Solving Connected Dominating Set Variants Using Constraint Programming

Tim Pfalz

A thesis presented for the degree of
Bachelor of Science

Algorithmic Bioinformatics
Heinrich Heine University Düsseldorf
Germany
1st March 2021

Abstract

In this thesis, we try to find optimal venation patterns of leaves so that the photosynthetic output can be maximized. One of the venation patterns objectives is to build a transporting system, which acts as a nutrient provider for photosynthesis producing cells and as a transporter of sugars created by photosynthesis. Therefore, we have to minimize the number of vein cells and maximize the number of photosynthesis producing non-vein cells to get a maximized photosynthetic output. If we ignore other venation pattern influencing aspects, we can represent the venation pattern through variations of the Connected Dominating Set problem. We solved this using Constraint Programming and compared it to solutions using Answer Set Programming and Integer Linear Programming. In comparison, ILP and ASP were able to solve the Dominating Set variants much faster.

Contents

1 Introduction

Plants try to optimize their architecture to fulfil several objectives, e.g., maximizing the photosynthetic output. The overall plant architecture tends to have a nearly Pareto optimal trade-off between minimizing cost and maximizing performance (efficiency), by minimizing total branch length and nutrient transport distances. This gained them a selective fitness advantage over other architectures [6]. The premise of this work is, that the leaf architecture follows the overall plant's architectures Pareto optimality between maximizing efficiency and minimizing cost (investment), by minimizing total vein length [11]. The goal of this thesis is to use Constraint Programming (CP) to find optimal leaf venation patterns and compare the results and runtime to the solutions of ILP [14] and ASP [13]. This work focuses on minimizing the number of connected venation cells and maximizing the number of mesophyll cells in a leaf and thereby increase photosynthesis. Other factors influencing the architecture, e.g., temperature, are not taken into account.

There are three main tissues found in plant leaves: the mesophyll, the epidermis and the vascular tissue. Each one of those is composed of layers of cells. The leaf vascular is located on the inside of the mesophyll, which is surrounded by the epidermis tissue [7, p.325]. Leaf veins consist of vascular tissue, which itself has two main tissues: the xylem which distributes water and other nutrients upward and the phloem which transports sugars downward through the plant [8, p.211-217]. The mesophyll cells, where most of the photosynthesis occurs, get provided with nutrients from the vascular system and use the vascular system for transporting sugars created by photosynthesis [9, p.166-167]. That is why they need to be near the leaf veins [10, 6, p.469].

Mathematically, the venation of a leaf can be represented through Dominating Set variants. Therefore we have to represent the leaf as a graph with leaf cells as vertices and plasmodesmata connections between the leaf cells as edges. A dominating set is a subset of vertices so that every vertex not in this subset is adjacent to at least one vertex in it [15, p.1]. As we try to minimize the number of venation cells, we also need to minimize the number of vertices part of the dominating set and due to the vein cell's need of being connected, a Minimum Connected Dominating Set (Min-CDS) variant would fit well. The Min-CDS is like the Dominating Set problem (DS) NP-hard [15, p.3].

The next section deals with MiniZinc, as well as defining constraint problem specifications. Then, in Section 3, we discuss which Dominating Set variants are suitable to represent a leaf's venation pattern and how to implement them. The last two sections contain the results on several undirected graphs, with a discussion about differences to the ASP's and ILP's results and conclusions.

2 Preliminaries

Described in the next two subsections are first the basic concepts of Constraint Programming, to understand the implementations in the methods chapter and secondly, an explanation of how MiniZinc, a language for specifying constraint optimization and decision problems, works.

2.1 Constraint Programming

Constraint Programming is used to solve combinatorial search problems and is based on logic programming. A constraint problem consists of variables, which are associated with domains and constraints. Domains are sets of values, that a given variable can take and constraints are predicate logical formulas [1], that define limitations on values that variables or combinations of variables can be assigned to. If a constraint problem has the task to find one or all solutions, whether a solution exists or a partial instantiation can be extended to a full solution, then it is called a constraint satisfaction problem (CSP). Adding an objective function turns a CSP into a constraint optimization problem (COP) [2]. A solution for a constraint problem does not extend the limitations, set by the domain of each variable. It also satisfies the associated constraints. If a solution does not exist, the problem is unsatisfiable [3].

2.2 MiniZinc

MiniZinc is a medium-level declarative modelling language for modelling constraint satisfaction or optimization problems. It comes with several predefined constraints, but also lets users formulate their own constraints and functions with the existential and universal quantifiers, sums over index sets and logical connectives. MiniZinc transforms a MiniZinc model with a belonging data model to a FlatZinc (a low-level solver input language) model, which consist of variable declarations and constraint definitions. Additionally, if faced with an optimization problem, the FlatZinc model contains an objective function. There is no standard modelling language for CP problems and usually, every solver has its own. However, because a solver only has to provide a FlatZinc front-end, MiniZinc is very appealing for developers that plan on using or comparing several solvers [5]. What also simplifies interfacing with MiniZinc for solvers is that, while translating a MiniZinc to a FlatZinc model, MiniZinc only produces the types of variables and constraints that are supported by the solver [4]. MiniZinc does not come with a default solver, instead, it relies on external solvers solely. The mainly utilized solver used in this thesis was the CP-SAT solver from Google OR-Tools. OR-Tools is software for solving combinatorial optimization problems that include solvers for constraint programming, linear and mixed-integer programming, vehicle routing and graph algorithms. Other CP solvers worth mentioning are Chuffed and Gecode.

