

# **Solving Connected Dominating Set Variants Using Integer Linear Programming**

**Mario Surlemont**

Bachelorarbeit

Beginn der Arbeit:	12. Mai 2020
Abgabe der Arbeit:	17. August 2020
Gutachter:	Prof. Dr. Gunnar Klau Prof. Dr. Michael Leuschel



## **Erklärung**

Hiermit versichere ich, dass ich diese Bachelorarbeit selbstständig verfasst habe. Ich habe dazu keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Düsseldorf, den 17. August 2020

---

Mario Surlemont



# 1 Abstract

Maximizing photosynthetic outcomes is one of many different objectives of a plant. In this thesis we present and evaluate a method to predict an optimal venation pattern for leafs based on the minimal number of leaf cells that have to be transformed into vein cells to supply the entire leaf with nutrients and water. The model only focuses on the number of cells and disregards other aspects of the vascular system, like the vein hierarchy. To implement this model we used a special variant of the Minimum Dominating Set Problem which we implemented using Integer Linear Programming. We call this variant to model the vascular system the Minimum Connected rooted  $k$ -hop Dominating Set Problem. Our results show that our implementation is not capable of solving larger instances in a reasonable amount of time. In comparison to an implementation in Answer Set Programming our implementation performs worse using the instances that represent plant leafs. We present a detailed comparison between both versions and tested instances of different structure and size. We analyzed why the Integer Linear Programming implementation performs bad on the leaf graphs. The tests also revealed that on randomly generated graphs the Integer Linear Programming implementation outperformed the Answer Set Programming implementation.





## Contents

<b>1 Abstract</b>	<b>1</b>
<b>2 Introduction</b>	<b>1</b>
<b>3 Preliminaries</b>	<b>3</b>
3.1 Linear Programming . . . . .	3
3.2 Definitions . . . . .	3
<b>4 Methods</b>	<b>5</b>
4.1 Minimum Dominating Set . . . . .	5
4.2 Minimum $k$ -hop Dominating Set . . . . .	5
4.3 Connectivity . . . . .	6
4.4 Minimum connected $k$ -hop Dominating Set . . . . .	11
4.5 Minimum rooted connected $k$ -hop Dominating Set . . . . .	12
4.6 Additional methods to tighten up the space of feasible solutions . . . . .	12
<b>5 Implementation</b>	<b>14</b>
<b>6 Results</b>	<b>15</b>
<b>7 Discussion</b>	<b>27</b>
<b>8 Conclusion</b>	<b>30</b>
<b>A Appendix</b>	<b>32</b>
<b>References</b>	<b>38</b>
<b>List of Figures</b>	<b>40</b>
<b>List of Tables</b>	<b>40</b>



## 2 Introduction

Plants try to optimize their architecture to fulfil different objectives. One of it is to maximize the photosynthetic output. Another one is to minimize the cost to build the vascular system [4]. To maximize the photosynthetic output plants optimize different parameters. As increasing one parameter can reduce another one, many parameters can not be optimized at the same time [4] [14]. In this thesis we focus on one particular mechanism how plants can optimize their photosynthetic output.

To generate photosynthetical gains plants need sunlight, carbondioxid and water. (Photosynthese zit. ) Water and nutrients are supplied via the vascular system. Xylem transports water to the leaves where the mesophyl cells produce sugars. These sugars are carried out to the whole plant by phloem, a tissue specialized on transporting sugars. Xylem and phloem cells are not able to generate sugars, but they are mandatory to supply water to the mesophyl cells and to transport sugars. To be satisfied with the amount of water mesophyl cells have access to, they must not be more than 2-3 cells away from a xylem cell. In this range water can flow from the xylem cells through mesophyl cells that are not next to a xylem cell via diffusion. At the same time sugars can be transported away from the mesophyl cells and supplied to the phloem if there is a phloem cell in the range of 2-3 cells. (Zitat finden. )

To produce as much sugar as possible the plant can try to (driven by evolutionary processes) maximize the number of mesophyl cells by minimizing the number of vein cells. In this thesis we describe a method to reproduce an optimal venation pattern that minimizes the number of vein cells with respect to the constraint that all mesophyl cells need to be in a fixed range to vein cells. Leaf veins have a hierarchy. In general there is at least one thick major vein branch and several narrow minor branches. This hierarchy is completely disregarded in our problem formulation. Environmental circumstances also influence the venation pattern [16]. These influences on the venation are also completely disregarded in our model. The input instance is given by an undirected graph  $G = (V, E)$  that represents a leaf. The set of vertices  $V$  represents the leaf cells while the set of edges  $E$  represents the connections between the leaf cells in the form of plasmodesmata. To find an optimal pattern we use a special variant of the dominating set problem. For this problem we present an ILP-formulation and an implementation in a branch and cut framework.

The dominating set problem and several variants are NP-hard [9]. For our specific case we demand connectivity between the members of the set. This connectivity in ILP-formulations is subject of different publications as it is not trivial. Hyunh [10] presented in her bachelors thesis an alternative to ILPs. She implemented an algorithm for our problem using Answer Set Programming (ASP). For larger input instances the ASP-version did not create optimal solutions in a reasonable amount of time. Hyunh [10] compared for the case where the dominating set does not need to be connected the runtime from an ILP-Version to the runtime from her ASP-version. Her tests revealed that for this particular problem the ILP-version performed significantly better.

Goal of this thesis is to formulate an ILP and to evaluate whether if this performs better on our input graphs. We compared the ASP-version with an ILP-formulation that was created in this thesis. Contrary to the presumption that the ILP-version could generate

solutions faster, on our input instances the ASP-version was significantly faster. However the ILP-version outperformed the ASP-version on random graphs. The different characteristics and the runtime for the graphs can be taken from the results section. In the discussion section we discuss which characteristics are responsible for the differences in the runtime and what effect initiates them. In Section 2, the Preliminaries, we will give a short introduction in ILP. Additionally important definitions are stated. After that in the following Section 3 we define the methods to find an optimal venation pattern. Section 4 demonstrates the implementation. At last in Section 4 and Section 5 we present the results and followed by a discussion on the effectiveness and limitations of the ILP-solution and which characteristics graphs hold to perform either better with the ILP-version or with the ASP-version.

### 3 Preliminaries

#### 3.1 Linear Programming

Linear programming is a technique to minimize linear functions. The following definition is based on the book [7]

A linear program (LP) problem consists of an linear objective function that is minimized with respect to a set of linear inequalities.

Linear programs can be expressed as

$$\min\{c^T x : Ax \geq b, x \geq 0\}$$

where  $b \in \mathbb{R}^m$  and  $c \in \mathbb{R}^n$  are constant vectors. The matrix  $A \in \mathbb{R}^{m \times n}$  contains the coefficients of the  $m$  inequalities. We minimize the objective function  $c^T x \in \mathbb{R}$ . The vector inequality  $Ax \geq b$  has to be satisfied for a valid solution. The vector  $x \in \mathbb{R}^n$  describes possible solutions. If  $x \in \mathbb{R}^n$  satisfies all inequalities it is called a feasible solution. A solution  $x^*$  is optimal if it respects all inequalities and is minimal.

Integer linear programs (ILPs) are linear programs with the additional restriction that all variables have to be integers:  $x \in \mathbb{Z}^n$ . The decision variant of an ILP is NP-complete [9].

Each line  $j$  of  $Ax \geq b$  can be expressed as the sum  $\sum_{i=1}^n a_{ij}x_i \geq b_j$ . The objective function can be expressed as  $\sum_{i=1}^n c_i x_i$ . In this thesis we use this notation as we perceive it as more readable. Combinatorial optimization problems can be modeled with ILPs. Every variable  $x_i \in \{0, 1\}$  denotes a possible decision to include item  $i \in \{1, \dots, n\}$  in the solution.

#### 3.2 Definitions

**Definition 1** (Neighborhood). Given an undirected graph  $G = (V, E)$ . Let  $N(v)$  denote the neighborhood of a vertex  $v$ .  $N(v)$  can formally be described as follows:

$$w \in N(v) \Leftrightarrow \exists (v, w) \in E$$

**Definition 2** (Dominating Set). Given an undirected Graph  $G = (V, E)$  a dominating set is a subset  $D \subset V$  such that each vertex  $v \in V$  is either included in the dominating set or adjacent to at least one vertex which is included in the dominating set. For a dominating set  $D$  the following statement is valid

$$\forall v \in V \setminus D : \exists u \in D, u \in N(v)$$

**Definition 3** (k-Neighborhood). The neighborhood of a single vertex  $N(v)$  is defined above. Let the neighborhood of a set of vertices  $W \subset V$  be defined as

$$N(W) := \bigcup_{u \in W} N(u)$$

Let  $k \in \mathbb{N}$ . With help of this definition the  $k$ -neighborhood  $N_k(v)$  of a single vertex  $v \in V$  can recursively be defined as:

$$N_k(v) := N(N_{k-1}(v)) \setminus v$$

whereas  $N_1(v) = N(v)$ . So  $N_k(v)$  is a set of all vertices which can be reached with at most  $k$  steps starting from  $v$ .

**Definition 4** ( $k$ -hop Dominating Set). A  $k$ -hop dominating set is a subset  $D \subset V$  such that for each vertex  $v \in V \setminus D$  there exists a path of length  $l \leq k$  between  $v$  and at least one vertex  $d \in D$ . Thus  $D$  is a  $k$ -hop dominating set if it satisfies the following requirement:

$$\forall v \in V \setminus D : \exists u \in D, u \in N_k(v)$$

This means that each vertex is either part of  $D$  or in  $N_k(w)$  for any  $w \in D$ .

**Definition 5** (connected  $k$ -hop Dominating Set). A  $k$ -hop dominating set  $D$  is a connected  $k$ -hop Dominating Set if the induced subgraph  $G[D]$  is connected.

**Definition 6** (rooted connected  $k$ -hop Dominating Set). Let  $v \in V$  be the *root*. A rooted connected  $k$ -hop dominating set  $D$  is as connected  $k$ -hop dominating set which also includes  $v$ .

(Add a definition for what "connected" means)

## 4 Methods

We represent a plant's leaf as an undirected graph  $G = (V, E)$ . Each vertex  $v$  represents a leaf cell whereas a root  $v_{root}$  is predefined. Leaf cells are connected to its neighboring cells via plasmodesmata. Plasmodesmata are microscopic channels that link plant cells, enabling transport of nutrients and water amongst of other things . Those connections are represented by the edges  $E$ . We then look for a minimum set of nodes such that the whole leaf can still be supplied with water and the nutrients can be collected. For this purpose these vein cells need as well as the root need to be connected. Those cells form our solution for a rooted connected  $k$ -hop dominating set  $D$

(Irgendwie unterbringen, dass die non-vein-Zellen nicht direkt mit den vein-Zellen benachbart sein müssen.)

We use a node based ILP-Formulation to solve this special variant of the dominating set. We start by introducing a formulation for the general  $k$ -hop dominating set. As the objective function for our special variant remains the same, we then add constraints in a stepwise manner until we can present an ILP-formulation for the rooted connected  $k$ -hop dominating set.

As our implementation is node based we omit decision variables for edges, and instead only assign a variable  $x_v \in \{0, 1\}$  for every  $v \in V$ , with the interpretation  $x_v = 1 \Leftrightarrow v \in DS$ .

