UNIVERSITÉ LIBRE DE BRUXELLES

Sciences

Computer Science Department

MEMO-F524

Master thesis

Resolution of the Minimum Connected Dominating Set Problem in Integer Programming

Lucie Bauwin

Lucie.Bauwin@ulb.ac.be

Promoter : **Bernard Fortz**

Master thesis

Master 2

Computer Science

Academic year 2019 - 2020

Contents

Abstract

Given a graph $G = (V, E)$, a dominating set D is a set of vertices such that any vertex not in D is connected to at least one vertex in D. Finding the set D with minimum cardinality such that the subgraph induced $G_D = (D, E(D))$ must be connected is known as the minimum connected dominating set problem. There is a need to find an efficient algorithm to solve the MCDSP as it is relevant to many applications. For example in wireless ad hoc network used in military applications, mobile network, vehicular and so on. This thesis will cover a state-of-the-art of different research made to solve such problems using a branch and cut algorithm and compare them through an implementation of the different formulation.

Introduction

A wireless ad hoc network is a multi-application decentralized wireless network. It is used in military operations, enabling on-the-move communication for deployment mission and also for the communication inter-ships in the navy. It is faster and not as limited as using satellite communications. What it does is to make every node in the network work as a router, where the node can move in any direction at any time, thus changing the relation between each nodes. Such procedures means that no third-parties need to intervene in the communication, only those concerned are part of the network. The need to design a dynamic routing protocol to find the virtual backbone is of course essential in this scenario. Smartphones are also influenced by those progresses with Apple starting to enable peer-to-peer network not depending on cellular carrier network, or traditional network structure. Finding this dynamic routing protocol can be reduced to the minimum connected dominating set problem known to be NP-hard.

There exist many approaches to solve the MCDS problem and this thesis will cover a few of them in order to compare them theoretically and experimentally. The aim of this thesis will be to introduce the MCDS problem as an integer program. Different formulations will be proposed in order to find the most efficient one when solved with a Branch and Cut algorithm.

In Chapter 1, a few definitions in graph theory are stated in order to understand the notions used in the paper and the integer programming formulation for the minimum dominating set problem is presented before proposing different approaches to represent the connectivity constraints for the MCDS problem with a small comparison of the models. In Chapter 2, an explanation of a branch and cut algorithm used to solve the different formulations is presented. Chapter 3 will explain the implementation using Cplex in python and present the results. Finally, Chapter 4 will discuss the results obtained experimentally with the results from the papers the formulations were found.

Chapter 1

Minimum Connected Dominating Set Problema

1.1 Definition and Formulation

To fully understand the concept of the minimum connected dominating set problem, let's first define different notions of graph theory using the following notation:

- $G = (V, E)$ an undirected graph
- V the set of vertices
- E the set of edges
- $E(S) := \{(i, j) \in E : S \subseteq V, i \in S, j \in S\}$ the set of edges of E with both endpoints in S.

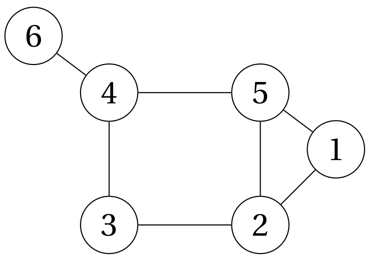


Figure 1.1 – An Undirected Graph

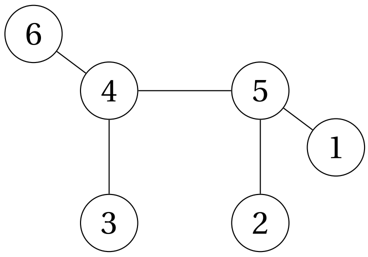


Figure 1.2 – Spanning tree of Figure 1.1

Looking at the example of an undirected graph in Figure 1.1, we can define the set $V = \{1, 2, 3, 4, 5, 6\}$, and the set $E = \{(1, 2), (1, 5), (2, 3), (2, 5), (3, 4), (4, 5), (4, 6)\}$. Let's define $S = \{1, 2, 4, 5\}$, then $E(S) = \{(1, 2), (1, 5), (2, 5), (4, 5)\}$

Definition 1 ***Connected Graph*** - *A graph $G = (V, E)$ is said to be connected if there exists a path between every pair of vertices. A graph that is not connected is called disconnected.*

Definition 2 ***Dominating Set*** - *Let $G = (V, E)$ be a graph and D a subset of the vertex set V such that each vertex $u \in V$ is either in D or adjacent to some vertex v in D. If there exists such a set D, it is called a dominating set. [18]*

Definition 3 ***Dominating Set Problem*** - *Given a graph G and an input K, find a dominating set D respecting the condition $|D| \leq K$.*

Definition 4 ***Minimum Dominating Set (MDS)*** - *A Minimum Dominating Set for a graph $G = (V, E)$ is the dominating set $D \subset V$ with the smallest cardinality.*

Definition 5 ***Minimum Dominating Set Problem*** - *Given a graph G, find the dominating set D with the smallest cardinality. The decision problem of MDS is known to be NP-complete.*

Looking back to the undirected graph in Figure 1.1, a dominating set could be $\{1, 2, 6\}$ but a minimum dominating set could be the $\{2, 6\}$. We still need to introduce the connectivity property.

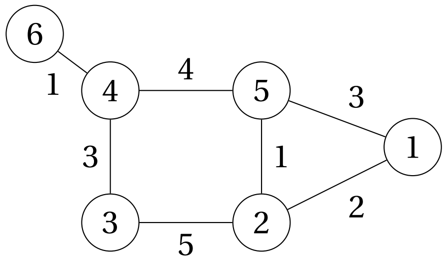


Figure 1.3 – An Undirected weighted Graph

Definition 6 ***Connected Dominating Set (CDS)*** *- A connected dominating set of a graph $G = (V, E)$ is a set of vertices with two properties: [18]*

1. *D is a dominating set in G.*
2. *D induces a connected sub-graph of G.*

Definition 7 ***Minimum Connected Dominating Set (MCDS)*** *- A Minimum Connected Dominating Set is a connected dominating set with the smallest possible cardinality among all the CDSs of G. [18]*

Still in the same undirected graph, we can find a minimum connected dominating set: $\{4, 5\}$.

Definition 8 ***Spanning Tree*** *- A graph is said to be a spanning tree $T = (V', E')$ of a graph $G = (V, E)$, if T contains all vertices of G ($V' = V$), $E' \subseteq E$, $|E'| = |V| - 1$ and T is connected and acyclic.*

A spanning tree of our graph is composed of the edges $\{(1, 5), (2, 5), (3, 4), (4, 5), (4, 6)\}$ as we can see in Figure 1.2.

Definition 9 ***Minimum Spanning Tree Problem*** *- Find the tree that contains all nodes of the weighted graph G and that minimizes the sum of the weighted edges of the tree. [14]*

If we add some weight to the edges of our graph such that it corresponds to Figure 1.3, we get the minimum spanning tree formed with the set of edges: $\{(1, 2), (2, 5), (3, 4), (4, 5), (4, 6)\}$

Definition 10 ***Characteristic Vector*** - *A characteristic vector of a subset* $T \subseteq V$ *is the vector* χ_T *of length* $|V|$ *where* $x_i = 1$ *if* $i \in T$ *and* $x_i = 0$ *if* $i \notin T$*.*

Take our graph, and let's say we take a subset T of V, where $T = \{1, 4, 5\}$, then $\chi_T = [1, 0, 0, 1, 1, 0]$.

Definition 11 ***Tree Polytope*** - *The tree polytope of* G *is the convex hull of the characteristic vectors of the trees of* G*. [8]*

Theorem 1 ***Spanning Tree Polytope*** - *Let* $G = (V, E)$ *be a connected (simple, finite, undirected) graph. The spanning tree polytope of* G *is the convex hull of the characteristic vectors of spanning trees of* G*. We denote this polytope as* $P_{sp.trees}(G)$*, and use the notation [10]*

$$P_{sp.trees}(G) = conv\{\chi^T \in \{0, 1\}^E | T \subseteq E, T \text{ spanning tree of } G\}$$

where χ^T *is the characteristic vector of* $T \subseteq E$*.*

Let's introduce the integer programming formulation for the simple Minimum Dominating Set problem as presented in [9]:

Let $G = (V, E)$ with $V = \{1, 2, ..., n\}$ be a graph,

and let $A = (a_{ij})_{n \times n}$ be the neighborhood matrix such that $a_{ij} = a_{ji} = 1$ if $(i, j) \in E$ or $i = j$, and $a_{ij} = a_{ji} = 0$ otherwise.

The edge set is defined as: $E : \{(i, j) = a_{ij} = 1, \forall i, j \in V$ with $i < j\}$.

For $i \in V$, let $x_i \in \{0,1\}$ be a decision variable such that $x_i = 1$ if vertex i is included in the dominating set; $x_i = 0$ otherwise.

$$\min \sum_{i \in V} x_i \tag{1.1a}$$

$$s.t. \sum_{j \in V} a_{ij} x_j \geq 1 \tag{1.1b}$$

$$x_i \in \{0,1\}, \qquad \forall i \in V \tag{1.1c}$$

Any feasible solution will form a dominating set D of G with $D = \{i : x_i = 1\}$. Let $G_D = (D, E_D)$ be the subgraph induced by D, where $E_D = \{(i,j) \in E : a_{ij}x_ix_j = 1, \forall i,j \in V \text{ with } i < j\}$.

Now that this simple system is defined, we need to add the constraints for the connectivity of the subgraph G_D to be able to formulate an IP for the MCDS problem.

Definition 12 ***Minimum Connected Dominating Set Problem*** *- Given a graph G, find the minimum dominating set D such that the vertices present in D are connected.*

There exist many formulations to represent the connectivity of the dominating set in a graph and thus solve the MCDS problem. Some of these solutions will be presented in the next section.

1.2 Optimization models for connectivity

The MCDSP can be presented as follow [21]:

Let $G = (V, E)$ be a connected undirected graph where V the set of vertices with $|V| = n$ and E the set of edges with $|E| = m$,

Given $i \in V$, assume that $\Gamma_i \subseteq V$ is the union of $\{i\}$ with the set of vertices of V that share an edge with i.

Given any set D, let $E(D) = \{(i, j) \in E : i, j \in D\}$ be the subset of edges of E with both endpoints in D.

The set D is connected if the subgraph $G_D = (D, E(D))$ of G is connected, or in other words, if there exists a spanning tree in G_D.

This gives all connected dominating set of G, there is still the need to find the set with minimum cardinality, this problem has been proven to be NP-hard. As said before, to prove that a graph is connected, we can check if a spanning tree exists. Therefore, to find the MCDS of G, we can find the minimum spanning tree of G_D, multiple formulations of the problem are based on the link between the minimum spanning tree and MCDS.

Four Integer Programming formulations to solve the MCDSP will be presented, the one proposed in [21] by Simonetti, Salles da Cunha and Lucena, and three from [9]: the Miller-Tucker-Zemlin constraints, the Martin constraints and the single commodity flow constraints.

1.2.1 Simonetti-Salles da Cunha-Lucena approach

The main idea behind the formulation is to find a spanning tree in the dominating set. If no spanning tree can be found, then it means that the dominating set is not connected and thus either there are no connected dominating set in the graph or we need to choose another set.

The IP formulation introduced by Simonetti, Salles da Cunha and Lucena [21] uses the following decision variables:

- $x_i \in \{0, 1\}$, $i \in V$ which will represent the vertices contained in the dominating set. If $x_i = 1$, the vertex i is part of the dominating set, $x_i = 0$ otherwise.
- $y_{ij} \in \{0, 1\}$, $(i, j) \in E$, $i, j \in V$ which will choose the edges ensuring that

the dominating set is connected.