3 Methods

In this chapter we describe the Dominating Set variants and how to implement and call the models of those in MiniZinc. We want to get all solutions, where the number of vertices part of the dominating set for each variant is minimal.
We represent the leaf as undirected graph $G = (V, E)$ with leaf cells as vertices V and plasmodesmata connections between the leaf cells as edges E. Because mesophyll cells depend on the transport functions of the vein cells, we can represent the venation as dominating set D. This is a variable Boolean array of length $|V|$, From, To and DE are arrays of length $|E|$, with From and To being integer arrays that hold the incoming or outgoing node for each edge and DE being a variable boolean array that represents the edges of the dominating set. The root node of V is called r.

Definition 1 (Dominating Set):

A dominating set is a subset of vertices $D \subseteq V$ so that every vertex $v \in V \setminus D$ is adjacent to at least one vertex in D.

$\forall v \in V \setminus D : \exists w \in D : (v, w) \in E$

One possible formulation of the equivalent Dominating Set constraint in MiniZinc is shown in Algorithm 1:

Algorithm 1 Dominating Set Constraint

```
1: for $v \in V$ with $D[v] ==$ false do
2:     count($e \in E : (v == \text{From}[e] \lor v == \text{To}[e]) \land (D[\text{To}[e]] ==$ True $\lor$
       $D[\text{From}[e]] ==$ True$)) > 0$
3: end for
```

This formulation constraints every node $v \in N$ to have at least one edge $e : \exists w \in V : (v, w) \in E$ where $w \lor v$ are part of the dominating set D.

Definition 2 (Rooted Connected Dominating Set):

Vein cells need to be connected to all other vein cells and to the rest of the plant to build a working transport system. In consequence, all nodes of the dominating set also need to be connected and one node has to be a root node that is connected to the petiole (otherwise the transport system can not transport nutrients up and down the plant). A rooted connected dominating set is a dominating set D, where $\forall v, w \in D : \exists$ path from v to w and the root vertex $r \in V$ is part of the dominating set D. The constraints of Algorithm 2 and Algorithm 3 could be added to a model, to ensure connectivity and rootedness:

Algorithm 2 Rooted Constraint

```
1: $\exists v \in D : v == r$
```

If this constraint is added to a Dominating Set model, it constrains the root node r to also be a node of the dominating set $D[r] ==$ `True`.

Algorithm 3 Connected and DE Constraint

```
1: for $e \in E$ with From$[e] \in D \wedge$ To$[e] \in D$ do
2:     $DE[e] ==$ True
3: end for
4: connected (From, To, $D$, $DE$)
```

For creating the Minimum Connected Dominating Set model, we implement a constraint that gives every edge $e \in DE$ a boolean value, depending on whether the two nodes of each edge are part of the dominating set $(v, w) = e : D[v] ==$ `True` $\wedge\ D[w] ==$ `True`. Then we use the graph constraint ‘connected’ from MiniZinc to ensure that the subgraph $G' \subseteq G : G' = (D, DE)$ is connected. The Minimum Rooted Connected Dominating Set model shown in Algorithm 4, makes use of MiniZinc’s global predicate ‘reachable.mzn’, instead of combining the rooted and connected constraints.

Algorithm 4 Reachable and DE Constraint

```
1: for $e \in E$ with From$[e] \in D \wedge$ To$[e] \in D$ do
2:     $DE[e] ==$ True
3: end for
4: reachable (From, To, $r$, $D$, $DE$)
```

The ‘reachable’ predicate takes a graph G, a root node r and the subgraph $S \subseteq G$ induced by the dominating set. Then it makes sure, that every node $v \in S$ is reachable from the root node.

Definition 3 (Canonical Connected Dominating Set Variants):

If we change the DE constraint and use the directed counterparts of 'connected'/'reachable' of our Connected Dominating Set variants like shown in Algorithm 5 or Algorithm 6, we can find solutions for otherwise too large input graphs.