### 4.1 Minimum Dominating Set

As we try to minimize the number of vertices in the dominating set our ILP is given as:  
*objective target:*

$$\min \left\{ \sum_{v \in V} x_v \right\} \quad (1)$$

*subject to:*

$$\sum_{w \in N(v)} x_w + x_v \geq 1, \forall v \in V \quad (2)$$

The family of inequalities (2) says that each vertex itself or at least one of its neighbors has to be included in the dominating set.

### 4.2 Minimum $k$ -hop Dominating Set

The objective target for this problem is the same as (1). But the family of inequalities (2) is not valid for this case. Instead another family of inequalities is valid:

$$\sum_{w \in N_k(v)} x_w \geq x_v, \forall v \in V \quad (3)$$

This family of inequalities serves to model the requirement that each vertex or at least one member of the  $k$ -neighborhood has to be included in the dominating set. For the case  $k = 1$  this family is the same as (2).

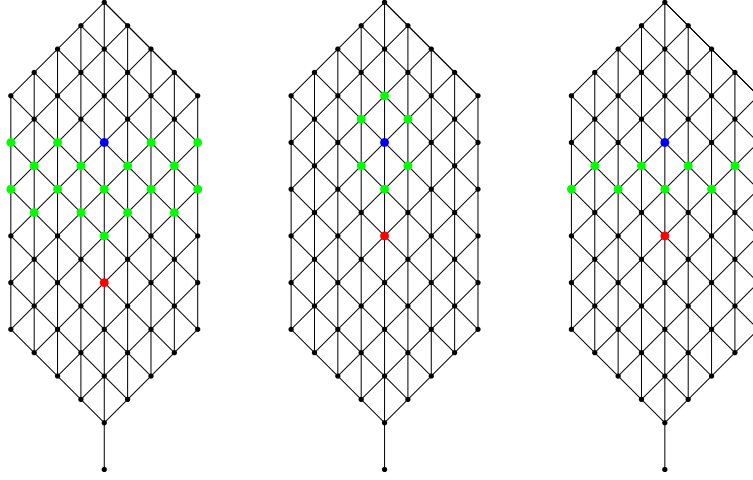


Figure 1: Illustration of vertex separators. In all three pictures the set of green nodes separates the blue and the red node. In the middle and on the right picture minimal separators are illustrated. If one of the green nodes is turned into a black node, the green set would not separate the blue and the red node anymore.

### 4.3 Connectivity

To enforce connectivity (using ILP) there are different approaches. As this is not trivial there have been many publications [1], [8], [2], [3], [17], [6] concerning this issue in the past years.

#### 4.3.1 Vertex separators

One approach is to use so called vertex separators. In [1] and [8] the authors used this approach to create ILP based algorithms to solve other graph theoretical optimization problems which require the solution to be connected. Bomersbach et al. [1] presented an ILP-formulation to solve the connected maximum coverage problem and Fischetti et al. [8] proposed ILP-formulations for different variants of the steiner tree problem. (that was solved in a branch and cut framework?). As [1] refers to [8] in terms of the connectivity constraints, both ILP-formulations use the same constraints to enforce connectivity. In [1], the authors compared the runtime of this implementation to previous proposed exact algorithms and to greedy approaches for the connected maximum coverage problem. In all test cases this implementation was significantly faster than all other exact algorithms. While in some cases the greedy algorithm was slightly faster, the proposed algorithm was more accurate. The algorithm from [8] significantly improved the runtime of an exact solver for all the different steiner tree problem variants and their proposed implementation won most of the different categories of the 11th DIMACS challenge on steiner trees.

Let  $v, w \in V$ . A  $v$ - $w$ -separator is a subset  $S_{v,w} \subset V$  such that  $G[V - S_{v,w}]$  has no path between  $v$  and  $w$ . A minimal  $v$ - $w$ -separator  $S_{v,w_{min}}$  is a  $v$ - $w$ -separator where no vertex can be removed. That is,  $S_{v,w_{min}} \setminus \{y\}$  is not a separator for  $v$  and  $w$ . Let  $S(v, w)$  (Use

different notation. This is misleading) denote the family of all minimal  $v$ - $w$ -separators. In [1] and [8] the following family of inequalities is used to enforce connectivity:

$$x_v + x_w \leq \sum_{u \in S_{v,w}} x_u + 1, \forall v, w \in V, v \neq w, \forall S_{v,w} \in S(v, w) \quad (4)$$

This inequalities require that for each combination of two vertices  $v$  and  $w$ , if both vertices included in the dominating set, at least one vertex from a minimum  $v$ - $w$ -separator, has also to be included.

In contrast to the problem from [1] we have a predefined root node which must be part of the solution. In [3] the authors introduced ILP-formulations for different problems motivated by forest planning. The objective of this problems is to find a profit maximizing harvest schedule, while old-growth-forest patches have to be conserved. Input instances are given as undirected graphs, with areas of the forest as nodes and edges between adjacent areas. There is one particular case where a predefined area is to be preserved plus all preserved areas need to be connected. This is very similar to our problem i.e. that there is a predefined root vertex and the requirement that all those vertices, which are included in the solution, need to be connected. Also minimum vertex separator constraints were used to enforce connectivity, but if a root node was present only those constraints which separate the connected component, that includes the root node, and all the other components were taken into account. The authors state that rooted inequalities are stronger as this was commonly noted in the literature on steiner tree problems. For our case we also only use constraints

$$x_v + x_w \leq \sum_{u \in S_{v,w}} x_u + 1, \forall v, w \in V, v \neq w, \forall S_{v,w} \in S(v, r) \quad (5)$$

for minimum vertex separators that include the root node.

The number of all minimum vertex separator constraints is potentially exponential [1]. Therefore in [1], [8] and [3] they treated these constraints as lazy constraints, which means in particular that none of those constraints are included in the initial model. Instead iteratively integer solutions are resolved [1], [8]. If such a solution is not connected, in [1] and [8] minimal vertex separators that separate single components are identified via a linear time algorithm, while in [3] a classical max-flow min-cut theorem is used to identify violated constraints.

Our algorithm to identify and add violated constraints is analogous the one from [1] with

the exception that we only search for violated constraints that include the root node.

---

**Algorithm 1:** Add violated constraints
 

---

```

1  $DS^* := \{v | x_v = 1\}$ 
2  $G' := G[DS]$ 
3 if  $G'$  is not connected then
4    $C :=$  set of all disjunct connected components
5    $c_{root} :=$  connected component that contains  $v_{root}$ 
6   for all components  $c$  in  $C \setminus \{c_{root}\}$  do
7      $v :=$  any node from  $c$ 
8      $s_1 := \text{findMinVertexSeparator}(G, DS^*, v \in c, v_{root}, c_{root})$ 
9      $s_2 := \text{findMinVertexSeparator}(G, DS^*, v_{root}, v \in c, c)$ 
10    for all  $w_1 \in c$  do
11      | add the following constraint to the model:  $\sum_{s \in s_1} x_s \geq x_{w_1} + x_{v_{root}} - 1$ 
12    end
13    for all  $w_2 \in c_{root}$  do
14      | add the following constraint to the model:  $\sum_{s \in s_2} x_s \geq x_{w_2} + x_v - 1$ 
15    end
16  end
17 end

```

---

This algorithm is executed each time an integer solution is resolved (using a branch and cut framework). Let  $D^*$  be an integer, not necessarily connected, solution. Let  $C$  be the set of all connected components from the graph  $G' = G[D^*]$  and let  $c_r$  be the component that contains the root node  $v_r$ . Then the algorithm detects for all single components  $c \in C \setminus \{c_r\}$  one minimal vertex separator that separates  $c$  and the component  $c_r$ . The constraints concerning these separators are then added to the model and the cutting plane procedure continues. It is important to mention that there is in general more than one minimal vertex separator which separates two arbitrary components. The Algorithm 2 detects exactly one, i.e., the separator, that is closest to the first component. By executing the Algorithm 2 with every component  $c \in C \setminus \{c_r\}$  as first component and  $c_r$  as second component and vice versa, we ensure that a minimal vertex separator that is closest to each of the components is added.

---

**Algorithm 2:** findMinVertexSeparator( $G, DS^*, v \in c_v, w, c_v$ )
 

---

```

1  $N(c_v) :=$  neighbors of nodes of  $c_w$  in  $G$  (Maybe use the formal definition from methods?)
2  $G' := G$  with all edges between vertices in  $c_v \cup N(c_v)$  removed
3  $R_w :=$  vertices that can be reached from  $w$  in  $G'$ 
4 return  $N(c_v) \cap R_w$ 

```

---

The algorithm above detects a minimal vertex separator that separates the node  $w$  and the connected component  $c_v$ . It is taken from [1] although Bomersbach et al. [1] took it initially from [8]. With this method the minimal vertex separator is found that is closest to the component  $c_v$ . In figure 2 one can see an illustration of the process. Suppose the red marked nodes are an unconnected solution  $D^*$ . The set of blue marked nodes is the minimal separator that is closest to the connected component on the upper graph while

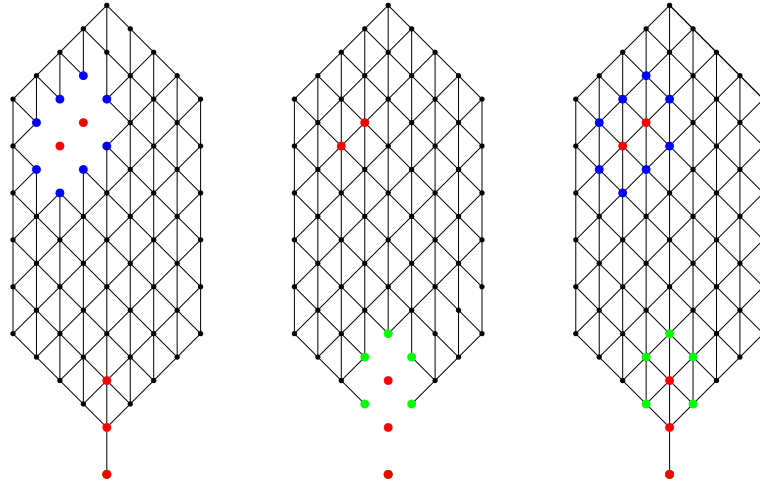


Figure 2:

the set of green marked nodes is the minimal separator that is closest to the component containing the root. On the picture in the middle and the right one can see the line 2 of the algorithm 2. As one can see, after removing all edges between the components and its neighborhood the blue marked nodes on the middle picture and the green marked nodes on the right picture are still reachable from the other component. Therefore the algorithm returns this selection of nodes as minimal vertex separator.

We add an additional constraint to the model to tighten up the feasible region and to prevent unnecessary iterations.

$$x_v \leq \sum_{w \in N(v)} x_w, \forall v \in V \setminus \{v_{root}\} \quad (6)$$

This constraint demands that for each vertex which is part of the dominating set at least one of its neighbors is also included. In [1] and [8] this constraint is also part of the model. As the neighborhood of a single vertex is always a minimal vertex separator that separates this node from any other vertex outside the neighborhood, this constraint is valid. We exclude the root node  $v_{root}$  to prevent that for the case of a valid solution that only contains 1 single vertex another one is added unnecessarily.