Assume that $\mathbb{B} = \{0, 1\}$ and $\mathbb{R}$ the set of real numbers. The formulation of the problem using these variables is:

$$\min \quad \left\{ \sum_{i \in V} x_i : (x, y) \in \mathscr{R}_0 \cap (\mathbb{R}^m_+, \mathbb{B}^n) \right\} \tag{1.2}$$

where $\mathscr{R}_0$ is the polyhedral region implied by:

$$\sum_{(i,j) \in E} y_{ij} = \sum_{i \in V} x_i - 1 \tag{1.3a}$$

$$\sum_{(i,j) \in E(S)} y_{ij} \leq \sum_{i \in S \setminus \{j\}} x_i, \qquad S \subset V, j \in S \tag{1.3b}$$

$$\sum_{j \in \Gamma_i} x_j \geq 1, \qquad i \in V \tag{1.3c}$$

$$y_{ij} \geq 0, \qquad \forall (i, j) \in E \tag{1.3d}$$

$$0 \leq x_i \leq 1, \qquad \forall i \in V \tag{1.3e}$$

$$y_{i,j} \leq x_i, \; y_{i,j} \leq x_j \qquad \forall (i, j) \in E \tag{1.3f}$$

The variable y is here to ensure that a spanning tree exists in the obtained subgraph $G_D = (D, E(D))$, thus that the subgraph is connected. The constraint (1.3a) ensures that there will be exactly one unit less in the edges set $E(D)$ of the subgraph compared to the vertices set D. Constraints (1.3b) are the Generalized Subtour Breaking Constraints ensuring that the set $E(D)$ implies a tree structure. The constraints (1.3c) are making sure that the resulting graph G_D is dominating by checking that each vertex is either in D or adjacent to a vertex present in D.

The polyhedral region encloses two structures: the constraints (1.3c) and (1.3e) are used to represent the *Covering Problem*[1]; the constraints (1.3a), (1.3b),

[1]Computational problems that ask whether a certain combinatorial structure covers an-

(1.3d) - (1.3e) define the tree polytope of G.

There exists a strengthened version of this formulation by lifting and replacing some constraints. We keep the constraints (1.3a), (1.3b), (1.3d) and (1.3e) and we add the following constraints resulting of modifications on (1.3c):

$$\sum_{j\in\Gamma_i} x_j - \sum_{(i,j)\in E(\Gamma_i)} y_{ij} \geq 1 \qquad \forall i \in V \tag{1.4a}$$

$$\sum_{(i,j)\in E(C)} y_{ij} \leq \sum_{j\in C} x_j - 1 \qquad C \subset V \tag{1.4b}$$

$$\sum_{(i,j)\in(S,V\backslash S)} y_{ij} \geq 1, \qquad \forall S \subset V : \Gamma(S) \neq V, \Gamma(V\backslash S) \neq V \tag{1.4c}$$

The constraints (1.4a) are obtained by lifting (1.3c). It is a strengthened version of the Generalized Subtour Breaking Constraints. The constraint (1.4b) is a lifting of (1.3b). Assume that, for any given way, we can certify that, out of those vertices in a particular given set $C \subset V$, at least one vertex must be included in a connected dominating set. The last constraint is the mathematical result of the following reasoning: assume that given $S \subset V, S \neq$, $\Gamma(S) := \bigcup_{i\in S} \Gamma_i$ and that $(S, V\backslash S) := \{(i,j) \in E : i \in S, j \notin S\}$ denotes the edges in the cut implied by S. Whenever $\Gamma(S) \neq V$ and $\Gamma(V\backslash S) \neq V$, at least one edge in $(\Gamma(S), V\backslash\Gamma(S))$ must be chosen. More explanation on the calculation can be found in the original article [21].

The resulting strengthened formulation $\mathscr{R}_1$ for the MCDSP is composed of the constraints (1.3a), (1.3b), (1.3d), (1.3e), (1.4a), (1.4b), (1.4c).

This results in an exponential number of constraints $O(2^{|V|})$ to solve the problem. Using an efficient branch and cut algorithm could reduce the number of constraints and thus be more efficient.

---

other, or how large the structure has to be to do that.[22]

1.2.2 Martin approach

The connectivity constraints are based on the article written by Martin [16], where he formulated a polynomial number of constraints to solve the minimum spanning tree problem rather than an exponential number of constraints. The constraints have then been used in [9] to propose connectivity constraints for the MCDSP. The approach is based on finding the spanning tree $T_D = (D, E_T)$ of the subgraph $G_D = (D, E_D)$. The formulation of the minimum spanning tree problem is:

Definition 13 ***Minimum Spanning Tree problem formulation*** *- Let y_{ij} be 1 if edge (i, j) is in the tree T. We need to ensure that there will be $n-1$ edges in T*

- $\sum_{ij \in E} y_{ij} = n - 1$

and that there are no cycle in T (Subtour Elimination constraint):

- $\sum_{ij \in E: i \in S, j \in S} y_{ij} \leq |S| - 1, \; \forall S \subseteq V$

We define the Martin connectivity constraints for the MCDS problem with the help of the following decision variables:

- $x_i \in \{0, 1\}$ for $i \in V$ such that $x_i = 1$ if the vertex i is part of the dominating set and $x_i = 0$ otherwise.
- $y_{ij} \in \{0, 1\}$ for $i, j \in V$, such that $y_{ij} = 1$ if the edge (i, j) is in T_D and $y_{ij} = 0$ otherwise.
- $z_{ij}^k \in \{0, 1\}$ for $i, j, k \in V$, such that $z_{ij}^k = 1$ if it respects two conditions:
  1. edge (i, j) is in the tree T_D of G_D
  2. vertex k is on side of j meaning that k is inside the same connected component as j after the removal of the edge (i, j) from T_D.

  If one or both conditions are not respected then $z_{ij}^k = 0$.

We define M as a large positive constant.

The objective function and dominating constraints are the ones introduced in the end of the Section 1.1.

$$\min \sum_{i \in V} x_i \tag{1.5}$$

$$s.t. \sum_{j \in V} a_{ij} x_j \geq 1 \tag{1.6}$$

$$x_i \in \{0,1\}, \qquad \forall i \in V \tag{1.7}$$

The Martin connectivity constraints are:

$$\sum_{(i,j) \in E} y_{ij} = \sum_{i \in V} x_i - 1 \tag{1.8a}$$

$$y_{ij} \leq x_i, y_{ij} \leq x_j, \qquad \forall (i,j) \in E \tag{1.8b}$$

$$z_{ij}^k \leq y_{ij}, z_{ij}^k \leq x_k, \qquad \forall (i,j) \in E, k \in V \tag{1.8c}$$

$$z_{ji}^k \leq y_{ij}, z_{ji}^k \leq x_k, \qquad \forall (i,j) \in E, k \in V \tag{1.8d}$$

$$y_{ij} - M(3 - x_i - x_j - x_k) \leq z_{ij}^k + z_{ji}^k \leq y_{ij} + M(3 - x_i - x_j - x_k), \forall i,j,k \in V \tag{1.8e}$$

$$1 - M(2 - x_i - x_j) \leq \sum_{k' \in V \setminus \{i,j\}} z_{ik'}^j + y_{ij} \leq 1 + M(2 - x_i - x_j), \forall i,j \in V \tag{1.8f}$$

$$y_{ij}, z_{ij}^k \in \{0,1\}, \forall (i,j) \in E, k \in V, y_{ij} = 0, z_{ij}^k = 0, \forall i,j,k \in V, (i,j) \notin E \tag{1.8g}$$

The first constraint (1.8a) is the same as (1.3a) presented in the Simonetti-Salles da Cunha-Lucena formulation, it ensures that the number of edges in T_D is equal to the number of vertices in T_D minus 1. Constraint (1.8b) ensures that the edges in E_D are in fact connecting two vertices in D. The constraints

(1.8c), (1.8d) and (1.8e) are linked, the first two ensure that if any one, two or three vertices $i, j, k \in V$ are not part of the tree of G_D , then $z_{ij}^k = z_{ji}^k = 0$ and the constraint (1.8e) is non-binding and does not influence the result since M is a large positive constant. Likewise, if any one or two of vertices $i, j \in V$ are not in G_D, the constraint (1.8f) is non-binding. On the contrary, if $i, j, k \in V$ are in the spanning tree of G_D, then by constraints (1.8c)-(1.8d) and (1.8e)-(1.8f), $z_{ij}^k, z_{ji}^k \in \{0, 1\}$ become

$$z_{ij}^k + z_{ji}^k = y_{ij}, \sum_{k \in D \setminus \{i,j\}} z_{ik}^j + y_{ij} = 1, \qquad \forall i, j, k \in D. \tag{1.9}$$

The first part hints that:

- if $(i, j) \in E_T$, then the vertex k is either on the side of i ($z_{ji}^k = 1$) or j ($z_{ji}^k = 1$),
- if $(i, j) \notin E_T$ then k is between i, j.

The second part of the constraint ensures that:

- if $(i, j) \in E_T$, edges (i, k) who connect i are on the side of i ($z_{ik}^j = z_{ij}^k = 0$ and $z_{ij}^k = 1$),
- if $(i, j) \notin E_T$, there has to be an edge (i, k) such that j is on the side of k.

The formulation Martin has $|V|^2 + |V|^3 = O(|V|^3)$ decision variables and $1 + 2|E| + 4|E||V| + 2|V|^3 + 2|V|^2 = O(|V|^3)$ constraints to ensure connectivity.

1.2.3 Single-Commodity Flow Approach

This formulation has been influenced by the research done in [7] for the Connected Subgraph Problem in Wildlife preservation and presented in [9]. Once again the dominating set is chosen with the dominating constraints introduced in Section 1.1.

The idea for the connectivity is to choose arbitrarily a vertex as the root inside the dominating set. The root will be responsible for sending $\sum_{i \in V} x_i - 1$ unit flow to the vertices in D, the vertices will consume exactly one unit flow, thus if all units are consumed, the connectivity of G_D is ensured.

There are two other decision variables other than x_i and a_{ij}:

$r_i \in \{0, 1\}$ for $i \in V$, $r_i = 1$ if i is the chosen root of G_D and $r_i = 0$ otherwise.

f_{ij}, the number of flow from i to j. For each edge $(i, j) \in E \cup E'$ and $E' = \{(j, i) : a_{ij} = 1, \forall i, j \in V$ with $i > j\}$.

So we have 3 sets of decision variables, r_i, f_{ij}, x_i with $i \in V$ and $(i, j) \in E$ for the connectivity constraints.

