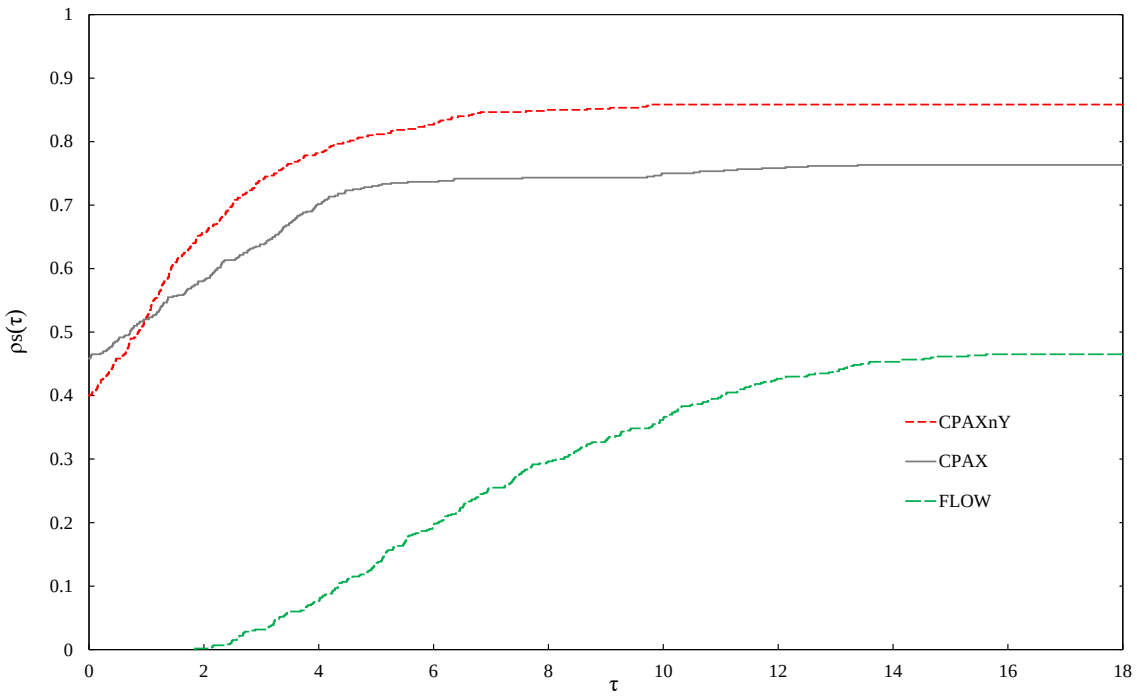


Figure 4.2. Performance profile in a $\log_2$ scale.

In order to see complete performance of the methods, since τ can be very large, we show a $\log_2$ scale of the performance profiles in Figure 4.2. In this case, each line of methods becomes flat after $\tau > 15.6$, which includes r_M. From the figure, it seems that CPAXnY can solve approximately 86% of instances optimally whereas for CPAX, it is 76%, which is competitive. FLOW can solve approximately 46% of instances optimally, which has the worst performance.

4.3. Feeding an Initial Feasible Solution

Providing an initial feasible solution can help solvers to reach the optimum solution more efficiently. We develop an algorithm to obtain an initial feasible solution and we analyze the objective values that the algorithm finds for each instance. We present our InitAA algorithm in Figure 4.3.

Require: A graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ with clustering $\mathcal{K}$
Ensure: A feasible selection $\mathbf{x}^*$ such that $\mathbf{G}[\mathbf{x}^*]$ is an induced tree.
  $\mathbf{L} \leftarrow \emptyset$
  Sort clusters according to the ascending order of their number of vertices.
  for all $\mathbf{V_k} \in \mathcal{K}$ **do**
    Sort vertices according to the ascending order of their degrees.
    for all $\mathbf{i} \in \mathbf{V_k}$ **do**
      if $\mathbf{L} == \emptyset$ **or** the number of edges from **i** to **L** equals 1 **then**
        $\mathbf{L} \leftarrow \{\mathbf{i}\}$
        break
      end if
    end for
  end for
  Find the corresponding $\mathbf{x}^*$ from **L**

Figure 4.3. InitAA Algorithm.