#### 4.3.2 Miller-Tucker-Zemlin Constraints

There are also formulations to enforce connectivity that only need a polynomially number of constraints. These constraints are not added lazily but instead all added initially. There exist some approaches that are based on the construction of a spanning tree. We have implemented one of these formulations in the scope of this thesis. This approach was used in [6] to generate an ILP-formulation for the Minimum Connected Dominating Set problem. In the scope of the publication four different formulations, all based on creating a spanning tree, were compared (experimentally). This particular formulation outperformed all three others on all six input graphs. With increasing size the difference in the runtime became larger.

In the scope of this thesis we therefore only compared this one with the vertex separator version.

The Miller Tucker Zemlin constraints were initially introduced to present an ILP-formulation for the Traveling Salesman Problem with only polynomial many constraints. Let  $G = (V, E)$  be our undirected input graph. We follow the description from [6] by defining  $G_d = (V \cup \{n+1, n+2\}, A)$  as directed graph, whereas  $A = \{(n+1, n+2)\} \cup \{\bigcup_{i=1}^n (n+1, i), (n+2, i)\} \cup E'$  and  $E' = \{(j, i), (i, j) : i, j \in E\}$ . Note that  $E'$  is the bidirected version of  $E$ , that means, we add an arc in both directions for every edge in  $E$ . Let  $n = |V|$ . We create two additional nodes  $n+1$  and  $n+2$ . Additionally we add an arc from  $n+1$  and from  $n+2$  to every vertex  $v \in V$ , and we add an arc from  $n+1$  to  $n+2$ . The idea behind the constraints is to create a directed spanning tree  $T_d = (V \cup n+1, n+2, E_d)$  on  $G_d$ , such that vertex  $n+1$  is a root and holds an arc (on  $T_d$ ) to every vertex, which is not part of  $D$  and to  $n+2$ . While  $n+2$  holds an arc to a node  $v_r$  within  $D$ . All the other nodes form a tree with root  $v_r$ .

Let  $y_{ij} \forall (i, j) \in A$  be decision variables, that specify whether the arc  $(i, j)$  is part of the spanning tree  $T_d$ . Let  $u_i \in \mathbb{Z}_+, \forall i \in V \cup \{n+1, n+2\}$  be auxiliary variables, that specify in which step the arc is passed starting from  $n+1$ . Those auxiliary variables eliminate sub tours as they also do in the Traveling Salesman Problem.

In the following we give a full ILP-formulation for to enforce connectivity via MTZ-constraints.

$$\sum_{i \in V} y_{n+2, i} = 1 \quad (7)$$

$$\sum_{(i, j) \in A} y_{ij} = 1, \forall j \in V \quad (8)$$

$$y_{n+1, i} + y_{ij} \leq 1, \forall (i, j) \in E' \quad (9)$$

$$(n+1) * y_{i, j} + u_i - u_j + (n-1) * y_{ji} \leq n, \forall (i, j) \in E' \quad (10)$$

$$(n+1) * y_{i, j} + u_i - u_j + (n-1) * y_{ji} \leq n, \forall (i, j) \in A \setminus E' \quad (11)$$

$$y_{n+1, n+2} = 1 \quad (12)$$

$$u_{n+1} = 0 \quad (13)$$

$$1 \leq u_i \leq n+1, i \in V \cup \{n+2\} \quad (14)$$

$$x_i = 1 - y_{n+1, i}, \forall i \in V \quad (15)$$

Constraints (7) ensure that there is exactly one root for the dominating set. In our case we replace this inequality by the following:  $y : n+2, v_{root} = 1$  and  $y_{n+2, i} = 0, \forall i \in V \setminus \{v_{root}\}$ .

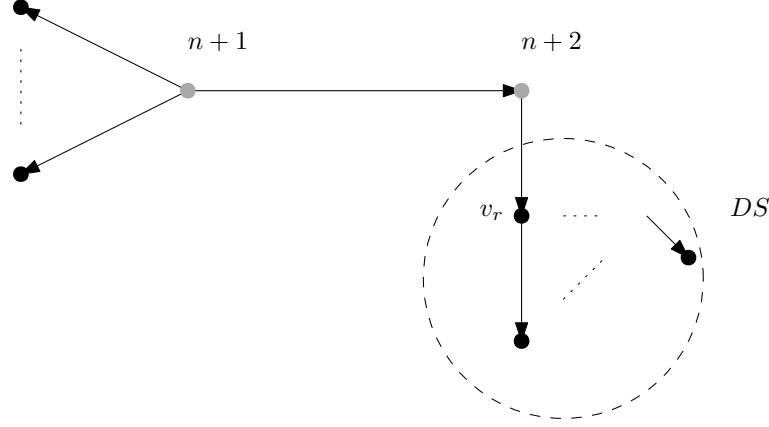


Figure 3: Illustration of the principle. The dashed circle outlines the dominating set. All vertices, that are connected to  $n + 1$  are not part of the dominating set.

Constraints (8) enforce that each node on the spanning tree  $T_d$  has exactly one incoming arc. While constraints (9) require that all the nodes from  $T_d$  are either connected to each other or have an incoming arc from node  $n + 1$ , the node which marks nodes that are not part of  $D$ . With the exception of the term  $(n - 1)y_{ji}$  the constraints (10) and (11) are the original MTZ constraints to eliminate sub tours from [12]. The mentioned term is an improvement from [5]. Constraint (12) demands the arc  $(n + 1, n + 2)$  to be included in  $T_d$ . Constraints (14) define the value of ranges for the auxiliary variables  $u_i$ . As these variables specify in which step the arc to node  $i$  is passed, only values from  $1 - n + 1$  (the number of incoming arcs) can be assigned to it. Finally the last constraints (15) ensure that if there is no incoming arc from node  $n + 1$  to a node  $i$ , then  $i$  must be included in  $D$  and vice versa (I think it is important to mention the backward direction as otherwise the impression could arise that only the MTZ constraints decide which vertices are included).

We combine the above mentioned ILP-formulation for MkCDS with this formulation to enforce connectivity. The solution of this formulation then is a optimal connected solution with  $v \in D \Leftrightarrow x_v = 1$ . As previously mentioned this formulation only needs polynomial many constraints. More precisely there are  $(|V| + 2) + (2|E| + 2|V| + 1) = O(|E| + |V|)$  decision variables and  $1 + |V| + 2|E| + 2|E| + (2|V| + 1) + 1 + 1 + |V| = O(|E| + |V|)$  constraints.

#### 4.4 Minimum connected $k$ -hop Dominating Set

A connected  $k$ -hop dominating set is a  $k$ -hop dominating set  $DS$  such that  $G[DS]$  is connected. (Maybe refer to methods as this is redundant?). Its ILP-Formulation consists of the objective target (1) and constraints (3) and a collection of constraints to induce connectivity (In the future different types of potential constraints should be added).

#### 4.5 Minimum rooted connected $k$ -hop Dominating Set

Let  $v_{root} \in V$  be the predefined root. The ILP-Model of this problem is the ILP-Model of 4.4 enriched with following constraint.

$$x_{v_{root}} \geq 1 \tag{16}$$

#### 4.6 Additional methods to tighten up the space of feasible solutions

In the scope of this thesis additional constraints were tested, that should tighten up the space of feasible solutions further. As it can potentially cost much time to create unconnected solutions, we want to prevent unnecessary iterations.

##### 4.6.1 Intermediate node constraint

In the paper about the Steiner Tree Problem [8] one inequality to reduce the number of unconnected feasible solutions is proposed. It demands that for each node in the solution, which is not a predefined terminal, to have two neighbors in the solution. A node that has two neighbors in the solution can be seen as an intermediate node. Let  $T$  be the set of all terminals. The inequality can formally be described as

$$2 * x_v \leq \sum_{w \in N(v)} x_w, \forall v \in T$$

Unfortunately this inequality can not be applied to our problem without potentially excluding optimal solutions. By this inequality solutions can be generated, which have additional nodes at the end of branches, that are not necessary for the MkCDS but that are necessary to fulfill this inequality. In our case we would need to require that for each vertex, which is not at the end of a branch, this inequality needs to be satisfied. But we can not decide which node will be at the end of a branch in advance.

In figure (4) there is an illustration that compares one optimal solution without this constraint on the left and one with this constraint on the right. On the right hand side the end of a branch is circled to outline the additional node generated by this constraint.

Even if the generated solutions are not inevitably optimal, the generated solutions are close to an optimal solution (in terms of the number of nodes). At the same time this constraint reduces the runtime in many instances drastically. That is why it can be considered to generate approximative solutions using this constraint. This constraint can also be used to generate a sufficient upper bound in the branch and cut process. But for the most instances this is not necessary as a sufficient upper bound is found quickly. It needs much more time to find a sufficient lower bound and to close the gap.

##### 4.6.2 Reduce path length

To exclude such solutions which contain single (unconnected) nodes, that are close to the rim we invented constraints to reduce the length of each path between the nodes of a

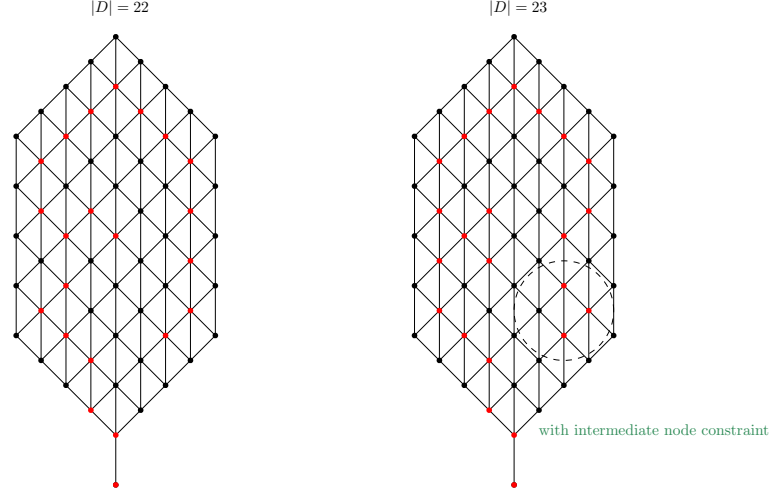


Figure 4: The dashed circle outlines the necessity to have connected triplets at the end of a branch

solution and the root node. The length of each path to an arbitrary node is naturally limited by the number of members of the dominating set. In the extreme there is one single branch, which has exactly the length of the number of all members of the dominating set. In the case of more than one branch the upper bound is still valid. On that account we started by following the naive approach to limit the path from the root node to each member of  $D$  by the size of  $D$ . The formal description is

$$\sum_{v \in V} x_v \geq \text{shortestpath}\{v_{\text{root}}, v\}, \forall v \in V \setminus \{v_{\text{root}}\} \quad (17)$$

As this constraint did not reduce the runtime we tried to refine it. There are too many possible (unconnected) solutions where the constraint is satisfied. Figure 5 shows one of it.

This circumstance leads to the following constraint, that makes use of the Gaussian sum formula. The idea is still to limit the distance between the root node  $v_{\text{root}}$  and all the members of  $D$ . In this advanced formulation we limit the sum of the distances to  $\sum_i^{|D^*|} i$ . This constraint cuts off unconnected solutions that are valid using only the previous constraint (??). But as our tests revealed this constraint did not generate a performance boost but even increased the runtime(As it probably adds too much complexity to the model).