The single-commodity flow constraints are:

$$\sum_{i \in V} r_i = 1 \tag{1.10a}$$

$$r_i \leq x_i, \qquad \forall i \in V \tag{1.10b}$$

$$f_{ij} \geq 0, \qquad \forall (i, j) \in E \cup E' \tag{1.10c}$$

$$f_{ij} \leq x_i \sum_{k \in V} x_k, f_{ij} \leq x_j \sum_{k \in V} x_k, \qquad \forall (i, j) \in E \cup E' \tag{1.10d}$$

$$\sum_j f_{ji} \leq n(1 - r_i), \qquad \forall i \in V \tag{1.10e}$$

$$\sum_j f_{ji} - \sum_j f_{ij} = x_i - r_i \sum_{j \in V} x_j, \qquad \forall i \in V \tag{1.10f}$$

$$r_i \in \{0, 1\}, \qquad \forall i \in V \tag{1.10g}$$

The first two constraints (1.10a) and (1.10b) ensure that there is only one root

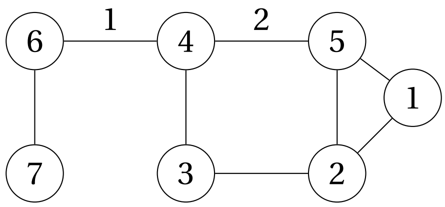


Figure 1.4 – Say $r = 5$ and the dominating set is $\{4, 5, 6\}$, r sends 2 unit of flow to node 4 and the flow continues to node 6.

selected among all vertices and that it is selected to be in the dominating set. The flow needs to be sent between vertices present in the dominating set and each vertex consumes exactly one unit of flow, so the flow is in $\{0, 1\}$. The constraint (1.10c) ensures the non-negativity of the flow, and (1.10d) ensures that the flow of an edge (i, j) is 0 if not both vertices of the edge are in the dominating set. Constraint (1.10e) makes sure that no flow goes into the root. The equality of the inflow and outflow of each vertex is ensured with constraint (1.10f). There are three cases:

- either vertex i is the root, then the outflow is equal to $\sum_{j \in V} x_j - 1$, so the flow is consumed by the vertices in D.
- Or vertex i is in the dominating set but is not the root, then the equality will be one since the difference between inflow and outflow will be one, meaning that the vertex i consume one flow unit,
- or finally vertex i is not in D and it will be equal to 0.

To suppress the quadratic dimension brought by constraint (1.10f) and (1.10d), we can introduce two new decision variables to make it linear:

- $w_{ij} = r_i x_j$ with sets of constraints:
  - $w_{ij} \leq r_i$
  - $w_{ij} \leq x_j$
  - $w_{ij} \geq r_i + x_j - 1$
  - $w_{ij} \geq 0$

- $w'_{ij} = x_i x_k$ with similar constraints.

We can add a constraint to make the first vertex appearing in D the root, it can *"reduce the degeneracy of the choice of root vertex within the dominating set"*:

$$r_i \leq (n + 1 - \sum_{i'=1}^{i} x_{i'})/n \qquad \forall i \in V \tag{1.11}$$

$r_i = 1$ for the node with the smallest index present in D.

The single-commodity flow formulation has $|V| + 2|E| = O(|E| + |V|)$ decision variables and $1 + |V| + 2|E| + 4|E| + |V| + |V| = O(|E| + |V|)$ constraints for the MDS problem with single-commodity flow constraints.

1.2.4 Miller-Tucker-Zemlin approach

This procedure is proposed in [9], with the method proposed in [19] to modify the undirected graph $G = (V, E)$ into a directed graph G_d.

Let $G_d = (V \cup \{n+1, n+2\}, A)$ be a directed graph based on $G = (V, E)$, where:

$A = \{(n+1, n+2)\} \cup \left\{ \bigcup_{i=1}^{n} \{(n+1, i), (n+2, i)\} \right\} \cup E \cup E'$ and,

$E' = \{(j, i) : a_{ij} = 1, \forall i, j \in V$ with $i > j\}$.

The transformation of G into G_d adds 2 new vertices $n + 1,\ n + 2$ to the directed graph and edges from $n + 1$ and $n + 2$ to every $i \in V$ as well as the edge $(n + 1, n + 2)$. Additional directed edges are added between every pair of vertices (i, j) where $a_{ij} = 1$ so that it becomes bi-directional.

The idea is to find a directed spanning tree $T_d = (V \cup \{n + 1, n + 2\}, E_d)$ of G_d. At the end of the process, $n + 1$ will be connected to $n + 2$ and all vertices not part of the dominating set and, $n + 2$ will be connected to a vertex considered

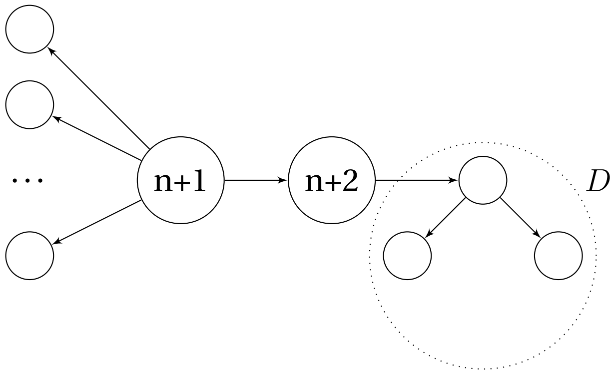


Figure 1.5 – Visual solution for MTZ

as the root of the tree representing the dominating set.

This approach uses the following decision variables:

- $y_{ij} \in \{0, 1\}$ for $i, j \in A$, such that $y_{ij} = 1$ if the edge (i, j) is selected to be in E_d and $y_{ij} = 0$ otherwise.
- $u_i \in \mathbb{R}_+$ for $i \in V \cup \{n+1, n+2\}$ indicates the number of arcs from $n+1$ to i.

Once again, the constraints to ensure a dominating set are the ones presented in section 1.1. The Miller-Tucker-Zemlin connectivity constraints are:

$$\sum_{i \in V} y_{n+2,i} = 1 \tag{1.12}$$

$$\sum_{i:(i,j) \in A} y_{ij} = 1, \qquad \forall j \in V \tag{1.13}$$

$$y_{n+1,i} + y_{i,j} \leq 1, \qquad \forall (i,j) \in E \cup E' \tag{1.14}$$

$$(n+1)y_{ij} + u_i - u_j + (n-1)y_{ji} \leq n, \qquad \forall (i,j) \in E \cup E' \tag{1.15}$$

$$(n+1)y_{ij} + u_i - u_j \leq n, \qquad \forall (i,j) \in A \setminus (E \cup E') \tag{1.16}$$

$$y_{n+1,n+2} = 1 \tag{1.17}$$

$$u_{n+1} = 0 \tag{1.18}$$

$$1 \leq u_i \leq n+1, \qquad i \in V \cup \{n+2\} \tag{1.19}$$

$$x_i = 1 - y_{n+1,i}, \qquad \forall i \in V \tag{1.20}$$

The constraint (1.12) chooses a vertex from the dominating set to be the root. Constraints (1.13) make sure that all vertices from V are connected to another vertex in A. Constraints (1.14) ensure that all vertices in V are either connected to the added vertex $n + 1$ or it may be connected to another vertex in the dominating set. Constraints (1.15) and (1.16) are the MTZ constraints ensuring that the solution have no sub-tours. Constraint (1.17) requires that vertex $n + 1$ is connected to $n + 2$ in T_d. Constraints (1.18) and (1.19) represent the choice of arbitrary non-negative integers for u_i. Finally, the last constraint (1.20) requires vertex i to either be connected to $n + 1$ or be a part of the dominating set.

This results in $(|V| + 2) + (2|E| + 2|V| + 1) = O(|E| + |V|)$ decision variables and $1 + |V| + 2|E| + 2|E| + (2|V| + 1) + 1 + 1 + |V| = O(|E| + |V|)$ constraints for the MTZ formulation.

1.3 Comparison

We can note that SSL and Martin have a better linear relaxation than SCF and MTZ but their formulation size is bigger than SCF and MTZ (Table 1.1) which could be a disadvantage when computing the results. The results will be presented in Chapter 3.

Table 1.1 – Formulation sizes

Formulation	decision vars	constraints
SSL	$O(\|E\| + \|V\|)$	$O(2^{\|V\|})$
MARTIN	$O(\|V\|^3)$	$O(\|V\|^3)$
SCF	$O(\|E\| + \|V\|)$	$O(\|E\| + \|V\|)$
MTZ	$O(\|E\| + \|V\|)$	$O(\|E\| + \|V\|)$

Chapter 2

Algorithm

The different formulations prensented in Section 1.2 have been solved using a Branch and Cut algorithm present in Cplex. The Branch and Cut algorithm is a mix of the Branch and Bound algorithm and the Cutting Planes method. The mechanics of the two algorithms and how they combine together will be explained before presenting the results of the simulations.

2.1 Branch and Bound algorithm

This algorithm consists of a divide and conquer strategy, this is a basic idea to resolve Integer Programming problems, and the bounding principles.

2.1.1 Divide and Conquer

First we consider the divide and conquer strategy, it decomposes a problem into sub-problems that are easier to solve.
Consider the problem:

$$z = \max\{cx : x \in S\}$$

Since S is the feasible region of the problem, we will divide it into smaller subsets or subproblems.

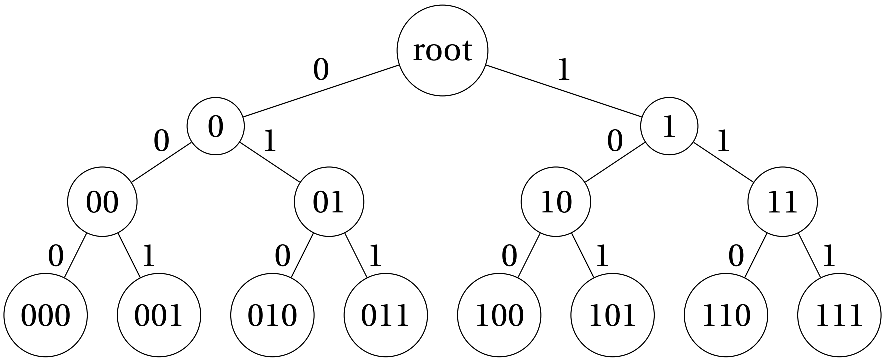


Figure 2.1 – Binary enumeration tree

Proposition 1 *Let $S = S_1 \cup S_2 \cup ... \cup S_K$ be a decomposition of S into smaller sets, and let $z^k = \max\{cx : x \in S_k\}$ for $k = 1, ..., K$. Then $z = \max_k z^k$ [23].*

A subproblem is constructed by deciding for a subset A of the variables, that these variables must be included in the in the solution constructed.

Definition 14 ***A branch** is the creation of two new nodes from a parent node. It occurs when the bounds on a single variable are modified, with the new bounds remaining in effect for that new node and for any of its descendants [5].*

A representation of this procedure is with an enumeration tree such that if $S \subseteq \{0,1\}^3$, we will obtain the enumeration tree as represented in Figure 2.1. We first divide S into $S_0 = \{x \in S : x_1 = 0\}$ and $S_1 = \{x \in S : x_1 = 1\}$, then $S_{00} = \{x \in S_0 : x_2 = 0\}$, $S_{01} = \{x \in S_0 : x_2 = 1\}$ and so on until we obtain all leaves.

Of course, complete enumeration is impossible when reaching a high number of variables as it would not be space efficient nor time efficient. We thus introduce some bounds to apply the bounding principle.