At the suffix of InitAA, A stands for sorting in ascending order. The first letter is for sorting clusters and the second is for sorting vertices. We compare InitAA and its different versions, where we use combinations of sorting in ascending and descending order. We call them InitDD, InitDA and InitAD. To test the performance of those Init algorithms, we calculate an average relative gap between the objective value the algorithm obtains for each instance as an upper bound (UB) and the maximum of lower bounds of 3 methods (FLOW, CPAXnY and CPAX) as a lower bound (LB). It is calculated as $\left(\frac{UB-LB}{UB} \times 100\right)$. We summarize the results as grouping instances by their cluster sizes and densities, in which each combination represents 100 instances in Table 4.5. The average running time of each Init algorithm is in under 1 second. Among Init algorithms, InitAA seems to be the best and InitAD is close to it. Although InitAD is better for medium-size low-density group and InitDD is better for small-size low-density group, in general, InitAA yields better average relative gaps than others. For each high-density group, it generates better average gap than FLOW, and it is close to CPAX for the large-size high-density group. It seems promising; thus, we test whether providing an initial solution by InitAA algorithm helps the methods to improve their performances.

Table 4.5. Comparison on average relative gap of methods and Init algorithms.

		FLOW	**CPAXnY**	**CPAX**	**InitAA**	**InitAD**	**InitDA**	**InitDD**
cluster size	density	avg % gap	avg % gap	avg % gap	avg % gap	avg % gap	avg % gap	avg % gap
small	low	16.69	0.00	1.24	81.23	81.56	87.88	78.94
	high	44.49	0.00	5.98	33.31	35.77	37.74	44.73
	all	30.59	0.00	3.61	57.27	58.66	62.81	61.84
medium	low	42.92	5.07	8.83	86.67	84.78	95.65	85.21
	high	67.37	14.78	31.23	45.52	49.54	55.76	65.17
	all	55.14	9.92	20.03	66.10	67.16	75.70	75.19
large	low	54.00	9.16	9.23	80.71	82.82	90.04	82.18
	high	76.94	36.82	51.28	62.93	64.96	74.21	82.55
	all	65.47	22.99	30.26	71.82	73.89	82.13	82.37
overall		50.40	10.97	17.96	65.06	66.57	73.55	73.13

For methods FLOW, CPAXnY and CPAX, we generate new versions with an initial solution provided by InitAA algorithm, and we call them FLOW_INIT, CPAXnY_INIT and CPAX_INIT. As we can see the results in Table 4.6, the method FLOW_INIT outperforms FLOW in terms of the number of optimally solved instances, the average of the relative optimality gap for both nonoptimal cases and overall, and the average time of instances optimally solved and the overall average time.

Table 4.6. Summary table of FLOW and FLOW_INIT for all graph instances.

cluster size	density	**FLOW**					**FLOW_INIT**				
		# opt	avg % gap in nonopt	avg % gap overall	avg time in opt	avg time overall	# opt	avg % gap in nonopt	avg % gap overall	avg time in opt	avg time overall
small	low	80	83.46	16.69	39.85	331.88	80	83.71	16.74	45.46	336.37
	high	42	76.71	44.49	77.02	902.35	44	76.26	42.70	116.26	891.16
	all	122	78.44	30.59	52.64	617.11	124	78.22	29.72	70.58	613.76
medium	low	57	99.82	42.92	129.19	718.64	65	99.64	34.87	173.16	637.55
	high	31	97.63	67.37	172.61	1088.51	32	97.20	66.10	193.45	1081.90
	all	88	98.47	55.14	144.49	903.57	97	98.03	50.48	179.85	859.73
large	low	46	100.00	54.00	202.20	903.01	55	100.00	45.00	48.02	701.41
	high	23	99.93	76.94	139.28	1187.03	26	99.94	73.96	162.92	1152.36
	all	69	99.96	65.47	181.22	1045.02	81	99.96	59.48	84.90	926.88
overall		279	94.21	50.40	113.41	855.24	302	93.75	46.56	109.52	800.13

As we can see in Table 4.7, CPAXnY outperforms CPAXnY_INIT in terms of the number of optimally solved instances, the average of the relative optimality gap overall, and the average time of instances optimally solved and the overall average time. Their relative optimality gaps are close to each other. However, CPAXnY_INIT has a bit better average of the relative optimality gap for nonoptimal cases. CPAXnY has better average time of instances optimally solved and overall average time, and CPAXnY solves more instances optimally.