(Maybe also mention that this constraint in isolation allows solutions which are forbidden using the previous one)

#### 4.6.3 Preventively adding separators

We use the lazy approach to prevent that too many constraints are added that are not mandatory to generate sufficient solutions. In despite of this we evaluated if adding particular separator constraints could reduce the runtime. It can be that a more appropriate LP bound is generated using this approach and unnecessary iterations can be prevented.

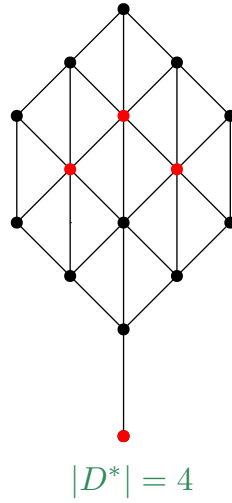


Figure 5: An unconnected solution where the path length constraint is satisfied.

## 5 Implementation

Now, we specify the implementation of the ILP-formulations from the Methods section. We implemented the ILP-formulations and Algorithms 1 and 2 using Python version 3.7.5. As branch and cut framework and MIP-solver we use Gurobi version 9.0.2. Gurobi offers a Python interface called *gurobipy* which can be called from inside python scripts. This interface offers access to functions included in Gurobi. Our implementation is embedded in a conda package. The package is called *k\_hop\_dominating\_set\_gurobi*. The source of the package can be found on <https://gitlab.cs.uni-duesseldorf.de/albi/albi-students/bachelor-mario-surlemont/>. The package itself can be build via

```
conda build .
```

after heading into the directory. To build the package *conda-build* needs to be installed.

Afterwards the package can be installed via

```
conda install --use-local k_hop_dominating_set_gurobi
```

It holds the dependencies *networkX*, *matplotlib.pyplot* and *gurobipy*.

The vertex separator constraints as well as the MTZ constraints can be chosen. The choice can be specified via the optional argument *-mtz*, for the use of MTZ-constraints. By default the vertex separators are chosen. If required the additional constraints that have been presented in the method section can also be added to the model via the optional argument *-imn* | *-rpl* | *-gaus* | *-pre* with *rpl* as abbreviation for the naive constraint to reduce the path length and *gaus* as abbreviation for the constraint involving the gaussian sum formula. The argument *-pre* adds separators to the model before the solution process is started. When the intermediate node constraint is added via *-imn* the generated solutions might not be optimal anymore.

As input *networkx* graphs stored as “.graphml” or “.gml” can be used. Also “.lp” files

from [10] can be used. A full programm call is

```
k_hop_dominating_set_gurobi -g graph.graphml -k k [OPTIONS]
```

with [OPTIONS] = {-mtz, -inm, -rpl, -gaus, -pre}.

If the vertex separators are chosen to induce connectivity a lazy approach is used. Gurobi offers a callback function which is called during the solution procedure when different events occur. The function offers a code that communicates the type of the occurred event. When the callback code *MIPSOLVE* is communicated a mixed ILP-solution was generated. That is a solution where those variables that must be integers are integers while those variables which do not need to be integers can be arbitrarily chosen (with respect to the inequalities). As we only have integer variables in our model the *MIPSOLVE* code tells us that an integer solution  $D^*$  was generated. In this case we check whether the graph is connected. We use a function that is included in *networkx* to check if the graph  $G[D^*]$  is connected. If not, algorithm 1 is used to add the corresponding constraints. After a valid solution was found the inputgraph it is plotted via *matplotlib.pyplot*. The members of the dominating set are displayed in red while all the other vertices are displayed green. The console output shows information about the solving process and the solution. Such as the current upper bound and lower bound.

## 6 Results

This section shows our results of the runtime for the Minimum Connected rooted k-hop Dominating Set problem. We test the graphs that represent plant leaves from [10] as well as randomly generated graphs and grid graphs. At first we briefly describe the graphs from [10] and our other test graphs. A more detailed description of the leaf graphs can be taken from [10]. All tests have been performed using a notebook with an Intel Core i7-4720HQ CPU @ 2.60GHz x 8 and 8 GB of RAM under Ubuntu 18.04.14 LTS and in all test cases we only looked for one single optimal solution.

As leaf graphs we use the instances *small-leaf*, *middle-leaf*, *bigger-leaf*, *maple* and *asymmetric*. The instances *small-leaf*, *middle-leaf* and *bigger-leaf* are similar in their structure. Each of the three graphs has the root at the bottom side and a symmetrical composition. They only differ in the number of nodes. The smallest graph *small-leaf* has only 15 nodes while *middle-leaf* has 62 nodes and *bigger-leaf* has 71 nodes. While *maple* represents a maple's leaf having 118 nodes, *asymmetric* is inspired by an alocasia leaf. The peculiarity here is that it has the root in the middle of the leaf. It has 378 nodes.

The following figure illustrates the 5 different leaf graphs.

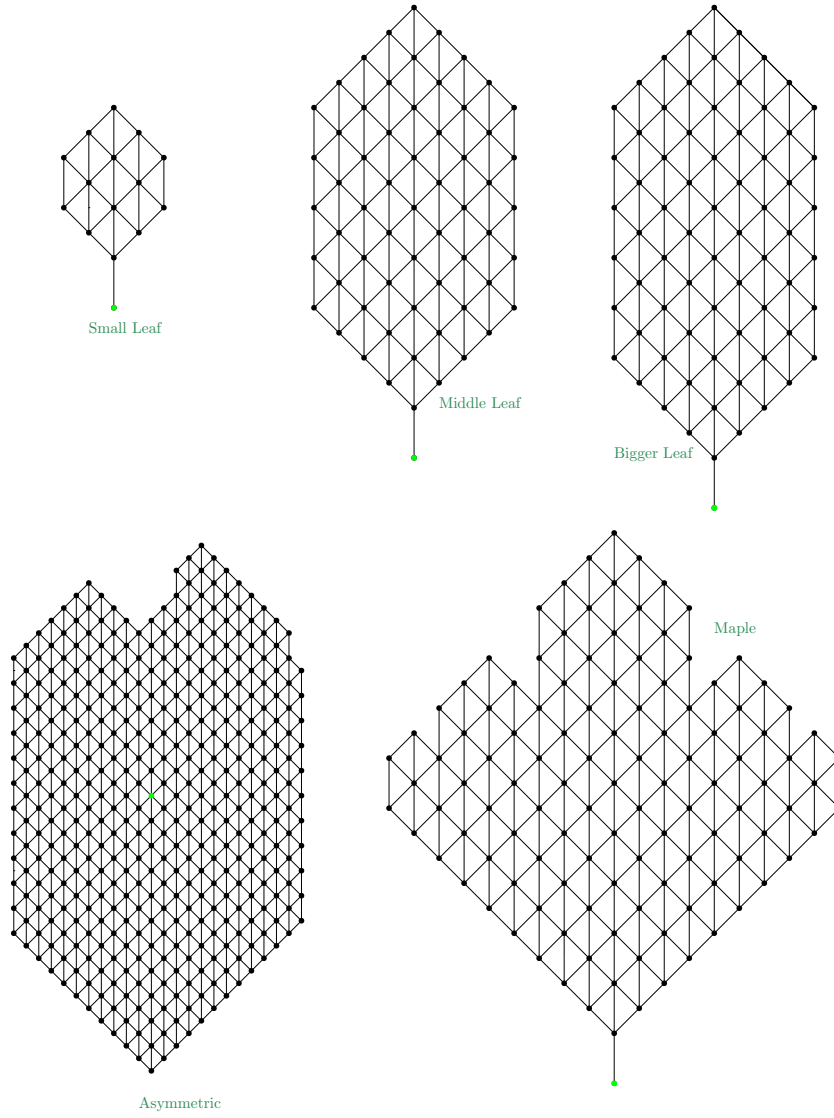


Figure 6: The graphs that represent plant leaves

First of all we introduce a table demonstrating different characteristics of the leaf graphs. The first two columns show the size of the graph, i.e., the number of nodes and edges. The density is shown in the third column. It is a measure that indicates the relative amount of the number of edges a graph has to the theoretical number a graph can have. Additionally the maximal, the average and the minimal node degree is shown. These parameters imply if a graph has at least one node with a much higher degree than the average or if the degrees are equally distributed.

name	V	E	density	max. degree	avg degree	median degree	min degree
small-leaf	15	30	0.29	6	4	4	1
middle-leaf	62	152	0.08	6	5	6	1
bigger-leaf	71	182	0.07	6	5	6	1
maple	118	308	0.04	7	5	6	1
asymmetric	378	1071	0.02	8	6	6	3

Table 1: The characteristics of the leaf graphs

We then continue with the runtime of our ILP-implementation using the leaf graphs as input. The following tables present the runtime in seconds as well as the number of constraints that were lazily added in the solution process. The last column shows the size of an optimal solution, i.e., the number of nodes that form a minimum dominating set. For the case that the solution process took more than 1000 seconds we state the upper bound and the lower bound that were determined within this time. The upper bound specifies the smallest solution that was found until the time was over. This means that an optimal solution will not be larger than the upper bound. In contrast the lower bound gives the smallest theoretical possible size of an optimal solution to that time. Let  $U$  be an upper bound and  $L$  be a lower bound. In the column *optimal* we used the denotation  $[U, L]$ .

name	k	# lazily added constraints	runtime(s)	optimal
small-leaf	1	9	0.01237	6
	2	4	0.007257	3
	3	0	0.007005	2
middle-leaf	1	4945	1099.324462	[22,21]
	2	2043	7.006414	14
	3	811	0.950746	10
bigger-leaf	1	6726	1058.414758	[25, 22]
	2	377	18.178422	15
	3	1266	2.606486	11
maple	1	194321	1129.807776	[41,31]
	2	9621	1074.40126	[26,20]
	3	8029	1532.499756	[20,17]
asymmetric	1	34255	1010.642396	[219, 80]
	2	2706	1065.584708	[161,38]
	3	13947	1026.665897	[63, 22]

Table 2: Minimum Connected rooted  $k$ -hop Dominating Set Results on the leaf graphs

With increasing parameter  $k$  the runtime decreases significantly. Additionally this table indicates a relation between the number of constraints that are added lazily and the runtime. Besides some outliers it seems like a high number of lazily added constraints implies a higher runtime. The more constraints that are added the more frequent unconnected integer solutions are found in the solution process. This effect occurs especially on input instances that have many symmetrical solutions which are unconnected. If the

input graph only has nodes that have a degree close to the average degree, then more likely this instance has many different symmetrical solutions. In such instances there is no node that is so valuable that it has to be included in the solution. If an unconnected integer solution is generated violated constraints are added to the model. After adding these constraints it most likely is cheaper to swap the nodes and use nodes where no violated constraints have been added yet than to use the same nodes and add those nodes, that the added constraints demand. On graphs where some nodes exist that have a significant higher degree than the average adding constraints more likely will not exclude them from a solution as they cover to many other vertices. This effect is roughly indicated by the number of lazily added constraints. If only a few constraints were added then there probably will not have been many options to swap valuable nodes without creating to many costs.