2.1.2 Bounding principle

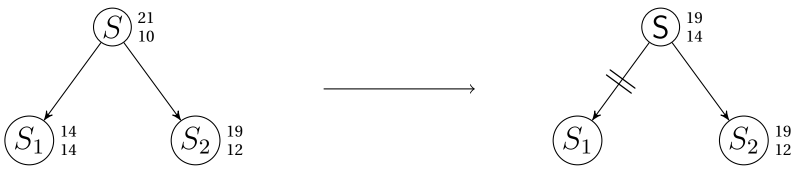
A simple principle from optimization that is very intuitive. Let S_1 and S_2 be two feasible set of a maximisation problem. If the upper bound of S_1 is lower

than the lower bound of S_2, then S_1 is not worth exploring and can be discarded for the rest of the algorithm.
The bounds are introduced to the tree using the following proposition:

Proposition 2 *Let $S = S_1 \cup ... \cup S_k$ be a decomposition of S into smaller sets, and let $z^k = \max\{cx : x \in S_k\}$, for $k = 1, ..., K$, $\bar{z}^k$ be an upper bound on z^k and $\underline{z}^k$ be a lower bound on z^k. Then $\bar{z} = max_k \bar{z}^k$ is an upper bound of z and $\underline{z} = \max_k \underline{z}^k$ is a lower bound on z [23].*

Those bounds are used to prune the tree, there are three cases in which we can cut part of the tree:

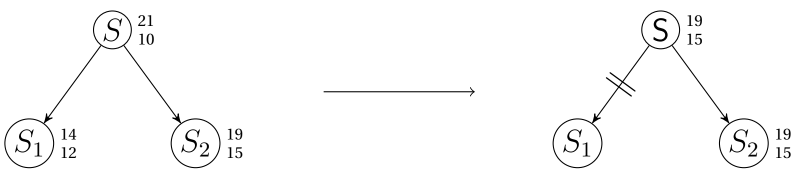
- pruned by optimality : $z_t = \{\max cx : x \in S_t\}$ has been solved. On Figure 2.2, we observe that the upper bound and lower bound of S_1 are equals, meaning that the optimal solution for that branch has already been found and we should not explore it further. Thus we prune the branch S_1



(a) (b)

Figure 2.2 – Pruned by optimality

- pruned by bound : $\bar{z}_t \leq \underline{z}$. On Figure 2.3, the upper bound of S_1 being lower than the lower bound of S_2, we prune the branch S_1 as the optimal solution cannot be better than the branch S_2.



(a) (b)

Figure 2.3 – Pruned by Bound

- pruned by infeasibility : $S_t = \phi$

The bounds are retrieved with the help of the feasible solutions for the primal/lower bound and by relaxation or duality for the dual/upper bound.

Definition 15 ***Linear programming relaxation*** *- It consists of dropping the integrality constraints on the decision variables.*

2.1.3 Main components of Branch and Bound

There are three main components to a good Branch and Bound algorithm for a maximisation problem [3]:

1. A *bounding function* to set the upper bound by solving the linear relaxation of the problem. If a feasible solution exists, take it as the lower bound, if there is none, use $\underline{z} = -\infty$.

2. A *Strategy for selecting* a node. Choosing arbitrarily a node from the set of active nodes[1]. Though it is better to use a strategy to choose the next node such as *Depth-First Search* strategy, *Breadth First* strategy or *Best-Node First* strategy.

3. A *Branching rule* to choose an integer variable that is basic and fractional and split the problem in two based on the fractional value. This can be done by adding constraints of the form of variable assignment. When the division creates two new branches, we talk about *dichotomic* branching, otherwise we talk about *polytomic* branching.

$$S_1 = S \cap \{x : x_j \leq \lfloor \bar{x}_j \rfloor\}$$

$$S_2 = S \cap \{x : x_j \geq \lceil \bar{x}_j \rceil\}$$

For an efficient branch and bound algorithm, we need a good preprocessing as a first step to tighten the formulation.

[1] the nodes that still need to be analyzed.

2.2 Cutting Plane method

The cutting plane method consists of finding the optimum solution of an integer program by using its linear relaxation program and by adding constraints into this linear program until an optimal integer solution is found.

Consider the integer program:

$$\max\{cx : x \in X\}$$

where $X = \{x : Ax \leq b,\ x \in \mathbb{Z}^n_+\}$.

Proposition 3 $conv(X) = \{x : \tilde{A}x \leq \tilde{b}, x \geq 0\}$ *is a polyhedron [23].*

And its linear relaxation:

$$\max\{cx : \tilde{A}x \leq \tilde{b},\ x \geq 0\}$$

To try and find an approximation of $conv(X)$ for a given instance in an effective way, we need to use the concept of valid inequality that will help strengthen the IP formulations.

2.2.1 Valid Inequalities

Definition 16 *An inequality* $\pi x \leq \pi_0$ *is a **valid inequality** for* $X \subset \mathbb{R}^n$ *if* $\pi x \leq \pi_0$ *for all* $x \in X$.

For example, if $X = \{x \in \mathbb{Z}^n : Ax \leq b\}$, then all of the constraints $a_i^T x \leq b_i$ are valid inequalities for X and if $conv(X) = \{x \in \mathbb{R}^n : \tilde{A}x \leq \tilde{b}\}$, all of the constraints $\tilde{a}_i^T x \leq \tilde{b}_i$ are valid inequalities for X.

The idea behind this method is to identify additional valid inequalities for X and add them to the formulation, those constraints will "cut" away regions that contain no feasible solutions therefore strengthening the formulation for X.

Proposition 4 *Let $X = \{y \in \mathbb{Z} : y \leq b\}$, then the inequality $y\lfloor b \rfloor$ is valid for x.*

From this proposition, we can deduce that it is possible to finitely generate the convex hull with valid inequalities.

Chvàtal-Gomory cuts procedure

Consider a linear program $\max\{cx|Ax \leq b, x$ integer$\}$, A the matrix of size $m \times n$ with columns $\{a_1, a_2, ..., a_n\}$.

We have the inequality $\sum_{j=1}^{n} a_j xj \leq b$.

For any real $\lambda > 0$, the inequality $\sum_{j=1}^{n} \lambda a_j x_j \leq \lambda b$ is valid.

Thus, we can deduce that for $x \geq 0$, $\sum_{j=1}^{n} \lfloor \lambda a_j \rfloor x_j \leq \lambda b$ is valid.

The Chvàtal-Gomory cut is therefore:

$$\sum_{j=1}^{n} \lfloor \lambda a_j \rfloor x_j \leq \lfloor \lambda b \rfloor$$

It is valid as x is integer, and thus $\sum_{j=1}^{n} \lfloor \lambda a_j \rfloor x_j$ is integer.

Theorem 2 *Every valid inequality for X can be obtained by applying the Chvàtal-Gomory procedure a finite number of times.*

This procedure may be sufficient for small problems but for bigger problems, too many cuts can be generated to include them from the start, we will thus find a way to include them recursively as we need them.

2.2.2 Generic Cutting Plane Algorithm

For the problem $\max\{cx : x \in P, x$ integer$\}$ and the family $\mathcal{F}$ of valid inequalities, the algorithm works as follow [23]:

1. Initialize the number of iteration $t = 0$ and the initial formulation $P^0 = P$
2. At iteration t:
   (a) Solve the linear program $\bar{z}^t = \max\{cx : x \in P^t\}$
   (b) We obtain the optimal solution x^t for the linear relaxation, if $x^t \in \mathbb{Z}^n$, the solution is integer and optimal. If $x^t \notin \mathbb{Z}^n$, solve the separation problem for x^t and the family F. If an inequality present in $\mathcal{F}$ is found such that it cuts off x^t, set $P^{t+1} = P^t \cap \{x : \pi^t x \leq \pi_0^t\}$ and augment t. Otherwise stop.
3. If the algorithm terminates without finding an integral solution for IP, then $P^t = P \cap \{x : \pi^i x \leq \pi_0^i \; i = 0, ..., t\}$ is an improved formulation that can be sent into an efficient Branch and Bound algorithm as input.

Adding cuts recursively may result in a very large linear program causing some numerical difficulties for an LP solver. Also the cutting plane method does not find any feasible solution until it finds the optimal solution, so if the computation is stopped before the end, there would not be any integer solution to offer. Those two points make this method unreliable.

Let's look at an example of a cut by looking at the following constraint with three binary variables:

$$20x + 25y + 30z <= 40$$

We can use a strengthened version of this constraint by adding a cut where no integer solutions are dismissed but the fractional solution will:

$$1x + 1y + 1z <= 1$$

This cut reduces the amount of searching needed.

The cutting plane method is fast but unreliable, as previously mentioned there may be some numerical difficulties or no intermediate feasible solution found, unlike the branch and bound algorithm which is reliable but slow due to the fact that there may be many node to be explored in order to find the optimum solution but even if stopped prematurely, in most cases there will be a feasible solution to offer at the end of the algorithm.

2.3 Branch and Cut

The branch and cut method is a mix between the branch and bound algorithm and the cutting plane method. It takes advantage of the rapidity of the cutting plane method and the reliability of the branch and bound algorithm resulting in a reliable algorithm faster than branch and bound alone.

The goal will be to use the cutting plane algorithm as preprocessing on the model before passing the result to the branch and bound algorithm, then the nodes of the branch and bound tree may call the cutting plane method to tighten their dual bound.

There exists several variations of the algorithm, the cutting planes could be used in one or more spots in the process and the condition to do so can also be different. The use of different variants of the algorithm can of course alter the results. For this section, I will focus on the way Cplex implement their Branch and Cut based on the information in [6].

The procedure of the Branch and Cut is to first only take into account the root node of the Branch and Bound tree, representing the relaxation of the entire problem. Some cuts are possibly created on the problem but not all possible cuts are applied directly such that the model size stay reasonable. If a temporary integer solution is found in this first iteration, it is kept to be used in the algorithm as the incumbent solution. After the first step is finalized, we can look at the node computed in the Brand and Bound tree.

For each processed node, Cplex solves the relaxation of the subproblem. If cuts are violated by the solution, some or all cuts are added to the problem and if cuts have been added, we try to solve the subproblem again. This step is iterated until there are no cuts violated or the algorithm decides that enough cuts have been added to this node. Note that if after adding cuts to the subproblem, it becomes infeasible, the node is pruned by infeasibility. On the other hand, if the solution found for the subproblem respect the integrality constraints and its objective value is better than the one from the incumbent solution, then it becomes the incumbent solution. And if there exists a solution that does not respect the integrality contraints, then we branch the tree on that node. The branching occurs after a heuristic method is called to see if a new incumbent solution can be implied from the solution at this node.

As mentioned in Section 2.1, there are three possible strategy for selecting a node to process. Cplex uses the *Best Node* strategy. This strategy looks at the value of the optimal objective Z of each node after its relaxation is solved. The node with the best Z is chosen to be processed further. This best node value is also used to compute the MIP Gap when compared to the incumbent solution. The MIP Gap is a percentage used to measure the progress towards finding the optimal objective. When no nodes are left to process, the value of Z should have converged to the value of the incumbent, meaning the MIP Gap reaches zero, thus proving the optimality of the incumbent.

When solving a model with Cplex, it is possible to terminate the algorithm before the MIP Gap reaches zero (or more exactly $< 0.01\%$). The user can change the MIP Gap value or set a time limit or a limit on the number of node to be processed. If the solver stops because it reached the time limit, we can look at the value of the MIP Gap to look at the optimality ratio of the solution.

It is important to note that there is the option to start the model with a solution given by the user. But if you are solving multiple times the same problem with small modification at each iteration, it is not necessary to explicitly establish the solution from the last iteration as it will be automatically looked at for a possible starting solution. This notion will be looked at for one of the implementation of SSL.

2.3.1 Branch and Cut flow chart

Let's look at the flow chart of a maximization problem $z = \max\{cx : x \in X\}$ with formulation P.

1. **Initialization**: initialize the model with $-\infty$ as a lower bound, and incumbent x^* empty. Execute the preprocess of the initial problem and put it on the node list, then go to 2.

2. **Node**: If the node list is empty, then go to exit 8. Otherwise choose a node i and remove it from the node list, then go to 3.

3. **Restore**: the formulation P^i of the set X^i and set $k = 1$, and $P^{i,1} = P^i$. Then go to 4

4. **LP Relaxation**: At iteration k, solve $\bar{z}^{i,k} = \max\{cx : x \in P^{i,k}\}$. If it is infeasible, prune the node and go back to 2. Otherwise we get a solution $x^{i,k}$ and go to 5

5. **Cut**: at iteration k, use the cutting plane method to try and cut off $x^{i,k}$. If there are no cuts such cuts, go to 6. Else add the cuts to the formulation $P^{i,k+1} = P^{i,k} +$ cuts. Increment k and go back to 4.