Table 4.7. Summary table of CPAXnY and CPAXnY_INIT for all graph instances.

cluster size	density	CPAXnY # opt	CPAXnY avg % gap in nonopt	CPAXnY avg % gap overall	CPAXnY avg time in opt	CPAXnY avg time overall	CPAXnY_INIT # opt	CPAXnY_INIT avg % gap in nonopt	CPAXnY_INIT avg % gap overall	CPAXnY_INIT avg time in opt	CPAXnY_INIT avg time overall
small	low	100	0.00	0.00	5.20	5.20	100	0.00	0.00	6.56	6.56
	high	100	0.00	0.00	31.96	31.96	100	0.00	0.00	38.41	38.41
	all	200	0.00	0.00	18.58	18.58	200	0.00	0.00	22.48	22.48
medium	low	92	63.35	5.07	30.24	147.82	93	65.56	4.59	43.52	145.48
	high	76	61.59	14.78	168.77	485.60	69	54.10	16.77	151.60	569.60
	all	168	62.03	9.92	92.91	316.71	162	56.21	10.68	89.55	357.54
large	low	90	91.64	9.16	43.62	189.26	90	94.47	9.45	38.41	184.57
	high	57	85.63	36.82	173.05	736.31	56	86.08	37.88	211.81	778.61
	all	147	86.77	22.99	93.80	462.78	146	87.63	23.66	104.92	481.59
overall		515	77.45	10.97	64.30	266.02	508	74.65	11.45	67.56	287.20

As we can see in Table 4.8, CPAX and CPAX_INIT yield similar results. They both have the same number of optimally solved instances, and their relative optimality gaps are close to each other. CPAX has better average time of instances optimally solved and overall average time. CPLEX incorporates its own heuristic procedures and it normally uses them to find an initial feasible solution. The reason why feeding an initial feasible solution worsens CPAXnY and does not improve CPAX may be that CPLEX cannot find some solutions that it normally can with its own heuristic procedures.

As a result, providing an initial solution by InitAA algorithm improves only FLOW method. Comparing the results in Table 4.6 with those in Tables 4.7 and 4.8, we can see that CPAXnY and CPAX methods significantly outperform FLOW_INIT despite the improvement, since FLOW is the worst method and the improvement obtained by feeding initial solution is small.

Table 4.8. Summary table of CPAX and CPAX_INIT for all graph instances.

cluster size	density	CPAX # opt	CPAX avg % gap in nonopt	CPAX avg % gap overall	CPAX avg time in opt	CPAX avg time overall	CPAX_INIT # opt	CPAX_INIT avg % gap in nonopt	CPAX_INIT avg % gap overall	CPAX_INIT avg time in opt	CPAX_INIT avg time overall
small	low	98	61.90	1.24	34.01	63.33	98	64.29	1.29	44.72	73.83
	high	81	31.48	5.98	107.90	372.40	81	32.37	6.15	114.41	377.67
	all	179	34.38	3.61	67.44	217.86	179	35.41	3.72	76.25	225.75
medium	low	89	80.28	8.83	28.84	190.67	89	81.15	8.93	29.22	191.01
	high	58	74.35	31.23	137.45	709.72	57	74.75	32.14	162.49	737.62
	all	147	75.58	20.03	71.69	450.19	146	76.06	20.54	81.25	464.31
large	low	90	92.33	9.23	31.47	178.32	90	93.01	9.30	20.32	168.28
	high	42	88.41	51.28	62.29	896.16	43	88.91	50.68	105.67	900.44
	all	132	88.99	30.26	41.27	537.24	133	89.53	29.99	47.91	534.36
overall		458	75.91	17.96	61.27	401.77	458	76.40	18.08	69.62	408.14

5. CONCLUSIONS AND FUTURE RESEARCH

In this thesis, we consider the Maximum Selective Tree Problem (MSelTP), which aims to choose, for a given undirected graph with a partition of its vertex set into clusters, the maximum number of vertices such that at most one vertex per cluster is selected and the graph induced by the selected vertices is a tree. MSelTP has not been studied in the literature to the best of our knowledge. Therefore, we first observe that the problem is NP-hard.