With increasing size, i.e., number of nodes a graph has, the density of our graphs decreases. The density of the graph is another indicator that roughly implies the runtime [2]. Especially on graphs with unequal distribution of node degrees. As with increasing size the density decreases on our graphs, the tests can not clearly indicate if the size is purely responsible for the runtime or if the density also has an influence. In the following we will test random generated graphs that have different size and for each size 10 different levels of density. On this graphs the density clearly is the determining factor for the runtime.

The next table shows the characteristics of the random graphs.

name	V	E	density	max. degree	avg. degree	median degree	min degree
GNM_50_122	50	122	0.1	9	5	5	1
GNM_50_245	50	245	0.2	15	10	9.5	6
GNM_50_368	50	368	0.3	22	15	15	7
GNM_50_490	50	490	0.4	28	20	19	13
GNM_50_612	50	612	0.5	35	24	24	17
GNM_50_735	50	735	0.6	36	29	30	17
GNM_50_858	50	858	0.7	41	34	34.5	28
GNM_50_980	50	980	0.8	44	39	39	34
GNM_50_1102	50	1102	0.9	49	44	44	38
GNM_50_1225	50	1225	1.0	49	49	49	49
GNM_100_495	100	495	0.1	17	10	10	1
GNM_100_990	100	990	0.2	29	20	19.5	8
GNM_100_1485	100	1485	0.3	40	30	29	21
GNM_100_1980	100	1980	0.4	51	40	40	26
GNM_100_2475	100	2475	0.5	62	50	50	35
GNM_100_2970	100	2970	0.6	70	59	60	48
GNM_100_3465	100	3465	0.7	80	69	69	56
GNM_100_3960	100	3960	0.8	88	79	80	70
GNM_100_4455	100	4455	0.9	95	89	89	83
GNM_100_4950	100	4950	1.0	99	99	99	9
GNM_250_3112	250	3112	0.1	38	25	24.5	14
GNM_250_6225	250	6225	0.2	67	50	49.5	28
GNM_250_9338	250	9338	0.3	91	75	75	51
GNM_250_12450	250	12450	0.4	119	100	100	82
GNM_250_15562	250	15562	0.5	144	124	124	109
GNM_250_18675	250	18675	0.6	173	149	150	131
GNM_250_21788	250	21788	0.7	195	174	174	156
GNM_250_24900	250	24900	0.8	216	199	199	180
GNM_250_28012	250	28012	0.9	236	224	224	210
GNM_250_31125	250	31125	1.0	249	249	249	249
GNM_500_12475	500	12475	0.1	68	50	50	30
GNM_500_24950	500	24950	0.2	128	100	100	68
GNM_500_37425	500	37425	0.3	183	150	150	118
GNM_500_49900	500	49900	0.4	243	200	200	166
GNM_500_62375	500	62375	0.5	286	250	250	207
GNM_500_74850	500	74850	0.6	328	299	299	266
GNM_500_87325	500	87325	0.7	380	349	350	318
GNM_500_99800	500	99800	0.8	427	399	399	372
GNM_500_112275	500	112275	0.9	467	449	450	427
GNM_500_124750	500	124750	1.0	499	499	499	499

Table 3: The characteristics of the random graphs

We have random graphs of four levels of size( $|V| = 50; 100; 250; 500$ ). For each of these levels we have ten levels of density(0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0) to explore

particularly its influence on the runtime. In the following table we only present the results for the density levels 0.1, 0.5 and 0.9 . The appendix contains the complete tables. The results clearly show that, despite the larger size of the random graphs, the runtime is significantly shorter than on the leaf graphs. The density here seems to be a reasonable parameter that implies the runtime. On dense graphs few nodes are mandatory to form a dominating set. This allows to find an optimal solution faster.

name	k	# lazily added constraints	runtime(s)	optimal
GNM_50_122	1	66	0.034878	11
GNM_50_122	2	67	11	0.03795
GNM_50_122	3	0	2	0.01651
GNM_50_612	1	0	0.017783	4
GNM_50_612	2	0	1	0.002223
GNM_50_612	3	0	1	0.002541
GNM_50_1102	1	3	0.019566	3
GNM_50_1102	2	0	1	0.012025
GNM_50_1102	3	0	1	0.012196
GNM_100_495	1	113	0.376731	14
GNM_100_495	2	6	4	0.108993
GNM_100_495	3	0	1	0.026969
GNM_100_2475	1	0	0.045136	4
GNM_100_2475	2	0	1	0.004791
GNM_100_2475	3	0	1	0.006448
GNM_100_4455	1	0	0.00505	2
GNM_100_4455	2	0	1	0.003927
GNM_100_4455	3	0	1	0.004094
GNM_250_3112	1	0	1017.303471	[17;15]
GNM_250_3112	2	0	2	0.270981
GNM_250_3112	3	14	1	0.141794
GNM_250_15562	1	0	12.29	5
GNM_250_15562	2	109	1	0.257635
GNM_250_15562	3	109	1	0.267159
GNM_250_28012	1	0	0.024473	2
GNM_250_28012	2	0	1	0.018999
GNM_250_28012	3	0	1	0.023179
GNM_500_12475	1	42	1004.920676	[21;13]
GNM_500_12475	2	0	2	1.123904
GNM_500_12475	3	0	1	0.634489
GNM_500_62375	1	0	178.495614	5
GNM_500_62375	2	0	1	0.29011
GNM_500_62375	3	0	1	0.544754
GNM_500_112275	1	0	0.189313	2
GNM_500_112275	2	0	1	0.148031
GNM_500_112275	3	0	1	0.205316

Table 4: Minimum Connected rooted  $k$ -hop Dominating Set Results on the random graphs

We also tested another class of graphs on their runtime. The structure of our leaf graphs is similar in the manner that all have a fixed neighborhood of 6 vertices, all are planar and almost all nodes have the same degree. Many grid graphs also have all these characteristics. This is why we tested our implementation also on grid graphs. Here we tested graphs that are quadratic as well as graphs that are more oblong. Especially on quadratic graphs the same behavior like on the leaf graphs has occurred. Here also comparatively many constraints were added lazily. Which indicates that here also many unconnected integer solutions were created. It seems like the “gridness” of a graph is the crucial factor that pushes the runtime over a reasonable extent. The gridness can be defined as the combination of the three described properties from the beginning. On grid graphs the ASP-version also performs much better than the ILP-Version.

Here also a short overview about the characteristics of the grid graphs.

name	V	E	density	max. degree	avg degree	median degree	min degree
GRID_6_4	24	38	0.14	4	3	3	2
GRID_8_8	64	112	0.06	4	4	4	2
GRID_16_4	64	108	0.05	4	3	3	2
GRID_18_2	36	52	0.08	3	3	3	2
GRID_32_2	64	94	0.05	3	3	3	2

Table 5: The characteristics of the grid graphs

name	k	# lazily added constraints	runtime(s)	optimal
GRID_6_4	1	178	0.054271	11
GRID_6_4	2	74	0.044738	7
GRID_6_4	3	112	0.05603	6
GRID_8_8	1	6451	774.59	26
GRID_8_8	2	865	81.970768	18
GRID_8_8	3	3634	15.546363	15
GRID_16_4	1	31	42.568463	28
GRID_16_4	2	2353	1.405127	17
GRID_16_4	3	2789	7.9726	16
GRID_18_2	1	383	0.116538	18
GRID_18_2	2	394	0.147668	17
GRID_18_2	3	319	0.17269	16
GRID_32_2	1	1090	0.261853	32
GRID_32_2	2	791	0.341931	31
GRID_32_2	3	876	0.286863	30

Table 6: Minimum Connected rooted  $k$ -hop Dominating Set Results on the grid graphs

Another important factor that comes with a long runtime for the ILP-version is when there is a large gap between the number of an unconnected solution and the number of a connected solution for an instance. The other way round the ILP-version performed good on graphs where the gap was tight such that only a few nodes needed to be added to an unconnected solution.

In the method section (refer at this place) we introduced the MTZ constraints (also refer) to induce connectivity. The next table shows the runtime of three graphs using the MTZ constraints.

name	k	runtime(s)	optimal
GRID_8_8	2	61.264993	18
middle-leaf	2	1229.65	[14;13]
bigger-leaf	3	197.451639	11

Table 7: Minimum Connected rooted  $k$ -hop Dominating Set Results using the MTZ constraints

The version using the vertex separator is in all testes cases many times faster. The version using the MTZ constraints seems not to be a reasonable alternative.

Now we study the case when some of the vertex separator constraints are preadded to the model. We preadded for all combinations  $c_v$  of a vertex  $v$  and its neighborhood  $N(v)$  the vertex separators that separate  $c_v$  and the root vertex  $v_r$ . As the following table reveals this generates a significant speedup to the runtime. However the bigger leaf instance can still not be solved optimal under 1000 seconds. In all test cases the ILP-version with preadded separators performed better than the ASP-version. Still many separator constraints needed to be added lazily. If these constraints can be identified in advance this could generate another speedup. At this point preadding the described separators itself does not improve the ILP-implementation in a manner that the runtime is satisfying.

name	k	# lazily added constraints	runtime(s)	optimal
small-leaf	1	0	0.003599	6
s middle-leaf	1	3699	710.18652	22
bigger-leaf	1	7105	1080.973378	[25, 23]
GRID_8_8	1	1061	40.536663	26
GRID_16_4	1	57	27.854317	28

Table 8: Minimum Connected rooted  $k$ -hop Dominating Set Results with preadded vertex separator constraints

At last we present tables that show the effect of the additional constraints (referenz) introduced in the method section(ref) on some graphs.

The first table shows the effect of the *intermediate node constraint*(ref) from [8]. To recap this constraint demands that every vertex that is part of the dominating set needs at least two neighbors which are also members of the dominating set. Roughly speaking every node of the dominating set(except for the root) needs to be an intermediate node. This constraint reduces the runtime drastically. However in most cases including this constraint adds nodes to the solution that would not be included without. For example the instances *middle-leaf* and *bigger-leaf* have one extra node in the optimal solution when this constraint is included.

name	k	# lazily added constraints	runtime(s)	optimal
small-leaf	1	8	0.011553	6
middle-leaf	1	643	0.723175	23
bigger-leaf	1	1157	1.396552	25
maple	1	1405	439.99668	41
asymmetric	1	4294	1006.370245	[256, 62]

Table 9: Minimum Connected rooted  $k$ -hop Dominating Set Results with IMN constraint

The next table shows the results using the naive constraint to reduce the path length from the root to members of the dominating set. It does not reduce the runtime but even increases it. For the cases where we stopped the solution process after a fixed time span the upper bounds and lower bounds are worse than without this constraint. However in some cases this constraint reduces the number of lazily added constraints which is an indicator that the room of possible unconnected solutions was reduced. But this effect did not reduce the runtime. Probably this constraint added complexity to the model which increased the runtime instead.

name	k	# lazily added constraints	runtime(s)	optimal
small-leaf	2	9	0.008948	3
middle-leaf	2	109	10.936048	14
bigger-leaf	2	67	23.457956	15
maple	2	5804	1011.766479	[26,20]
asymmetric	2	17391	1114.582689	[190,81]

Table 10: Minimum Connected rooted  $k$ -hop Dominating Set Results with SPL constraint

The additional constraint that uses the Gaussian sum formula even performed drastically worse. The runtime increased significantly as this constraint adds a high degree of complexity to the model.

name	k	# lazily added constraints	runtime(s)	optimal
small-leaf	2	8	0.009487	3
middle-leaf	2	457	87.4349	14
bigger-leaf	2	1566	317.235052	15

Table 11: Minimum Connected rooted  $k$ -hop Dominating Set Results with GAUS constraint

When using both constraints in conjunction the constraint with the Gaussian sum formula dominates the runtime.

name	k	# lazily added constraints	runtime(s)	optimal
small-leaf	2	0	0.004869	3
middle-leaf	2	1198	87.231026	14
bigger-leaf	2	882	317.309151	15

Table 12: Minimum Connected rooted  $k$ -hop Dominating Set Results with SPL and GAUS constraint

As we compared our ILP-version to the ASP-version from [10] the following tables list the runtime of the different graphs using the ASP-version.