6. **Prune**: If the upper bound $\bar{z}^{i,k} \leq z$, go to 2. If $x \in X$, set the lower bound $\underline{z} = \bar{z}^{i,k}$ and update the incumbent to $x^{i,k}$, then go back to 2. Otherwise go to 7.

7. **Branching**: Create two or more branch with the new subproblems X_t^i with formulation P_t^i and add them to the node list. Go back to 2.

8. **Exit** The incumbent solution $x*$ is optimal with objective value z.

Chapter 3

Results

In this chapter, a brief segment will show the results obtained in the different articles where the formulations have been found. The second part will explain how the formulations have been implemented and will display the results from the local experiments on those formulations.

3.1 Article Results

Simonetti, Salles da Cunha and Lucena implemented their own branch and cut algorithm using a best-node first strategy with calls to the XPRESS MIP solver (release 19.00). An explicit table of the results of the multiple experiments can be found in [21]. They have experimented on graphs with value $n \in \{30, 50, 70, 100, 120, 150, 200\}$ and a density varying between 5% to 70%.

It shows that the quality of the lower bounds of the formulation SSL $\mathscr{R}_1$ is better than SSL $\mathscr{R}_0$. It is also said that for small instances and large dense instances, their algorithm done on formulation $\mathscr{R}_1$ is better than the BC based on Directed Graph Reformulation (DGR) [12] but for large sparse instances, DGR remains stronger.

The three other have been solved using CPLEX 12.1 via IBM's Concert Technology library, version 2.9, also involving a branch and cut algorithm. They experimented on 6 types of graphs. An explicit table with the graphs' name and computation time can be found in [9].

It shows that overall, the MTZ constraints have the best performances for all types of graphs tested. But their test are limited to very particular graphs (IEEE-xx-Bus and RTS-96), for the experiments conducted in this thesis, there will be multiple types of graphs taken into account.

Comparing the first formulation SSL with the others would not be relevant since the environment in which they have been tested, the instances and the algorithms used are different. Which is one of the reason a comparison is done in this thesis.

3.2 Experimental results

3.2.1 Environment

All MIP formulations are implemented in python and solved using CPLEX 12.9 via IBM's Concert Technology library. All experiments are performed on a Linux workstation with 4 Intel(R) Core(TM) i7-4710MQ CPU @ 2.50GHz processors and 8GB RAM.

The optimality gap was set to 0.05.

3.2.2 Implementation

Here I will explain the evolution of the implementation for the different formulations and the problems encountered.

All codes can be found on `https://github.com/lbauwin/MCDSP`

Simonetti-Salles da Cunha-Lucena constraints

There has been three steps to the implementation process, first a simple as is without any optimization in the code. The implementation started by adding all the constraints and set the objective before solving it. Of course this meant that the number of constraints grew exponentially depending on the number of nodes in the graph (see Table 1.1). Therefore there was a limit of 15 nodes to not overload the test environment. A first optimization had to be done for the Constraints (1.3b).

The second step to optimize was to solve the problem by first omitting the Constraints (1.3b) such that we start with formulation P^0 and iteration $t = 0$. Cplex would then offer an integer solution for P^0 that would be tested to find any subtour existing in the solution. If such subtour exists, then the Constraints (1.3b) for the subset of V composed of the nodes part of the found subtours are added to the model giving the new formulation $P^1 \leftarrow P + GSEC$ and $t \leftarrow t + 1$, the given model is solved and so on until no subtour are found. Once the solution is subtour free, the solution is a valid integer solution for our problem. This implementation can be found in the file *SSL.py*.

Another option that I could see to optimize the implementation was to use the module Cplex Callbacks with lazy constraints to check the GSEC constraints. A callback is an object where the user can define the main method and Cplex can call this object at specific points during the optimization. It is a special object that can retrieve information about the current solution in the algorithm. Lazy constraints are constraints that are added to the model if they have been violated. So this time it starts by solving the model without the GSEC constraints, then during the algorithm, it calls on the callback object to look for violated GSEC constraints and add them to the model on the fly. The implementation can be found in the file *SSL_lazy.py*.

Martin constraints

All constraints and the objective function are implemented into the program as is and Big M is set to 10 for Constraints (1.8e) and (1.8f). The code can be found in the file *Martin.py*.

I attempted to optimize the formulation by modifying the Constraints (1.8e) and (1.8f) with indicator constraints on the variable $z_{i,j,k}$, but all attempts unfortunately failed to work or were not as efficient as the original code. The goal of an indicator constraints in Cplex is to create a link between a binary variable $(z_{i,j,k})$ and the constraints to determine if the constraint is active. It can be used to replace the Big M data that are artificial data and can be unstable. The indicator constraint formulation can be more robust, accurate and stable.

Single-Commodity Flow constraints

The single commodity flow constraints have been implemented using indicator constraints for the Constraints (1.10d) and (1.10f). I used it here so that there would be less computation, Constraints (1.10d) looks at the value in x_i and x_j such that if the value is null, the constraints simply equals zero. $f_{i,j} = 0$ if $x_i = 0$ or $x_j = 0$. Constraints (1.10f) analyze the value of r_i such that if the value is null, then the right-hand part of the inequality is simply x_i, giving the constraint $\sum_j f_{ji} - \sum_j f_{ij} = x_i, \forall i \in V$.

The code can be found in the file *SCF.py*

Miller-Tucker-Zemlin constraints

The Miller-Tucker-Zemlin constraints were straightforward and implemented as is without the need for callbacks or indicator constraints. The implementation is in the file *MTZ.py*.

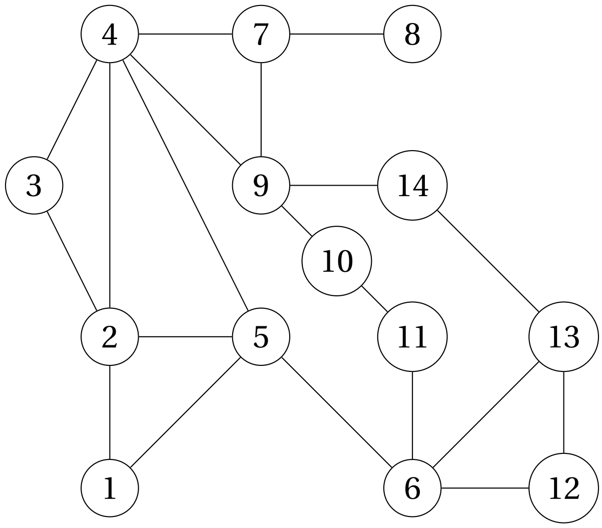


Figure 3.1 – Graphic representation of IEEE-14-Bus

3.2.3 Generated Instances

The experiment have been performed on 4 types of graphs:.

- IEEE-14-Bus with $|V| = 14$ and $|E| = 20$.
- IEEE-30-Bus with $|V| = 30$ and $|E| = 41$
- IEEE-57-Bus with $|V| = 57$ and $|E| = 78$

  Those three are some of the instances they used in [9]. A graphical representation of the first two can be seen in Figure 3.1 and Figure 3.2 respectively. Due to the large number of nodes to represent for the IEEE-57-Bus, it will not be portrayed here.

- Random graphs that are generated with two given values, the number of nodes $|V|$ and the number of edges $|E|$. The construction will start by adding $|V| - 1$ edges such that the graph is connected, the rest of the edges will be added randomly between two nodes.

The graphs are stored in 2 lists, V and E, where V is a list of integer representing the nodes and E is a list of tuples representing the edges.

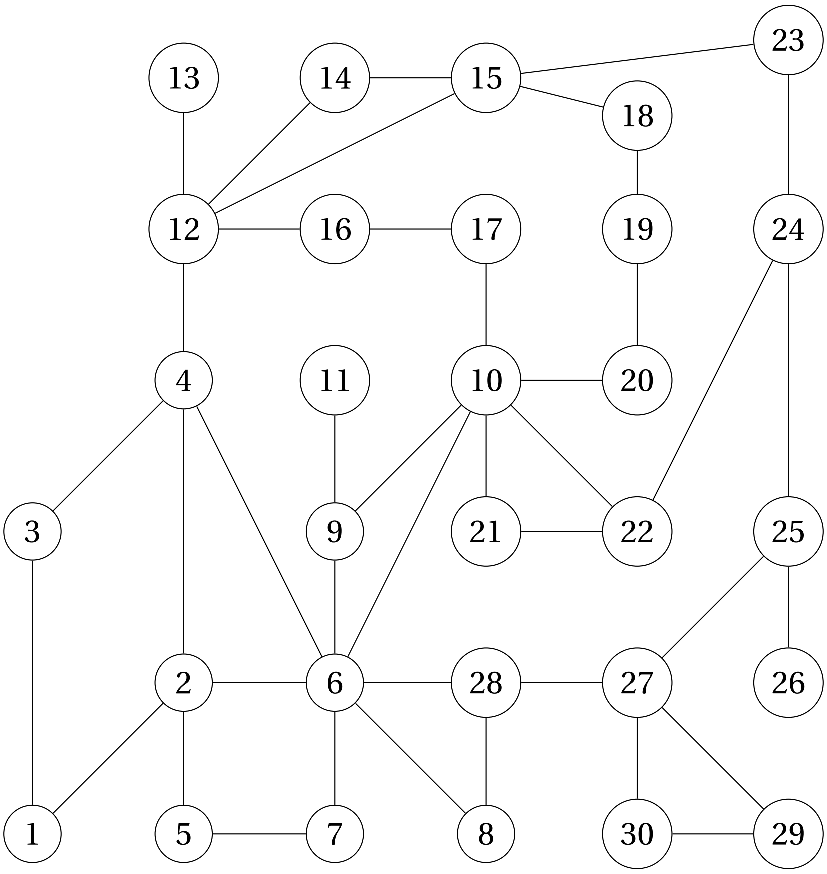


Figure 3.2 – Graphic representation of IEEE-30-Bus

3.2.4 Results

The results are based on several run for each type of graphs and displayed in this section. The graphs IEEE-xx-Bus have been run 5 times each, for the generated random graphs, the numbers of run varies from 5 to 10. It is important to note that for each run on random graphs with the same number of nodes and same density, the graph generated submitted to all 4 formulations but different between each run.

The dimension of the random graphs varies from $\{30, 40, 50, 60, 70, 80, 90, 100, 120, 150, 200\}$ nodes with a density range between 10% to 70%. The time limit for each formulation is set to 1 hour.

The results have been recorded into a file such that the number of nodes, the number of edges, the time to solve the model, the objective value, the number of constraints and the number of variables are known for each problem. Also, the system output have been redirected into a file to be able to go back and verify the details of the solve to get the information about the first lower bound and the number of nodes that have been used in the Branch and Cut algorithm.

Table 3.1 to 3.12 records the the type of graphs being tested, the IP formulation for which the results in the row are for, the number of variables and the number of constraints present in the model, the average, maximum and minimum time for each run on the type of graph considered, the maximum number of nodes generated in the Branch and Cut algorithm, the lower bound of the first relaxation for one of the given run with the objective value for that same run. The last two columns are there to represent the maximum gap in case the time limit has been encountered and how many run have successfully ran without interruption. It is important to note that the average time is computed with the successful run only, so even if the average time display is better, it is not conform to the reality.

During the experiments, some of the formulations have been abandoned when there were a significant under performance when compared to the other. The first example can already be seen for the IEEE-57-Bus for the Martin constraints. Then it is ran again for the random graphs with 30 nodes and then discarded for later run.