We present two mixed integer programming formulations. One of them is a compact formulation and the other one is a formulation with an exponential number of constraints. Based on the latter formulation with an exponential number of constraints, we propose two cutting plane algorithms to solve MSelTP. We test and compare the algorithms for graphs with different density and structure in a computational experiment. The results show that the CPAXnY procedure yields better outcomes for most of the cases. As an evaluation technique, we compare them by performance profiles, as well. CPAX has more wins than others; however, CPAXnY solves 52% or a higher portion of instances with better efficiency. Additionally, we discuss on feeding an initial feasible solution to FLOW, CPAXnY and CPAX. Providing initial solution improves only FLOW method. Despite the improvement, CPAXnY and CPAX still outperform it, significantly.

As a future research direction, MSelTP can be examined for different kinds of graph classes in terms of algorithms, formulations, and complexity. Moreover, a variant of MSelTP can be studied for forests instead of trees and this variant can be adapted to a generalized version of the feedback vertex set problem, which has not been studied to the best of our knowledge. Also, a variant of MSelTP where the clusters are not mutually exclusive can be discussed. In some cases, there are multiple optima for MSelTP. Thus, to find a minimum weighted maximum selective tree on a vertex-weighted or an edge-weighted graph can be studied, as well.

REFERENCES

1. Erdös, P. and Z. Palka, "Trees in Random Graphs", *Discret. Math.*, Vol. 46, pp. 145–150, 1983.

2. Karonski, M. and Z. Palka, "On the Size of a Maximal Induced Tree in a Random Graph", *Math. Slovaca*, Vol. 30, pp. 151–155, 1980.

3. Erdös, P., M. Saks and V. T. Sós, "Maximum Induced Trees in Graphs", *Journal of Combinatorial Theory, Series B*, Vol. 41, No. 1, pp. 61–79, 1986.

4. Chudnovsky, M. and P. D. Seymour, "The Three-in-a-tree Problem", *Combinatorica*, Vol. 30, pp. 387–417, 2010.

5. Derhy, N. and C. Picouleau, "Finding Induced Trees", *Discrete Applied Mathematics*, Vol. 157, pp. 3552–3557, 2009.

6. Rautenbach, D., "Dominating and Large Induced Trees in Regular Graphs", *Discrete Mathematics*, Vol. 307, No. 24, pp. 3177–3186, 2007.

7. Hertz, A., O. Marcotte and D. Schindl, "On the Maximum Orders of an Induced Forest, an Induced Tree, and a Stable Set", *Yugoslav Journal of Operations Research*, Vol. 24, pp. 199–215, 2014.

8. Melo, R. A. and C. C. Ribeiro, "Maximum Weighted Induced Forests and Trees: New Formulations and a Computational Comparative Review", *International Transactions in Operational Research*, Vol. 29, No. 4, pp. 2263–2287, 2022.

9. Fox, J., P.-S. Loh and B. Sudakov, "Large Induced Trees in K_r-free Graphs", *Journal of Combinatorial Theory, Series B*, Vol. 99, No. 2, pp. 494–501, 2009.

10. Beineke, L. W. and R. C. Vandell, "Decycling Graphs", *Journal of Graph Theory*,

Vol. 25, No. 1, pp. 59–77, 1997.

11. Li, G. and R. Simha, "The Partition Coloring Problem and its Application to Wavelength Routing and Assignment", *In Proceedings of the First Workshop on Optical Networks*, 2000.

12. Demange, M., T. Ekim, B. Ries and C. Tanasescu, "On Some Applications of the Selective Graph Coloring Problem", *European Journal of Operational Research*, Vol. 240, No. 2, pp. 307–314, 2015.