We start with our leaf graphs. This table clearly shows that the ASP-version performs much better on these graphs. As for example for the *middle-leaf* instance with parameter  $k = 1$  the ASP-version finds a solution in 154 seconds, after 1100 seconds the ILP-version does not find a solution(ref).

name	k	# lazily added constraints	runtime(s)	optimal
small-leaf	1	9	0.008	6
small-leaf	2	4	0.009	3
small-leaf	3	0	0.009	2
middle-leaf	1	4945	153.605	22
middle-leaf	2	2043	0.597	14
middle-leaf	3	811	0.038	10
bigger-leaf	1	6726	1002.022	[25, 24]
bigger-leaf	2	377	1.735	15
bigger-leaf	3	1266	0.069	11
maple	1	194321	1129.807776	[41,31]
maple	2	9621	1008.548	[26,24]
maple	3	8029	1006.839	[21,20]
asymmetric	1	34255	1011.016	[164, 29]
asymmetric	2	2706	1009.839	[102,20]
asymmetric	3	34255	1012.392	[69, 18]

Table 13: Minimum Connected rooted  $k$ -hop Dominating Set Results on the leaf graphs using ASP

We continue with the runtime of the ASP-version on random graphs. This tables clearly indicate that the ILP-version performs better on random graphs.

name	k	runtime(s)	optimal
GNM_50_122	1	0.014	11
GNM_50_122	2	5	0.025
GNM_50_122	3	2	0.022
GNM_50_612	1	0.055	4
GNM_50_612	2	1	0.038
GNM_50_612	3	1	0.041
GNM_50_1102	1	0.052	3
GNM_50_1102	2	1	0.052
GNM_50_1102	3	1	0.051
GNM_100_495	1	32.451	14
GNM_100_495	2	4	0.084
GNM_100_495	3	1	0.082
GNM_100_2475	1	0.655	4
GNM_100_2475	2	1	0.151
GNM_100_2475	3	1	0.163
GNM_100_4455	1	0.253	2
GNM_100_4455	2	1	0.220
GNM_100_4455	3	1	0.227
GNM_250_3112	1	1017.204	[23;9]
GNM_250_3112	2	2	0.521
GNM_250_3112	3	1	0.529
GNM_250_15562	1	1008.099	[5;4]
GNM_250_15562	2	1	0.972
GNM_250_15562	3	1	0.967
GNM_250_28012	1	3.400	2
GNM_250_28012	2	1	1.453
GNM_250_28012	3	1	1.489
GNM_500_12475	1	1016.396	[29;7]
GNM_500_12475	2	2	2.314
GNM_500_12475	3	1	2.297
GNM_500_62375	1	1006.141	[6;4]
GNM_500_62375	2	1	4.218
GNM_500_62375	3	1	4.513
GNM_500_112275	1	8.705	2
GNM_500_112275	2	1	6.268
GNM_500_112275	3	1	6.490

Table 14: Minimum Connected rooted  $k$ -hop Dominating Set Results on the random graphs using ASP

On the other hand the ASP-version performs better on the grid graphs. This is as we expected.

name	k	runtime(s)	optimal
GRID_6_4	1	0.009	11
GRID_6_4	2	0.011	7
GRID_6_4	3	0.013	6
GRID_8_8	1	92.739	26
GRID_8_8	2	1.534	18
GRID_8_8	3	1.747	15
GRID_16_4	1	0.281	28
GRID_16_4	2	0.014	17
GRID_16_4	3	0.023	16
GRID_18_2	1	0.010	18
GRID_18_2	2	0.011	17
GRID_18_2	3	0.013	16
GRID_32_2	1	0.015	32
GRID_32_2	2	0.015	31
GRID_32_2	3	0.023	30

Table 15: Minimum Connected rooted  $k$ -hop Dominating Set Results on the grid graphs using ASP

At very last we want to have a deeper look into one particular aspect. During the solution process upper and lower bounds are determined. Most of the time the ILP-version is capable of finding a solid upper bound quickly. The vast majority of the time needed to find an optimal solution is spent on closing the gap to the lower bound. To illustrate this the next table shows after what time an upper bound that is 20%, 10%, 5% and 0% different from an optimal solution is found. In the cases where the ASP-version performs better it also finds a proper upper bound faster. In the one case where the ILP-version performs better it finds an appropriate upper bound faster.

name	type	k	20%	10%	5%	0%	time to close the gap	# lazily added con- straints	runtime(s)	optimal
middle-leaf	ILP	1	0s	0s	0s	0s	1099s	4945	1099.324462	[22,21]
middle-leaf	ASP	1	0s	0s	0s	0s	154s	-	153.605	22
bigger-leaf	ILP	1	1s	4s	4s	14s	1044s	6726	1058.414758	[25, 22]
bigger-leaf	ASP	1	0s	2s	5s	5s	997s	-	1002.022	[25, 24]
GNM_250_6225	ILP	1	0s	0s	6s	6s	894s	0	900.64	10
GNM_250_6225	ASP	1	238s	-	-	-	-	-	-	10
GRID_8_8	ILP	1	0s	2s	5s	599s	175s	6451	774.59	26
GRID_8_8	ASP	1	0s	0s	0s	11s	81s	-	92.739	26

Table 16: Time that is necessary to find appropriate upper bounds

## 7 Discussion

As already mentioned and as Hyunh [10] stated our model has some shortcomings and disregards aspects that influence an optimal venation pattern in real plants. We only focus on minimizing the number of cells that have to be transformed into vein cells, under the condition that the entire leaf can still be supplied with water and nutrients. Doing so the number of photo synthetic active cells and their outcome should be maximized. Our model completely disregards the vein hierarchy and among other things that environmental circumstances also influence the venation pattern [16]. The fact that plants try to minimize their total branch length and the transport distance for nutrients [4] is also disregarded.

As our results revealed/ showed the neither the ILP implementation nor the ASP implementation are capable of generating solutions for our leaf graphs in a reasonable amount of time. The ILP implementation is incapable of finding an optimal solution in under 1000 seconds for the instance *middle-leaf*, having only 62 nodes, with parameter  $k = 1$ . The ASP implementation on the other hand needed only 154 seconds to find an optimal solution. However both version find an appropriate upper bound in less than 1 second. The rest of the solving time is entirely used to close the gap from the lower bound. The instance *GNM\_500\_62375* on the contrary has 500 nodes but the ILP implementation nevertheless finds a solution in 154 seconds, whereas the ASP version could not find an optimal solution after 1000 seconds. As the results show the same difference in runtime on other rather sparse and large random graphs the ILP version seems to perform better on random graphs in general. As the results for the random graphs indicated our ILP implementation might be a reasonable approach applied to other problems which can be modeled with the *Minimum Connected (rooted) k-hop Dominating Set* depending on the structure of the input instances.

As well as Hyunh [10] made the observation for the ASP implementation that an increasing parameter  $k$  reduces the runtime significantly our tests showed the same effect using the ILP implementation. For the random graphs and parameter  $k = 2$  or  $k = 3$  every instance could be solved in less than 1 second. It should also be noted that for most of the instances in this case only a few or even none constraints needed to be added lazily. Optimal solutions consisted in this case for the most instances only of the single root node or contained also a few additional nodes. These results can not unconditionally applied to other real world problems as their graphs can have specific structures that differ from random graphs. Also on our leaf graphs an increasing  $k$  implied a better runtime. However in the case of  $k = 2$  and  $k = 3$  the instances *maple* and *asymmetric* could not be solved under 1000 seconds. We can not simply arbitrarily increase the parameter  $k$  in our model as vein cells must be in a range of 2-3 cells from mesophyll cells [13, p. 469]. The runtime of the grid graphs also went down with increased  $k$ . For this graphs even with  $k = 1$  an optimal solution could be found in under 1000 seconds. Admittedly all instances only had 64 nodes. As for the instance *GRID\_8\_8* the time to find an optimal solution was 775 seconds it can be assumed that for larger instances the runtime exceeds 1000 seconds.

Using the *intermediate node constraints* reduced the runtime the most. However in the most cases this constraints added unnecessary nodes to a solution which are not included without using this constraint. Nonetheless it could be considered to use this method to

create approximative solutions. But for this purpose it would be desirable to formally prove the maximal amount of extra nodes in relation to an optimal solution. However our results show, at least exemplarily, that in most cases even without this additional constraint in rather short time appropriate upper bounds were established. For the instance *middle-leaf* for example the ILP implementation as well as the ASP implementation found an upper bound in less than 1 second that does not differ from an optimal solution. Thus an approximation for the upper bound does not seem to be necessary. In fact a heuristic that generates an appropriate lower bound is much more desirable as closing the gap to the upper bound takes the major amount of time. Even for the rather large instance *maple* an upper bound that does not differ from the optimal solution using the *intermediate node constraint* is found after 29 seconds. At best this constraint could be used to evaluate how good upper bounds from the solving process are. But for this purpose an approximation factor would be necessary. For the *asymmetric* an optimal solution could not be found under 1000 seconds even using this constraint. According to this there is still need for optimization to create a satisfying implementation even if this constraint is used.

According to the current information using vertex separators seem to be the best method to induce connectivity on graph theoretical problems. Alternative approaches from [6] or [11] were not as successful for the corresponding problems in comparison to formulations that use vertex separators. Especially for the steiner tree problem Fischetti et al. [8] could achieve good results compared to other approaches. Also Bomersbach et al. [1] could achieve good results for the Connected Maximum Coverage Problem. In [3] and [2] this method was evaluated as promising. For our problem and especially for the graphs that represent our leafs this method was not satisfying. The same applies to quadratical grid graphs. We assume the high number of unconnected integer solutions that are generated in the iteration process as being crucial. These solutions are most likely in some manner symmetrical such that an appropriate symmetry breaker could reduce the runtime drastically.

In general the ASP implementation performed better on our graphs representing the leafs. Hyunh [10] mentioned different aspects in the conclusion of her thesis how the ASP implementation can be improved. As this implementation performed better than the ILP implementation so far it might be more reasonable to improve the ASP implementation rather than the ILP.

Another aspect that our tests revealed is that especially on such instance where there is a rather large gap between the size of an optimal unconnected solution and an optimal connected solution the runtime is relatively high. This is probably related to the fact that in such cases many constraints were added lazily, which indicates that there is a high amount of unconnected integer solutions. For the instances where the gap was rather tight the runtime was much better. In the tests from [10] an ILP implementation for the unconnected Minimum  $k$ -hop Dominating Set could create solutions much faster than the ASP implementation. This specific superiority is reflected here such that quickly valid solutions could be generated and it only needs to be verified if the solution is connected and otherwise only a few constraints needed to be added.