Graph	IP formulation	$\lvert$Var$\rvert$	$\lvert$Csts$\rvert$	Average time	Max time	Min time	$\lvert$Node list$\rvert$	First Relaxation	Objective	Max Gap	Resolved /Try
IEEE-14-Bus	SSL	34	103	0.007	0.008	0.007	0	5	5	0	5/5
	SSL_lazy	34	103	0.006	0.006	0.006	0	5	5	0	5/5
	MTZ	99	184	0.036	0.057	0.030	0	4.0909	5	0	5/5
	SCF	68	271	0.054	0.090	0.043	0	4	5	0	5/5
	Martin	2954	9395	0.118	0.125	0.115	0	1	5	0	5/5
IEEE-30-Bus	SSL	71	214	0.011	0.013	0.010	0	11	11	0	5/5
	SSL_lazy	71	214	0.007	0.007	0.007	0	11	11	0	5/5
	MTZ	205	380	0.056	0.058	0.054	0	10	11	0	5/5
	SCF	142	561	0.114	0.117	0.107	0	10	11	0	5/5
	Martin	27930	86191	1.265	1.288	1.243	0	3	11	0	5/5
IEEE-57-Bus	SSL	135	3432	1.125	1.816	0.888	0	28.67	31	0	5/5
	SSL_lazy	135	406	1.273	1.724	1.098	0	28.67	31	0	5/5
	MTZ	387	717	4.400	4.060	5.386	8028	16.25	31	0	5/5
	SCF	270	1066	1960.379	$> 1h$	1940.6	3450996	21	31	3.23%	4/5
	Martin	-	-	-	-	-	-	-	-	-	-

Table 3.1 – Summary of computational results for IEEE graphs

Graph	IP formulation	|Var|	|Csts|	Average time	Max time	Min time	|Node list|	First Relaxation	Objective	Max Gap	Resolved /Try
	SSL	76	510	6.589	45.119	0.012	0	9	12	0	7/7
$\|V\| = 30$	SSL_lazy	76	230	0.694	4.116	0.016	113	10.1	12	0	7/7
	MTZ	215	400	0.129	0.257	0.059	0	10.2	12	0	7/7
den = 10%	SCF	152	611	0.346	0.715	0.104	0	9	12	0	7/7
	Martin	27930	86491	1.826	3.081	1.033	0	7	12	0	7/7
	SSL	120	361	0.663	5.873	0.028	0	5.25	6	0	10/10
$\|V\| = 30$	SSL_lazy	120	361	0.131	0.384	0.010	0	5.25	6	0	10/10
	MTZ	303	576	0.107	0.194	0.069	0	5	6	0	10/10
den = 20%	SCF	240	1051	0.234	0.489	0.159	0	5	6	0	10/10
	Martin	27930	89131	5.944	12.998	2.147	0	1	6	0	10/10
	SSL	165	496	0.062	0.146	0.017	0	3.2863	4	0	10/10
$\|V\| = 30$	SSL_lazy	165	496	0.071	0.180	0.013	0	3.2863	4	0	10/10
	MTZ	393	756	0.112	0.138	0.081	0	2.8667	4	0	10/10
den = 30%	SCF	330	1501	0.235	0.432	0.128	0	2.8667	4	0	10/10
	Martin	27930	91831	4.559	7.788	2.782	0	2.8667	4	0	10/10
	SSL	210	631	0.066	0.130	0.015	0	2.8391	3	0	10/10
$\|V\| = 30$	SSL_lazy	210	631	0.059	0.108	0.010	0	2.8391	3	0	10/10
	MTZ	483	936	0.124	0.198	0.047	0	2.4854	3	0	10/10
den = 40%	SCF	420	1951	0.215	0.260	0.167	0	2.4854	3	0	10/10
	Martin	27930	94531	5.242	10.592	2.781	0	2.4854	3	0	10/10
	SSL	240	721	0.045	0.087	0.017	0	2.4499	3	0	10/10
$\|V\| = 30$	SSL_lazy	240	721	0.060	0.120	0.012	0	2.4499	3	0	10/10
	MTZ	543	1056	0.107	0.175	0.057	0	1.9939	3	0	10/10
den = 50%	SCF	480	2251	0.186	0.246	0.151	0	1.9939	3	0	10/10

	Martin	27930	96331	7.275	18.151	3.929	0	2	3	0	10/10
	SSL	285	856	0.024	0.038	0.020	0	2	2	0	10/10
$\lvert V\rvert = 30$	SSL_lazy	285	856	0.025	0.064	0.014	0	2	2	0	10/10
	MTZ	633	1236	0.113	0.157	0.87	0	1.7072	2	0	10/10
den = 60%	SCF	570	2701	0.228	0.355	0.148	0	1.7072	2	0	10/10
	Martin	27930	99031	12.169	21.117	5.312	0	2	2	0	10/10
	SSL	330	991	0.032	0.051	0.026	0	1.5332	2	0	10/10
$\lvert V\rvert = 30$	SSL_lazy	330	991	0.048	0.070	0.028	0	1.5332	2	0	10/10
	MTZ	723	1416	0.080	0.137	0.059	0	1.3333	2	0	10/10
den = 70%	SCF	660	3151	0.208	0.168	0.279	0	1.3333	2	0	10/10
	Martin	27930	101731	7.117	9.735	6.245	0	2	2	0	10/10

Table 3.2 – Summary of computational results for random graphs with 30 nodes

Graph	IP formulation	|Var|	|Csts|	Average time	Max time	Min time	|Node list|	First Relaxation	Objective	Max Gap	Resolved /Try
	SSL	121	457	1.034	2.013	0.014	0	10.8333	13	0	10/10
$\lvert V\rvert = 40$	SSL_lazy	121	361	1.695	11.694	0.017	90	10.8333	13	0	10/10
den = 10%	MTZ	323	606	0.315	0.612	0.113	2	10	13	0	10/10
	SCF	240	1001	1.502	7.292	0.299	2	10	13	0	10/10
	SSL	200	601	0.226	0.510	0.023	0	5.3193	7	0	10/10
$\lvert V\rvert = 40$	SSL_lazy	200	601	0.922	2.309	0.013	4	5.3193	7	0	10/10
den = 20%	MTZ	483	926	0.188	0.238	0.133	0	4.7565	7	0	10/10
	SCF	400	1801	0.689	1.120	0.213	2	4.7565	7	0	10/10
	SSL	280	841	0.191	0.318	0.051	0	3.3205	4	0	10/10
$\lvert V\rvert = 40$	SSL_lazy	280	841	0.329	0.532	0.037	0	3.3204	4	0	10/10
den = 30%	MTZ	643	1246	0.232	0.359	0.155	0	3.1792	4	0	10/10

	SCF	560	2601	0.521	0.893	0.309	0	3.1792	4	0	10/10
$\lvert V\rvert = 40$ den = 40%	SSL	360	1081	0.119	0.234	0.032	0	2.7831	4	0	10/10
	SSL_lazy	360	1081	0.152	0.319	0.026	0	2.7831	4	0	10/10
	MTZ	803	1566	0.187	0.301	0.156	0	2.4511	4	0	10/10
	SCF	720	3401	0.519	0.686	0.371	0	2.4511	4	0	10/10
$\lvert V\rvert = 40$ den = 50%	SSL	440	1321	0.071	0.161	0.030	0	2.2888	3	0	10/10
	SSL_lazy	440	1321	0.098	0.156	0.022	0	2.2888	3	0	10/10
	MTZ	963	1886	0.151	0.187	0.116	0	1.8779	3	0	10/10
	SCF	880	4201	0.434	0.714	0.341	0	1.8779	3	0	10/10
$\lvert V\rvert = 40$ den = 60%	SSL	500	1501	0.065	0.174	0.036	0	1.9827	3	0	10/10
	SSL_lazy	500	1501	0.073	0.141	0.025	0	1.9827	3	0	10/10
	MTZ	1083	2126	0.171	0.280	0.133	0	1.6224	3	0	10/10
	SCF	1000	4801	0.569	0.933	0.439	0	1.6224	3	0	10/10
$\lvert V\rvert = 40$ den = 70%	SSL	580	1741	0.077	0.134	0.060	0	1.7459	2	0	10/10
	SSL_lazy	580	1741	0.126	0.165	0.091	0	1.7459	2	0	10/10
	MTZ	1243	2446	0.171	0.232	0.108	0	1.4164	2	0	10/10
	SCF	1160	5601	0.480	0.576	0.399	0	1.4164	2	0	10/10

Table 3.3 – Summary of computational results for random graphs with 40 nodes

Graph	IP formulation	|Var|	|Csts|	Average time	Max time	Min time	|Node list|	First Relaxation	Objective	Max Gap	Resolved /Try
	SSL	175	708	5.4340	45.169	0.022	0	10.5	12	0	10/10
$\|V\| = 50$	SSL_lazy	175	526	33.051	259.160	0.072	189	10.5	12	0	10/10
den = 10%	MTZ	453	856	0.507	1.536	0.191	0	9.8101	12	0	10/10
	SCF	350	1501	1.612	4.107	0.350	994	9.75	12	0	10/10
	SSL	300	906	0.350	0.771	0.150	0	5.1091	6	0	10/10
$\|V\| = 50$	SSL_lazy	300	901	2.669	16.563	0.550	0	5.1091	6	0	10/10
den = 20%	MTZ	703	1356	0.335	0.597	0.173	0	4.6867	6	0	10/10
	SCF	600	2751	0.862	1.409	0.602	0	4.6867	6	0	10/10
	SSL	425	1276	0.375	0.680	0.175	0	3.5482	5	0	10/10
$\|V\| = 50$	SSL_lazy	425	1276	0.989	2.318	0.366	14	3.5482	5	0	10/10
den = 30%	MTZ	953	1856	0.389	0.482	0.328	0	3.3498	5	0	10/10
	SCF	850	4001	0.922	1.432	0.636	0	3.3498	5	0	10/10
	SSL	550	1651	0.213	0.301	0.162	0	2.8033	4	0	10/10
$\|V\| = 50$	SSL_lazy	550	1651	0.636	1.403	0.127	0	2.8033	4	0	10/10
den = 40%	MTZ	1203	2356	0.314	0.458	0.178	0	2.4072	4	0	10/10
	SCF	1100	5251	1.062	1.418	0.843	0	2.4072	4	0	10/10
	SSL	650	1951	0.260	0.731	0.069	0	2.3615	3	0	10/10
$\|V\| = 50$	SSL_lazy	650	1951	0.585	1.125	0.204	0	2.3615	3	0	10/10
den = 50%	MTZ	1403	2756	0.301	0.469	0.176	0	2.0551	3	0	10/10
	SCF	1300	6251	1.144	1.976	0.718	0	2.0551	3	0	10/10
	SSL	775	2326	0.100	0.159	0.067	0	2.1319	3	0	10/10
$\|V\| = 50$	SSL_lazy	775	2326	0.245	0.384	0.046	0	2.1319	3	0	10/10
den = 60%	MTZ	1653	3256	0.382	0.494	0.314	0	1.6629	3	0	10/10
	SCF	1550	7501	1.337	2.050	0.856	2	1.6629	3	0	10/10

	SSL	900	2701	0.132	0.206	0.102	0	1.8476	2	0	10/10
$\|V\| = 50$	SSL_lazy	900	2701	0.209	0.281	0.103	0	1.8476	2	0	10/10
den = 70%	MTZ	1903	3756	0.287	0.374	0.230	0	1.4417	2	0	10/10
	SCF	1800	8751	0.884	1.254	0.717	0	1.4417	2	0	10/10