13. Pop, P., B. Hu and G. Raidl, "A Memetic Algorithm with Two Distinct Solution Representations for the Partition Graph Coloring Problem", *Revised Selected Papers of the 14th International Conference on Computer Aided Systems Theory - EUROCAST*, Vol. 8111, p. 219–226, Springer-Verlag, 2013.

14. Frota, Y., N. Maculan, T. F. Noronha and C. C. Ribeiro, "A Branch-and-cut Algorithm for Partition Coloring", *Networks: An International Journal*, Vol. 55, No. 3, pp. 194–204, 2010.

15. Demange, M., J. Monnot, P. Pop and B. Ries, "On the Complexity of the Selective Graph Coloring Problem in Some Special Classes of Graphs", *Theoretical Computer Science*, Vol. 540-541, pp. 89–102, 2014.

16. Furini, F., E. Malaguti and A. Santini, "An Exact Algorithm for the Partition Coloring Problem", *Computers & Operations Research*, Vol. 92, pp. 170–181, 2018.

17. Şeker, O., T. Ekim and Z. C. Taşkın, "A Decomposition Approach to Solve the Selective Graph Coloring Problem in Some Perfect Graph Families", *Networks*, Vol. 73, No. 2, pp. 145–169, 2019.

18. Şeker, O., T. Ekim and Z. C. Taşkın, "An Exact Cutting Plane Algorithm to Solve the Selective Graph Coloring Problem in Perfect Graphs", *European Journal of Operational Research*, Vol. 291, No. 1, pp. 67–83, 2021.

19. Demange, M., T. Ekim and B. Ries, "On the Minimum and Maximum Selective Graph Coloring Problems in Some Graph Classes", *Discrete Applied Mathematics*, Vol. 204, pp. 77–89, 2016.

20. Myung, Y. S., C. H. Lee and D. W. Tcha, "On the Generalized Minimum Spanning Tree Problem", *Networks*, Vol. 26, No. 4, pp. 231–241, 1995.

21. Feremans, C., M. Labbé and G. Laporte, "Generalized Network Design Problems", *European Journal of Operational Research*, Vol. 148, No. 1, pp. 1–13, 2003.

22. Pop, P., "The Generalized Minimum Spanning Tree Problem: An Overview of Formulations, Solution Procedures and Latest Advances", *European Journal of Operational Research*, Vol. 283, No. 1, pp. 1–15, 2020.

23. Reich, G. and P. Widmayer, "Beyond Steiner's Problem: A VLSI Oriented Generalization", *Graph-Theoretic Concepts in Computer Science*, Vol. 411 of *Lecture Notes in Computer Science*, pp. 196–210, Springer, Berlin, Heidelberg, 1990.

24. Ihler, E., G. Reich and P. Widmayer, "Class Steiner Trees and VLSI-design", *Discrete Applied Mathematics*, Vol. 90, No. 1, pp. 173–194, 1999.

25. Pop, P., W. Kern and G. Still, *The Generalized Minimum Spanning Tree Problem*, University of Twente, Department of Applied Mathematics, 2000.

26. Feremans, C., M. Labbé and G. Laporte, "A Comparative Analysis of Several Formulations for the Generalized Minimum Spanning Tree Problem", *Networks*, Vol. 39, pp. 29–34, 2002.

27. Feremans, C., M. Labbé, G. Laporte and E. Commerciales, "The Generalized Minimum Spanning Tree Problem: Polyhedral Analysis and Branch-and-cut Algorithm", *Networks*, Vol. 43, 2002.

28. Haouari, M. and J. Siala, "Upper and Lower Bounding Strategies for the Gener-

alized Minimum Spanning Tree Problem", *European Journal of Operational Research*, Vol. 171, pp. 632–647, 2006.

29. Golden, B., S. Raghavan and D. Stanojevic, "The Prize-collecting Generalized Minimum Spanning Tree Problem", *Journal of Heuristics*, Vol. 14, pp. 69–93, 2008.