The density has also shown as a parameter which highly influences the runtime. On sparse graphs both the ILP implementation and the ASP implementation performed rather bad. For the random graphs instances with 250 and 500 nodes could not be solved

under 1000 seconds on rather sparse graphs with parameter  $k = 1$ . Our leaf graphs are all very sparse such that this effect plays a role as well. With increasing size the density of our graphs even decreases.

Preadding vertex separator constraints had an measurable influence on the runtime. Unfortunately this effect alone could not improve the runtime in a manner that a satisfying implementation for our model could be created. Despite the fact that many constraints were preadded there were still a lot constraints that were added in the iteration process. It could make sense to identify the types of constraints that are still added in the solution process to prevent unnecessary iterations when they are added beforehand. This might lead to a better runtime.

Another approach to improve the implementation can be to add violated constraints not only after integer solutions are created but already when LP relaxations are calculated. This approach was used in [3] and lead to sufficient LP bounds. Eine weitere Möglichkeit, das Verfahren zu optimieren, wäre es, constraints nicht nur dann hinzuzufügen, wenn eine ganzzahlige Lösung ermittelt wurde, sondern schon dann, wenn eine LP relaxierung ermittelt wird. Dieser Ansatz wurde auch in [3] verfolgt. Dabei konnten sehr gute Erfolge hinsichtlich der Lp Bound erzielt werden.

Recently a paper was published that compared different ILP formulations for the MWCSP [15]. Rehfeldt et al. [15] compare theoretically and empirically an edge based ILP formulation called Extended Steiner Arborescence Formulation (ESA) with the ILP formulation from [8]. In this paper it is proven that the polyhedron of the ESA is a real subset of the node based formulation from [8]. The computational results show that the ESA outperforms the node based one as the runtime was shorter for most instances. Also the with the ESA it was possible to solve previous unsolved instances. It is possible to create an ILP formulation for our model which uses the connectivity inducing constraints of the ESA. The implementation of the ESA is embedded in the upcoming version of *SCIP-Jack*, a C based branch-and-cut framework for the Steiner Tree problem. The ESA also needs exponentially many constraints to induce connectivity. As the efficiency and the runtime of a branch-and-cut approach depends on concrete implementation details and used heuristics, it would be necessary to explore the source code and the documentation. There are also several publications that can be found on the official SCIP webpage <https://www.scipopt.org/> that can be helpful. It could also be reasonable to combine both approaches, such that a minimum  $k$ -hop dominating set  $D_t$  is found at first and afterwards a minimum weight connected steiner tree  $D$  with  $D_t$  as set of terminals is found. This method could benefit from the facts that our ILP formulation can find unconnected minimum  $k$ -hop dominating sets rather quickly and MWCST instances can be solved very quick using ESA. However this might lead to not necessarily optimal solutions.

## 8 Conclusion

Given the fact that we adopted the model from [10] and only implemented it in another framework the models shortcomings are still present. It disregards different aspects that play a role in the venation pattern for real plants.

Additionally our implementation, in its current version, is not capable of generating optimal solutions in a reasonable amount of time for the leaf representing graphs. The ASP implementation performs better on these graphs and therefore is the better choice for to implement the model. Even after different approaches to reduce the runtime were evaluated the ASP implementation performed better. Nevertheless there are still approaches that can be evaluated.

The next step for the ILP implementation should either be to adapt the edge based ILP formulation ESA and aspects of its implementation from the current SCIP-Jack software, or to improve the formulation of this thesis. It probably can be improved by inventing a symmetry breaker that reduces the number of symmetrical unconnected integer solutions which are determined in the solving process. Additionally it should be evaluated which type of constraints can be further preadded that would otherwise be added anyway in the process. Another important point is to find heuristics that allow to determine sufficient lower bounds faster.

Though it is also reasonable to implement the suggestions from Hyunh [10] to further improve the ASP implementation as it outperformed the ILP implementation.



## A Appendix

### Full Tables

#### ILP

name	k	# lazily added constraints	runtime(s)	optimal
GNM_ 50_ 122	1	66	0.034878	11
GNM_ 50_ 245	1	9	0.07	7
GNM_ 50_ 368	1	0	0.013882	5
GNM_ 50_ 490	1	4	0.016478	4
GNM_ 50_ 612	1	0	0.017783	4
GNM_ 50_ 735	1	3	0.018471	3
GNM_ 50_ 858	1	3	0.038161	3
GNM_ 50_ 980	1	3	0.023549	3
GNM_ 50_ 1102	1	3	0.019566	3
GNM_ 50_ 1225	1	0	0.002396	1
GNM_ 100_ 495	1	113	0.376731	14
GNM_ 100_ 990	1	17	0.488522	8
GNM_ 100_ 1485	1	7	0.396982	6
GNM_ 100_ 1980	1	0	0.315584	5
GNM_ 100_ 2475	1	0	0.045136	4
GNM_ 100_ 2970	1	0	0.013737	3
GNM_ 100_ 3465	1	0	0.010702	3
GNM_ 100_ 3960	1	0	0.007955	2
GNM_ 100_ 4455	1	0	0.00505	2
GNM_ 100_ 4950	1	0	0.00535	1
GNM_ 250_ 3112	1	0	1017.303471	[17;15]
GNM_ 250_ 6225	1	0	900.64	10
GNM_ 250_ 9338	1	0	29.67	7
GNM_ 250_ 12450	1	0	46.78	6
GNM_ 250_ 15562	1	0	12.29	5
GNM_ 250_ 18675	1	0	0.97	4
GNM_ 250_ 21788	1	3	0.415836	3
GNM_ 250_ 24900	1	0	0.040482	3
GNM_ 250_ 28012	1	0	0.024473	2
GNM_ 250_ 31125	1	0	0.017227	1
GNM_ 500_ 12475	1	42	1004.920676	[21;13]
GNM_ 500_ 24950	1	0	1051.277153	[12;8]
GNM_ 500_ 37425	1	0	9.89	4
GNM_ 500_ 49900	1	0	1017.23594	[6;5]
GNM_ 500_ 62375	1	0	178.495614	5
GNM_ 500_ 74850	1	0	9.753998	4
GNM_ 500_ 87325	1	0	21.368156	4
GNM_ 500_ 99800	1	0	0.286309	3
GNM_ 500_ 112275	1	0	0.189313	2
GNM_ 500_ 124750	1	0	0.11	1

Table 17: Minimum Connected rooted 1-hop Dominating Set Results on the random graphs

name	k	# lazily added constraints	optimal	runtime(s)
GNM_50_122	2	67	11	0.03795
GNM_50_245	2	9	7	0.066219
GNM_50_368	2	0	1	0.008017
GNM_50_490	2	0	1	0.002605
GNM_50_612	2	0	1	0.002223
GNM_50_735	2	0	1	0.002411
GNM_50_858	2	0	1	0.002486
GNM_50_980	2	0	1	0.002173
GNM_50_1102	2	0	1	0.012025
GNM_50_1225	2	0	1	0.001756
GNM_100_495	2	6	4	0.108993
GNM_100_990	2	12	2	0.060489
GNM_100_1485	2	0	1	0.022559
GNM_100_1980	2	0	1	0.004219
GNM_100_2475	2	0	1	0.004791
GNM_100_2970	2	0	1	0.044863
GNM_100_3465	2	0	1	0.004259
GNM_100_3960	2	0	1	0.004273
GNM_100_4455	2	0	1	0.003927
GNM_100_4950	2	0	1	0.003468
GNM_250_3112	2	0	2	0.270981
GNM_250_6225	2	28	1	0.101028
GNM_250_9338	2	0	1	0.17136
GNM_250_12450	2	0	1	0.031756
GNM_250_15562	2	109	1	0.257635
GNM_250_18675	2	0	1	0.035879
GNM_250_21788	2	0	1	0.030358
GNM_250_24900	2	0	1	0.024402
GNM_250_28012	2	0	1	0.018999
GNM_250_31125	2	0	1	0.016561
GNM_500_12475	2	0	2	1.123904
GNM_500_24950	2	0	1	0.663096
GNM_500_37425	2	0	1	0.228299
GNM_500_49900	2	0	1	0.272308
GNM_500_62375	2	0	1	0.29011
GNM_500_74850	2	0	1	0.249534
GNM_500_87325	2	0	1	0.250321
GNM_500_99800	2	0	1	0.170296
GNM_500_112275	2	0	1	0.148031
GNM_500_124750	2	0	1	0.119448

Table 18: Minimum Connected rooted 2-hop Dominating Set Results on the random graphs

name	k	# lazily added constraints	optimal	runtime(s)
GNM_50_122	3	0	2	0.01651
GNM_50_245	3	0	1	0.005787
GNM_50_368	3	0	1	0.007788
GNM_50_490	3	0	1	0.002089
GNM_50_612	3	0	1	0.002541
GNM_50_735	3	0	1	0.00202
GNM_50_858	3	0	1	0.001855
GNM_50_980	3	0	1	0.00213
GNM_50_1102	3	0	1	0.012196
GNM_50_1225	3	0	1	0.001661
GNM_100_495	3	0	1	0.026969
GNM_100_990	3	0	1	0.022669
GNM_100_1485	3	0	1	0.022822
GNM_100_1980	3	0	1	0.004204
GNM_100_2475	3	0	1	0.006448
GNM_100_2970	3	0	1	0.044946
GNM_100_3465	3	0	1	0.004356
GNM_100_3960	3	0	1	0.004163
GNM_100_4455	3	0	1	0.004094
GNM_100_4950	3	0	1	0.003533
GNM_250_3112	3	14	1	0.141794
GNM_250_6225	3	28	1	0.106819
GNM_250_9338	3	51	1	0.205765
GNM_250_12450	3	82	1	0.03714
GNM_250_15562	3	109	1	0.267159
GNM_250_18675	3	0	1	0.036207
GNM_250_21788	3	0	1	0.042911
GNM_250_24900	3	0	1	0.038669
GNM_250_28012	3	0	1	0.023179
GNM_250_31125	3	0	1	0.020695
GNM_500_12475	3	0	1	0.634489
GNM_500_24950	3	68	1	0.947696
GNM_500_37425	3	118	1	0.288719
GNM_500_49900	3	0	1	0.405276
GNM_500_62375	3	0	1	0.544754
GNM_500_74850	3	0	1	0.265611
GNM_500_87325	3	0	1	0.270045
GNM_500_99800	3	0	1	0.404701
GNM_500_112275	3	0	1	0.205316
GNM_500_124750	3	0	1	0.225787

Table 19: Minimum Connected rooted 3-hop Dominating Set Results on the random graphs