Table 3.4 – Summary of computational results for random graphs with 50 nodes

Graph	IP formulation	|Var|	|Csts|	Average time	Max time	Min time	|Node list|	First Relaxation	Objective	Max Gap	Resolved /Try
$\|V\| = 60$	SSL	240	839	2.416	11.327	0.180	0	10.6466	12	0	10/10
den = 10%	SSL_lazy	240	721	57.963	236.611	0.281	0	10.6466	12	0	10/10
	MTZ	603	1146	0.828	1.626	0.210	41	10.2045	12	0	10/10
	SCF	480	2101	4.237	18.962	1.306	834	10.2045	12	0	10/10
$\|V\| = 60$	SSL	420	1269	1.721	3.749	0.216	0	5.0733	5	0	10/10
den = 20%	SSL_lazy	420	1261	10.890	46.237	0.972	24	5.0733	6	0	10/10
	MTZ	963	1866	0.835	1.777	0.459	0	4.8238	6	0	10/10
	SCF	840	3901	1.967	2.846	1.245	0	4.8238	6	0	10/10
$\|V\| = 60$	SSL	600	1801	0.581	0.841	0.317	0	3.4702	5	0	10/10
den = 30%	SSL_lazy	600	1801	6.780	44.104	1.073	13	3.4702	5	0	10/10
	MTZ	1323	2586	0.497	0.620	0.416	0	3.1383	5	0	10/10
	SCF	1200	5701	1.967	3.860	1.207	293	3.1383	5	0	10/10
$\|V\| = 60$	SSL	780	2341	0.518	0.859	0.336	0	2.8062	4	0	10/10
den = 40%	SSL_lazy	780	2341	1.503	3.281	0.648	20	2.8062	4	0	10/10
	MTZ	1683	3306	0.584	0.775	0.436	0	2.3638	4	0	10/10
	SCF	1560	7501	2.185	3.338	1.379	0	2.3638	4	0	10/10
$\|V\| = 60$	SSL	960	2881	0.428	1.254	0.132	0	2.2683	3	0	10/10

den = 50%	SSL_lazy	960	2881	0.688	2.190	0.372	0	2.2683	3	0	10/10
	MTZ	2043	4026	0.482	0.793	0.245	0	1.9184	3	0	10/10
	SCF	1920	9301	1.880	3.376	1.179	0	1.9184	3	0	10/10
$\|V\| = 60$ den = 60%	SSL	1110	3331	0.170	0.295	0.099	0	2	2	0	10/10
	SSL_lazy	1110	3331	0.416	0.573	0.64	0	2	2	0	10/10
	MTZ	2343	4626	0.523	0.710	0.306	0	1.6504	2	0	10/10
	SCF	2220	10801	2.405	3.938	1.343	0	1.6504	2	0	10/10
$\|V\| = 60$ den = 70%	SSL	1290	3871	0.278	0.584	0.164	0	1.8308	2	0	10/10
	SSL_lazy	1290	3871	0.364	0.481	0.193	0	1.8308	2	0	10/10
	MTZ	2703	5346	0.437	0.626	0.237	0	1.3858	2	0	10/10
	SCF	2580	12601	1.633	2.493	1.289	0	1.3858	2	0	10/10

Table 3.5 – Summary of computational results for random graphs with 60 nodes

Graph	IP formulation	|Var|	|Csts|	Average time	Max time	Min time	|Node list|	First Relaxation	Objective	Max Gap	Resolved /Try
$\|V\| = 70$ den = 10%	SSL	315	1096	4.186	11.400	0.346	0	10.0180	12	0	10/10
	SSL_lazy	315	946	557.240	1355.090	62.189	107	10.0180	12	0	10/10
	MTZ	773	1476	1.329	2.354	0.703	2	9.5408	12	0	10/10
	SCF	630	2801	26.935	124.408	3.836	4806	9.5408	12	0	10/10
$\|V\| = 70$ den = 20%	SSL	560	1681	1.944	4.681	1.031	0	5.0665	7	0	10/10
	SSL_lazy	560	1681	107.934	752.931	1.758	50	5.0665	7	0	10/10
	MTZ	1263	2456	1.049	2.197	0.564	0	4.9190	7	0	10/10
	SCF	1120	5251	3.292	6.999	1.828	0	4.9190	7	0	10/10
$\|V\| = 70$ den = 30%	SSL	805	2416	1.061	2.302	0.362	0	3.5872	5	0	10/10
	SSL_lazy	805	2416	4.768	7.675	1.784	50	3.5872	5	0	10/10
	MTZ	1753	3436	1.259	3.862	0.716	0	3.2583	5	0	10/10

	SCF	1610	7701	5.041	10.734	3.027	271	3.2583	5	0	10/10
$\lvert V\rvert = 70$	SSL	1050	3151	1.157	3.151	0.405	0	2.8801	4	0	10/10
den = 40%	SSL_lazy	1050	3151	3.480	5.532	1.691	10	2.8801	4	0	10/10
	MTZ	2243	4416	0.910	1.413	0.732	2	2.4383	4	0	10/10
	SCF	2100	10151	4.814	10.795	2.064	518	2.4383	4	0	10/10
$\lvert V\rvert = 70$	SSL	1260	3781	1.054	3.898	0.206	0	2.3484	3	0	10/10
den = 50%	SSL_lazy	1260	3781	3.164	7.330	0.665	103	2.3484	3	0	10/10
	MTZ	2663	5256	0.755	1.423	0.435	0	2.0220	3	0	10/10
	SCF	2520	12251	3.713	9.995	1.626	0	2.0220	3	0	10/10
$\lvert V\rvert = 70$	SSL	1505	4516	0.350	0.984	0.170	0	2.1447	3	0	10/10
den = 60%	SSL_lazy	1505	4516	0.809	1.103	0.113	0	2.1447	3	0	10/10
	MTZ	3153	6236	0.928	1.279	0.478	0	1.6748	3	0	10/10
	SCF	3010	14701	3.993	4.913	2.468	29	1.6748	2	0	10/10
$\lvert V\rvert = 70$	SSL	1750	5251	0.527	0.834	0.273	0	1.7947	2	0	10/10
den = 70%	SSL_lazy	1750	5251	0.616	0.704	0.495	0	1.7947	2	0	10/10
	MTZ	3643	7216	0.769	1.007	0.432	0	1.4330	2	0	10/10
	SCF	3500	17151	2.841	4.410	2.226	0	1.4330	2	0	10/10

Table 3.6 – Summary of computational results for random graphs with 70 nodes

Graph	IP formulation	|Var|	|Csts|	Average time	Max time	Min time	|Node list|	First Relaxation	Objective	Max Gap	Resolved /Try
$\|V\| = 80$ den = 10%	SSL	400	1405	17.664	37.087	4.076	2	9.9853	12	0	5/5
	SSL_lazy	400	1201	672.302	1321.504	52.569	988	9.9853	12	0	5/5
	MTZ	963	1846	3.303	5.050	1.852	1031	9.3171	12	0	5/5
$\|V\| = 80$ den = 20%	SSL	720	2168	4.973	9.457	2.918	62	5.1431	7	0	5/5
	SSL_lazy	720	2161	762.244	1536.386	17.860	876	5.1431	7	0	5/5
	MTZ	1603	3126	2.863	4.184	1.378	0	4.8017	7	0	5/5
$\|V\| = 80$ den = 30%	SSL	1040	3121	3.253	7.568	1.236	5	3.5790	5	0	5/5
	SSL_lazy	1040	3121	9.841	19.580	2.441	60	3.5790	5	0	5/5
	MTZ	2243	4406	1.986	2.503	1.605	180	3.2012	5	0	5/5
$\|V\| = 80$ den = 40%	SSL	1360	4081	2.025	4.572	0.735	31	2.9549	4	0	5/5
	SSL_lazy	1360	4081	6.904	14.535	2.302	153	2.9549	4	0	5/5
	MTZ	2883	5686	1.492	1.968	1.347	0	2.5839	4	0	5/5
$\|V\| = 80$ den = 50%	SSL	1680	5041	2.221	5.217	0.418	0	2.3487	3	0	5/5
	SSL_lazy	1680	5041	5.107	11.518	1.179	18	2.3487	3	0	5/5
	MTZ	3523	6966	1.509	3.476	0.855	0	1.9182	3	0	5/5
$\|V\| = 80$ den = 60%	SSL	1960	5881	0.541	0.891	0.363	0	2.0824	3	0	5/5
	SSL_lazy	1960	5881	1.508	1.968	1.151	0	2.0824	3	0	5/5
	MTZ	4083	8086	1.666	2.238	1.302	0	1.6666	3	0	5/5
$\|V\| = 80$ den = 70%	SSL	2280	6841	1.198	1.401	0.959	0	1.8037	2	0	5/5
	SSL_lazy	2280	6841	0.944	0.573	1.118	0	1.8037	2	0	5/5
	MTZ	4723	9366	1.126	1.499	0.823	0	1.3883	2	0	5/5

Table 3.7 – Summary of computational results for random graphs with 80 nodes

Graph	IP formulation	|Var|	|Csts|	Average time	Max time	Min time	|Node list|	First Relaxation	Objective	Max Gap	Resolved /Try
$\|V\| = 90$ den = 10%	SSL	495	1744	62.643	153.345	8.327	1689	9.9242	13	0	5/5
	SSL_lazy	-	-	-	-	-	-	-	-	-	-
	MTZ	1173	2256	11.214	22.615	3.529	1229	9.4430	13	0	5/5
$\|V\| = 90$ den = 20%	SSL	900	2701	10.170	32.853	2.725	323	5.1120	7	0	5/5
	SSL_lazy	900	2701	299.219	1079.678	21.166	1338	5.1120	7	0	5/5
	MTZ	1983	3876	4.614	6.039	1.592	73	4.9220	7	0	5/5
$\|V\| = 90$ den = 30%	SSL	1305	3916	10.704	33.120	2.812	11	3.6066	5	0	5/5
	SSL_lazy	1305	3916	72.772	$> 1h$	19.118	1753	3.6066	6	30.25%	4/5
	MTZ	2793	5496	3.324	7.772	1.677	27	3.3326	5	0	5/5
$\|V\| = 90$ den = 40%	SSL	1710	5131	5.350	9.372	1.591	18	2.8489	4	0	5/5
	SSL_lazy	1710	5131	19.002	38.851	4.094	497	2.8489	4	0	5/5
	MTZ	3603	7116	3.854	8.326	1.108	42	2.4245	4	0	5/5
$\|V\| = 90$ den = 50%	SSL	2070	6211	6.316	9.743	1.062	76	2.4654	4	0	5/5
	SSL_lazy	2070	6211	12.489	20.175	1.863	217	2.4654	4	0	5/5
	MTZ	4323	8556	2.621	5.176	0.695	456	2.0001	4	0	5/5
$\|V\| = 90$ den = 60%	SSL	2475	7426	1.264	2.976	0.588	0	1.9819	3	0	5/5
	SSL_lazy	2475	7426	1.946	2.174	1.686	0	1.9819	3	0	5/5
	MTZ	5133	10176	2.421	2.912	1.780	0	1.6136	3	0	5/5
$\|V\| = 90$ den = 70%	SSL	2880	8641	1.919	2.763	0.880	0	1.8606	2	0	5/5
	SSL_lazy	2880	8641	2.086	2.667	1.626	0	1.8606	2	0	5/5
	MTZ	5943	11796	1.514	1.777	1.310	0	1.4266	2	0	5/5