30. Oncan, T., J.-F. Cordeau and G. Laporte, "A Tabu Search Heuristic for the Generalized Minimum Spanning Tree Problem", *European Journal of Operational Research*, Vol. 191, pp. 306–319, 2008.

31. Pop, P., O. Matei, C. Sabo and A. Petrovan, "A Two-level Solution Approach for Solving the Generalized Minimum Spanning Tree Problem", *European Journal of Operational Research*, Vol. 265, No. 2, pp. 478–487, 2018.

32. de Sousa, E. G., R. C. de Andrade and A. C. Santos, "A Multigraph Formulation for the Generalized Minimum Spanning Tree Problem", *ISCO 2018: Combinatorial Optimization*, Vol. 10856 of *Lecture Notes in Computer Science*, pp. 133–143, Springer, 2018.

33. Ferreira, C. S., L. S. Ochi, V. Parada and E. Uchoa, "A GRASP-based Approach to the Generalized Minimum Spanning Tree Problem", *Expert Systems with Applications*, Vol. 39, No. 3, pp. 3526–3536, 2012.

34. Duin, C., A. Volgenant and S. Voß, "Solving Group Steiner Problems as Steiner Problems", *European Journal of Operational Research*, Vol. 154, No. 1, pp. 323–329, 2004.

35. Noon, C. E. and J. C. Bean, "A Lagrangian Based Approach for the Asymmetric Generalized Traveling Salesman Problem", *Operations Research*, Vol. 39, No. 4, pp. 623–632, 1991.

36. Laporte, G. and Y. Nobert, "Generalized Travelling Salesman Problem through n

Sets of Nodes: An Integer Programming Approach", *INFOR: Information Systems and Operational Research*, Vol. 21, No. 1, pp. 61–75, 1983.

37. Laporte, G., A. Asef-Vaziri and C. Sriskandarajah, "Some Applications of the Generalized Travelling Salesman Problem", *The Journal of the Operational Research Society*, Vol. 47, No. 12, pp. 1461–1467, 1996.

38. Bondy, J. A. and U. S. R. Murty, *Graph Theory with Applications*, Elsevier Science Publishing Co., Inc., 1976.

39. Laporte, G. and F. V. Louveaux, "The Integer L-shaped Method for Stochastic Integer Programs with Complete Recourse", *Operations Research Letters*, Vol. 13, No. 3, pp. 133–142, 1993.

40. Dolan, E. and J. J. Moré, "Benchmarking Optimization Software with Performance Profiles", *Mathematical Programming*, Vol. 91, No. 2, p. 201–213, 2002.

41. De Marchi, A., "On a Primal-dual Newton Proximal Method for Convex Quadratic Programs", *Computational Optimization and Applications*, Vol. 81, pp. 369–395, 2022.

42. Diniz-Ehrhardt, M. A., D. G. Ferreira and S. A. Santos, "Applying the Pattern Search Implicit Filtering Algorithm for Solving a Noisy Problem of Parameter Identification", *Computational Optimization and Applications*, Vol. 76, p. 835–866, 2020.

43. Du, X., P. Zhang and W. Ma, "Some Modified Conjugate Gradient Methods for Unconstrained Optimization", *Journal of Computational and Applied Mathematics*, Vol. 305, pp. 92–114, 2016.

44. Audet, C., J. Bigeon, D. Cartier, S. Le Digabel and L. Salomon, "Performance Indicators in Multiobjective Optimization", *European Journal of Operational Research*, Vol. 292, No. 2, pp. 397–422, 2021.

45. Wächter, A. and L. T. Biegler, "On the Implementation of an Interior-point Filter Line-search Algorithm for Large-scale Nonlinear Programming", *Mathematical Programming*, Vol. 106, pp. 25–57, 2006.

46. Gill, P. E., W. Murray and M. A. Saunders, "SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization", *SIAM Review*, Vol. 47, No. 1, pp. 99–131, 2005.

47. Gören, M. and Z. Taşkın, "A Column Generation Approach for Evaluating Delivery Efficiencies of Collimator Technologies in IMRT Treatment Planning", *Physics in Medicine and Biology*, Vol. 60, pp. 1989–2004, 2015.