## ASP

name	k	runtime(s)	optimal
GNM_50_122	1	0.014	11
GNM_50_245	1	0.033	7
GNM_50_368	1	0.031	5
GNM_50_490	1	0.050	4
GNM_50_612	1	0.055	4
GNM_50_735	1	0.044	3
GNM_50_858	1	0.050	3
GNM_50_980	1	0.059	2
GNM_50_1102	1	0.052	3
GNM_50_1225	1	0.055	1
GNM_100_495	1	32.451	14
GNM_100_990	1	278.296	8
GNM_100_1485	1	42.545	6
GNM_100_1980	1	4.049	6
GNM_100_2475	1	0.655	4
GNM_100_2970	1	0.226	3
GNM_100_3465	1	0.208	3
GNM_100_3960	1	0.234	2
GNM_100_4455	1	0.253	2
GNM_100_4950	1	0.246	1
GNM_250_3112	1	1017.204	[23;9]
GNM_250_6225	1	1009.124	[12;6]
GNM_250_9338	1	1009.402	[8;5]
GNM_250_12450	1	1013.976	[6;4]
GNM_250_15562	1	1008.099	[5;4]
GNM_250_18675	1	25.687	4
GNM_250_21788	1	1.749	3
GNM_250_24900	1	1.830	3
GNM_250_28012	1	3.400	2
GNM_250_31125	1	1.651	1
GNM_500_12475	1	1016.396	[29;7]
GNM_500_24950	1	1011.967	[15;4]
GNM_500_37425	1	1010.582	[10;4]
GNM_500_49900	1	1007.821	[7;4]
GNM_500_62375	1	1006.141	[6;4]
GNM_500_74850	1	597.053	4
GNM_500_87325	1	621.053	4
GNM_500_99800	1	13.348	3
GNM_500_112275	1	8.705	2
GNM_500_124750	1	8.058	1

Table 20: Minimum Connected rooted 1-hop Dominating Set Results on the random graphs using ASP

name	k	optimal	runtime(s)
GNM_50_122	2	5	0.025
GNM_50_245	2	1	0.030
GNM_50_368	2	1	0.036
GNM_50_490	2	1	0.036
GNM_50_612	2	1	0.038
GNM_50_735	2	1	0.046
GNM_50_858	2	1	0.047
GNM_50_980	2	1	0.049
GNM_50_1102	2	1	0.052
GNM_50_1225	2	1	0.048
GNM_100_495	2	4	0.084
GNM_100_990	2	2	0.098
GNM_100_1485	2	1	0.111
GNM_100_1980	2	1	0.143
GNM_100_2475	2	1	0.151
GNM_100_2970	2	1	0.174
GNM_100_3465	2	1	0.188
GNM_100_3960	2	1	0.206
GNM_100_4455	2	1	0.220
GNM_100_4950	2	1	0.213
GNM_250_3112	2	2	0.521
GNM_250_6225	2	1	0.652
GNM_250_9338	2	1	0.737
GNM_250_12450	2	1	0.867
GNM_250_15562	2	1	0.972
GNM_250_18675	2	1	1.141
GNM_250_21788	2	1	1.221
GNM_250_24900	2	1	1.305
GNM_250_28012	2	1	1.453
GNM_250_31125	2	1	1.519
GNM_500_12475	2	2	2.314
GNM_500_24950	2	1	2.770
GNM_500_37425	2	1	3.236
GNM_500_49900	2	1	3.702
GNM_500_62375	2	1	4.218
GNM_500_74850	2	1	4.799
GNM_500_87325	2	1	5.456
GNM_500_99800	2	1	6.199
GNM_500_112275	2	1	6.268
GNM_500_124750	2	1	6.522

Table 21: Minimum Connected rooted 2-hop Dominating Set Results on the random graphs using ASP

name	k	optimal	runtime(s)
GNM_50_122	3	2	0.022
GNM_50_245	3	1	0.029
GNM_50_368	3	1	0.032
GNM_50_490	3	1	0.039
GNM_50_612	3	1	0.041
GNM_50_735	3	1	0.040
GNM_50_858	3	1	0.041
GNM_50_980	3	1	0.048
GNM_50_1102	3	1	0.051
GNM_50_1225	3	1	0.053
GNM_100_495	3	1	0.082
GNM_100_990	3	1	0.101s
GNM_100_1485	3	1	0.119
GNM_100_1980	3	1	0.140
GNM_100_2475	3	1	0.163
GNM_100_2970	3	1	0.172
GNM_100_3465	3	1	0.186
GNM_100_3960	3	1	0.214
GNM_100_4455	3	1	0.227
GNM_100_4950	3	1	0.223
GNM_250_3112	3	1	0.529
GNM_250_6225	3	1	0.657
GNM_250_9338	3	1	0.782
GNM_250_12450	3	1	0.885
GNM_250_15562	3	1	0.967
GNM_250_18675	3	1	1.114
GNM_250_21788	3	1	1.263
GNM_250_24900	3	1	1.323
GNM_250_28012	3	1	1.489
GNM_250_31125	3	1	1.510
GNM_500_12475	3	1	2.297
GNM_500_24950	3	1	2.714
GNM_500_37425	3	1	3.250
GNM_500_49900	3	1	3.719
GNM_500_62375	3	1	4.513
GNM_500_74850	3	1	4.786
GNM_500_87325	3	1	5.305
GNM_500_99800	3	1	5.845
GNM_500_112275	3	1	6.490
GNM_500_124750	3	1	6.802

Table 22: Minimum Connected rooted 3-hop Dominating Set Results on the random graphs using ASP

## References

- [1] A. Bomersbach, M. Chiarandini, and F. Vandin. “An Efficient Branch and Cut Algorithm to Find Frequently Mutated Subnetworks in Cancer”. In: *Algorithms in Bioinformatics*. Ed. by M. Frith and C. Nørgaard Storm P. Springer International Publishing, 2016, pp. 27–39.
- [2] A. Buchanan, J. Sung, S. Butenko, and E. Pasiliao. “An Integer Programming Approach for Fault-Tolerant Connected Dominating Sets”. In: *INFORMS Journal on Computing* 27 (Feb. 2015), pp. 178–188.
- [3] R. Carvajal, M. Constantino, M. Goycoolea, J. Vielma, and A. Weintraub. “Imposing Connectivity Constraints in Forest Planning Models”. In: *Operations Research* 61 (Aug. 2013), pp. 824–836.
- [4] A. Conn, U. Pedmale, J. Chory, and S. Navlakha. “High-Resolution Laser Scanning Reveals Plant Architectures that Reflect Universal Network Design Principles”. In: *Cell Systems* 5 (July 2017), 53–62.e3.
- [5] M. Desrochers and G. Laporte. “Improvements and Extensions to the Miller-Tucker-Zemlin Subtour Elimination Constraints”. In: *Oper. Res. Lett.* 10.1 (Feb. 1991), pp. 27–36.
- [6] N. Fan and J.-P. Watson. “Solving the Connected Dominating Set Problem and Power Dominating Set Problem by Integer Programming”. In: *Combinatorial Optimization and Applications*. Ed. by G. Lin. Springer Berlin Heidelberg, 2012, pp. 371–383.
- [7] M. Fischetti. *Introduction to Mathematical Optimization*. Independently Published, 2019.
- [8] M. Fischetti, M. Leitner, I. Ljubic, M. Luipersbeck, M. Monaci, M. Resch, D. Salvagnin, and M. Sinnl. “Thinning out Steiner trees: a node based model for uniform edge costs”. English. In: *Mathematical Programming Computation* 9.2 (2017), pp. 203–229.
- [9] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman and Co., 1990.
- [10] M. K. Hyunh. *Solving Dominating Set Using Answer Set Programming*. Feb. 2020.
- [11] M. El-Kebir and G. W. Klau. *Solving the Maximum-Weight Connected Subgraph Problem to Optimality*. 2014. arXiv: [1409.5308 \[cs.DS\]](#).
- [12] C. E. Miller, A. W. Tucker, and R. A. Zemlin. “Integer Programming Formulation of Traveling Salesman Problems”. In: *J. ACM* 7.4 (Oct. 1960), pp. 326–329.
- [13] P. S. Nobel. *Physicochemical and Environmental Plant Physiology*. 4. Elsevier, 2009.
- [14] J. Posada, R. Sievänen, C. Messier, J. Perttunen, E. Nikinmaa, and M. Lechowicz. “Contributions of leaf photosynthetic capacity, leaf angle and self-shading to the maximization of net photosynthesis in *Acer saccharum*: a modelling assessment”. In: *Annals of botany* 110 (June 2012), pp. 731–41.
- [15] D. Rehfeldt, H. Franz, and T. Koch. *Optimal Connected Subgraphs: Formulations and Algorithms*. eng. Tech. rep. 20-23. ZIB, 2020.

- [16] L. Sack and C. Scoffoni. “Leaf venation: Structure, function, development, evolution, ecology and applications in the past, present and future”. In: *The New phytologist* 198 (Apr. 2013).
- [17] Y. Wang, A. Buchanan, and S. Butenko. “On imposing connectivity constraints in integer programs”. In: *Mathematical Programming* (Feb. 2017).

## List of Figures

1	Illustration of vertex separators . . . . .	6
2	. . . . .	9
3	Illustration of the principle. The dashed circle outlines the dominating set. All vertices, that are connected to $n + 1$ are not part of the dominating set.	11
4	The dashed circle outlines the necessity to have connected triplets at the end of a branch . . . . .	13
5	An unconnected solution where the path length constraint is satisfied. . .	14
6	The leaf graphs . . . . .	16

## List of Tables

1	The characteristics of the leaf graphs . . . . .	17
2	Minimum Connected rooted $k$ -hop Dominating Set Results on the leaf graphs . . . . .	17
3	The characteristics of the random graphs . . . . .	19
4	Minimum Connected rooted $k$ -hop Dominating Set Results on the random graphs . . . . .	20
5	The characteristics of the grid graphs . . . . .	21
6	Minimum Connected rooted $k$ -hop Dominating Set Results on the grid graphs . . . . .	21
7	Minimum Connected rooted $k$ -hop Dominating Set Results using the MTZ constraints . . . . .	22
8	Minimum Connected rooted $k$ -hop Dominating Set Results with preadded vertex separator constraints . . . . .	22
9	Minimum Connected rooted $k$ -hop Dominating Set Results with IMN constraint . . . . .	23
10	Minimum Connected rooted $k$ -hop Dominating Set Results with SPL constraint . . . . .	23
11	Minimum Connected rooted $k$ -hop Dominating Set Results with GAUS constraint . . . . .	23
12	Minimum Connected rooted $k$ -hop Dominating Set Results with SPL and GAUS constraint . . . . .	24
13	Minimum Connected rooted $k$ -hop Dominating Set Results on the leaf graphs using ASP . . . . .	24
14	Minimum Connected rooted $k$ -hop Dominating Set Results on the random graphs using ASP . . . . .	25

15	Minimum Connected rooted $k$ -hop Dominating Set Results on the grid graphs using ASP . . . . .	26
16	Time that is necessary to find appropriate upper bounds . . . . .	26
17	Minimum Connected rooted 1-hop Dominating Set Results on the random graphs . . . . .	32
18	Minimum Connected rooted 2-hop Dominating Set Results on the random graphs . . . . .	33
19	Minimum Connected rooted 3-hop Dominating Set Results on the random graphs . . . . .	34
20	Minimum Connected rooted 1-hop Dominating Set Results on the random graphs using ASP . . . . .	35
21	Minimum Connected rooted 2-hop Dominating Set Results on the random graphs using ASP . . . . .	36
22	Minimum Connected rooted 3-hop Dominating Set Results on the random graphs using ASP . . . . .	37