Table 3.8 – Summary of computational results for random graphs with 90 nodes

Graph	IP formulation	|Var|	|Csts|	Average time	Max time	Min time	|Node list|	First Relaxation	Objective	Max Gap	Resolved /Try
$\|V\| = 100$	SSL	600	1881	34.540	6.343	89.221	631	10.6992	13	0	5/5
den = 10%	MTZ	1403	2706	13.877	28.167	5.258	3478	10.3611	13	0	5/5
$\|V\| = 100$	SSL	1100	3323	51.074	102.880	5.075	3603	5.2104	8	0	5/5
den = 20%	MTZ	2403	4706	11.685	12.570	10.115	2667	4.9846	8	0	5/5
$\|V\| = 100$	SSL	1600	4801	10.750	18.858	5.261	22	3.6687	5	0	5/5
den = 30%	MTZ	3403	6706	5.996	9.995	1.825	1013	3.3084	5	0	5/5
$\|V\| = 100$	SSL	2100	6301	11.723	28.216	3.238	0	2.8615	4	0	5/5
den = 40%	MTZ	4403	8706	3.550	5.044	2.555	23	2.4443	4	0	5/5
$\|V\| = 100$	SSL	2600	7801	11.880	19.001	0.686	53	2.4125	3	0	5/5
den = 50%	MTZ	5403	10706	3.485	8.289	1.556	0	1.9645	3	0	5/5
$\|V\| = 100$	SSL	3050	9151	1.170	1.451	0.849	0	2.1036	3	0	5/5
den = 60%	MTZ	6303	12506	3.585	5.350	2.789	0	1.7007	3	0	5/5
$\|V\| = 100$	SSL	3550	10651	2.909	3.410	2.513	0	1.8512	2	0	5/5
den = 70%	MTZ	7303	14506	3.066	3.977	2.244	0	1.4228	2	0	5/5

Table 3.9 – Summary of computational results for random graphs with 100 nodes

Graph	IP formulation	|Var|	|Csts|	Average time	Max time	Min time	|Node list|	First Relaxation	Objective	Max Gap	Resolved /Try
$\|V\| = 120$	SSL	840	2735	607.854	$> 1h$	56.717	257592	10.2163	15	8.01%	4/5
den = 10%	MTZ	1923	3726	92.241	288.577	17.018	9449	10.0174	14	0	5/5
$\|V\| = 120$	SSL	1560	4681	117.869	279.642	29.699	6301	5.2037	8	0	5/5
den = 20%	MTZ	3363	6606	62.062	94.231	30.747	4496	4.7844	8	0	5/5
$\|V\| = 120$	SSL	2280	6841	61.736	117.746	44.786	1259	3.7388	6	0	5/5
den = 30%	MTZ	4803	9486	27.845	33.930	20.039	4684	3.3105	6	0	5/5
$\|V\| = 120$	SSL	3000	9001	79.270	98.539	47.271	1775	2.9639	5	0	5/5
den = 40%	MTZ	6243	12366	26.257	33.591	4.926	2875	2.5355	5	0	5/5
$\|V\| = 120$	SSL	3720	11161	47.792	49.231	46.588	145	2.4038	4	0	5/5
den = 50%	MTZ	7683	15246	20.981	26.111	17.558	845	1.9808	4	0	5/5
$\|V\| = 120$	SSL	4380	13141	12.633	15.405	4.240	5	2.1147	3	0	5/5
den = 60%	MTZ	9003	17886	8.068	11.140	6.410	0	1.6729	3	0	5/5
$\|V\| = 120$	SSL	5100	15301	7.863	9.788	5.241	0	1.9079	3	0	5/5
den = 70%	MTZ	10443	20766	12.817	16.932	2.891	98	1.4203	3	0	5/5

Table 3.10 – Summary of computational results for random graphs with 120 nodes

Graph	IP formulation	|Var|	|Csts|	Average time	Max time	Min time	|Node list|	First Relaxation	Objective	Max Gap	Resolved /Try
$\|V\| = 150$	SSL	1275	3852	466.234	$> 1h$	42.010	689283	10.2049	14	10.21%	3/5
den = 10%	MTZ	2853	5556	932.217	1751.682	138.336	412708	9.9486	15	0	5/5
$\|V\| = 150$	SSL	2400	7211	1207.397	$> 1h$	123.006	170958	5.2743	9	15.01%	3/5
den = 20%	MTZ	5103	10056	1047.145	$> 1h$	56.987	269930	4.9731	9	18.37%	3/5
$\|V\| = 150$	SSL	3525	10580	536.604	925.468	164.617	2593	3.7911	6	0	5/5

den = 30%	MTZ	7353	14556	111.412	250.344	62.402	8634	3.3768	6	0	5/5
$\|V\| = 150$	SSL	4650	13951	261.217	314.043	99.459	2971	2.9264	5	0	5/5
den = 40%	MTZ	9603	19056	115.608	262.720	12.357	4326	2.4576	5	0	5/5
$\|V\| = 150$	SSL	5700	17101	123.145	134.962	110.676	175	2.4336	4	0	5/5
den = 50%	MTZ	11703	23256	47.005	48.333	45.264	774	2.0046	4	0	5/5
$\|V\| = 150$	SSL	6825	20476	40.246	57.585	14.532	7	2.1716	3	0	5/5
den = 60%	MTZ	13953	27756	18.217	22.274	14.912	91	1.6617	3	0	5/5
$\|V\| = 150$	SSL	7950	23851	29.636	37.225	26.129	0	1.9021	3	0	5/5
den = 70%	MTZ	16203	32256	27.336	32.328	23.587	146	1.4160	3	0	5/5

Table 3.11 – Summary of computational results for random graphs with 150 nodes

Graph	IP formulation	|Var|	|Csts|	Average time	Max time	Min time	|Node list|	First Relaxation	Objective	Max Gap	Resolved /Try
$\|V\| = 200$	SSL	8200	24601	1511.247	$> 1h$	1453.185	10056	2.9626	5	32.24%	3/5
den = 40%	MTZ	16803	33406	1178.250	1522.920	1023.268	22693	2.4533	5	0	5/5
$\|V\| = 200$	SSL	10200	30601	618.304	691.194	558.405	291	2.4598	4	0	5/5
den = 50%	MTZ	20803	41406	180.532	214.567	160.248	1540	1.9694	4	0	5/5
$\|V\| = 200$	SSL	12100	36301	579.632	795.192	171.590	76	2.1994	3	0	5/5
den = 60%	MTZ	24603	49006	62.866	70.696	53.421	154	1.6631	3	0	5/5
$\|V\| = 200$	SSL	14100	42301	103.075	117.834	84.836	0	1.8867	3	0	5/5
den = 70%	MTZ	28603	57006	105.880	116.085	98.678	249	1.4187	3	0	5/5

Table 3.12 – Summary of computational results for random graphs with 200 nodes

Chapter 4

Discussion

Martin connectivity constraints have a much bigger size than any other formulation, one reason for the model to be slower than all others. It can be already seen for the IEEE-xx-Bus and the random graphs with 30 nodes. This is why no other experiences were made with the Martin constraints. Also, the space required to perform the Martin model was too big too fast for the available space on the computer which lead to a system crash.

The Single Commodity Flow constraints were also very slow when compared with MTZ and SSL for most graphs. As for the Martin model, the experiences on SCF were stopped after we reached a graph of more than 50 nodes as it was slower and it showed no amelioration.

SSL and MTZ are the only two with competitive results. The SSL formulation is most efficient for large densities in graphs and is clearly behind when it comes to smaller densities. It can clearly be explained by how it was implemented, for small densities, there are more chances that the first solution found contains a subtour and thus that we need to add more constraints and a few iteration. It also explains the large difference between the maximum time and minimum time for SSL as it depends on how many times the model had to add the GSEC constraints. MTZ outperforms SSL for small density graphs and has competitive efficiency for large density graphs.

It is important to note that SCF and Martin connectivity constraints had the right objective value soon in the process but the analysis of all the nodes open made it slower by at least a factor of 10.

Conclusion

List of Figures

List of Tables

Bibliography

[1] Sergiy Butenko, Xiuzhen Cheng, Carlos A. Oliveira, and P. M. Pardalos. *A New Heuristic for the Minimum Connected Dominating Set Problem on Ad Hoc Wireless Networks*, pages 61–73. Springer US, Boston, MA, 2004.

[2] Chandra Chekuri. Combinatorial optimization: Spanning trees. Accessed: 2019, 2009.

[3] Jens Clausen. Branch and bound algorithms – principles and examples, 1999.

[4] William Cook. Subtour elimination. http://www.math.uwaterloo.ca/tsp/methods/opt/subtour.htm. Accessed: 2020-06-30.

[5] IBM® ILOG® CPLEX®. *Concert Technology version 12.1 C++ API Reference Manual*. IBM.

[6] IBM® ILOG® CPLEX®. Ibm knowledge center.

[7] Bistra Dilkina and Carla Gomes. Solving connected subgraph problems in wildlife conservation. volume 6140, pages 102–116, 12 2010.

[8] Th. M. Liebling F. Margot, A. Prodon. Tree polytope on 2-trees. In *Mathematical Programming*, pages 183–191, January 1994.

[9] Neng Fan and Jean-Paul Watson. Solving the connected dominating set problem and power dominating set problem by integer programming. pages 371–382, August 2012.

[10] Samuel Fiorini, Tony Huynh, Gwenaël Joret, and Kanstantsin Pashkovich. Discrete and computational geometry. pages 757–761, 04 2016.

[11] Kamyar Khodamoradi and Ramesh Krishnamurti. Prize collecting travelling salesman problem - fast heuristic separations. pages 380–387, 01 2016.

[12] Abilio Lucena, Nelson Maculan, and Luidi Simonetti. Reformulations and solution algorithms for the maximum leaf spanning tree problem. *Computational Management Science*, 7(3):289–311, July 2010.

[13] Jim Luedtke. The branch-and-cut algorithm for solving mixed-integer optimization problems. Accessed: 2019-11-21, 2016.

[14] Thomas L. Magnanti and Laurence A. Wolsey. Chapter 9 optimal trees. In *Network Models*, volume 7 of *Handbooks in Operations Research and Management Science*, pages 503 – 615. Elsevier, 1995.

[15] Hugues Marchand, Alexander Martin, Robert Weismantel, and Laurence Wolsey. Cutting planes in integer and mixed integer programming. *Discrete Applied Mathematics*, 123(1):397 – 446, 2002.

[16] R.Kipp Martin. Using separation algorithms to generate mixed integer model reformulations. *Oper. Res. Lett.*, 10(3):119–128, April 1991.

[17] Marin Vlastelica Pogančić. The branch and bound algorithm. http://www.math.uwaterloo.ca/tsp/methods/opt/subtour.htm, 2019. Accessed: 2020-06-30.

[18] Gn Purohit and Sharma Usha. Constructing minimum connected dominating set: Algorithmic approach. *International Journal on Applications of Graph Theory in Wireless ad hoc Networks and Sensor Networks*, 2, September 2010.

[19] Frederico P. Quintão, Alexandre Salles da Cunha, Geraldo R. Mateus, and Abilio Lucena. The k-cardinality tree problem: Reformulations and lagrangian relaxation. *Discrete Applied Mathematics*, 158(12):1305 – 1314,

2010. Traces from LAGOS'07 IV Latin American Algorithms, Graphs, and Optimization Symposium Puerto Varas - 2007.

[20] Paul A. Rubin. User cuts versus lazy constraints. https://orinanobworld.blogspot.com/2012/08/user-cuts-versus-lazy-constraints.html. Accessed: 2020-04-24.

[21] Luidi Simonetti, Alexandre Salles da Cunha, and Abilio Lucena. The minimum connected dominating set problem: Formulation, valid inequalities and a branch-and-cut algorithm. In Julia Pahl, Torsten Reiners, and Stefan Voß, editors, *Network Optimization*, pages 162–169, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[22] Wikipedia contributors. Covering problems — Wikipedia, the free encyclopedia, 2019. [Online; accessed 14-August-2019].

[23] L.A. Wolsey. *Integer Programming*. Wiley Series in Discrete Mathematics and Optimization. Wiley, 1998.