Algorithm 5 Connected and DE Symmetry Breaking Constraint

```
1: for $e \in E$ with $\mathrm{From}[e] \in D \land \mathrm{To}[e] \in D \land \mathrm{From}[e] < \mathrm{To}[e]$ do
2:     $DE[e] ==$ True
3: end for
4: for $e_1 \in E$ with $DE[e_1] ==$ True do
5:     $\nexists e_2 \in E$ with $\mathrm{From}[e_2] == \mathrm{To}[e_1] \land DE[e_2] ==$ True $: \mathrm{From}[e_1] > \mathrm{To}[e_2]$
6: end for
7: dconnected $(\mathrm{From}, \mathrm{To}, D, DE)$
```

Algorithm 6 Rooted Connected and DE Symmetry Breaking Constraints

```
1: for $e \in E$ with $\mathrm{From}[e] \in D \land \mathrm{To}[e] \in D \land \mathrm{From}[e] < \mathrm{To}[e]$ do
2:     $DE[e] ==$ True
3: end for
4: for $e_1 \in E$ with $DE[e_1] ==$ True do
5:     $\nexists e_2 \in E$ with $\mathrm{From}[e_2] == \mathrm{To}[e_1] \land DE[e_2] ==$ True $: \mathrm{From}[e_1] > \mathrm{To}[e_2]$
6: end for
7: dreachable $(\mathrm{From}, \mathrm{To}, r, D, DE)$
```

Using 'dconnected'/'dreachable', the connected/reachable predicate for directed graphs, allows us to add a symmetry breaking constraint that ensures us that there can not be two edges $(v_1, w_1), (v_2, w_2) \in DE$ where $w_1 == v_2 \land w_2 < v_1$. By doing this we ignore all mirror-symmetric solutions and therefore get a result much faster.

Definition 4 (k-Hop Dominating Set Variants):

Because it suffices the mesophyll cells if a vein cell is not directly adjacent, but only a few cells away, the k-hop Dominating Set is even more suitable to represent the venation pattern. A k-hop dominating set is a dominating set, where every vertex $v \in V \setminus D$ has a path p to a vertex $w \in D$ with length l, $l \leq k$.

$\forall v \in V \setminus D : \exists w \in D : |p(v, w)| \leq k$

A k-hop constraint can be realized by preprocessing a k-transitive closure, as shown in Algorithm 7, on the input graph and then call a Dominating Set variant model with the modified graph. For every model that contains the connected constraint, an additional model is needed, where we also add the original graph G as a parameter.

Algorithm 7 k-Transitive Closure

```
Input: $G = (V, E)$, $E' = E$, $T = (V, E')$, $k$ = $k$-hop number
 1: for $v \in V$ do
 2:     currentNeighbours = neighbours of $v$
 3:     for $i$ from 0 to $k - 1$ do
 4:         for $n \in$ currentNeighbours do
 5:             newNeighbours = neighbours of $n$
 6:             for $w \in$ newNeighbours do
 7:                 if $v \neq w \wedge (v, w) \notin E'$ then
 8:                     add $(v, w)$ to $E'$
 9:                 end if
10:             end for
11:             currentNeighbours += newNeighbours
12:         end for
13:     end for
14: end for
```

At first we make a copy T of the input graph G, with $T = (V, E')$. Then we iterate through every neighbours neighbour w of a node v and add an edge (v, w) to E', when $(v, w) \notin E'$ and $v \neq w$. Then we append the neighbours neighbourhood to the neighbourhood of v. We repeat this k times for every node $v \in V$ and by doing so turn T into a k-transitive closure of G.

Definition 5 (Solving):

As MiniZinc only supports to return all solutions to a CSP, to solve a model we temporarily add the minimizing function: minimize $\sum D$, which transforms our CSP into a COP. Solving this gives us the first optimal solution D' for the model. Now to get all optimal solutions we use the first optimal solution to add a constraint to our CSP model: $\sum D == \sum D'$ and solve it by finding all solutions.

In summary, an optimal venation pattern is connected, rooted and can have a distance of a few cells to photosynthesis producing cells. Therefore our final method to solve the problem is the Minimum k-hop Rooted Connected Dominating Set.

4 Results

This section shows our results for the different Minimum Dominating Set variants. Further, it contains short descriptions of the characteristics of our test leaves. Subsequently, we discuss those results. More detailed descriptions of the test leaves can be found here [13, 14]. All test were performed using a notebook with an Intel (R) Core (TM) i7–7500U CPU @ 2.70GHz 2.90 Ghz and 16.0 GB of RAM under Windows 10.1909.

The leaf (a)*small-leaf* is the smallest instance and has 15 nodes. The leaves (b)*left-right-leaf* and (c)*middle-leaf* have both 62 nodes but differ in the numbering of those. Instances (d)*bigger-leaf* and (e)*rippedleaf* have 71 nodes, but while *bigger-leaf* follows the numbering structure of *left-right-leaf*, *rippedleaf* has edges missing, so that it is not connected. The two biggest graphs are (f)*maple* and (g)*asymmetric*. They have 118 and 378 nodes. *Maple* is also numbered from left to right and *asymmetric* is numbered completely different, because it has its root node close to the middle and not as the others on the bottom of the leaf.

The first column of our result tables contains the names of the test leaves. The columns containing the runtimes in seconds to find a single optimal solution and all optimal solutions, are called 'Time single' and 'Time all'. The column 'Threads' states the number of threads that have been used, 'Optimum' is the minimum number of nodes that are part of the dominating set for an optimal solution and 'Models' is the number of optimal solutions found. If there is no solution found in 20 minutes using at most 24 threads (4 for non-connected models), the 'Optimum' column contains the upper bound u and lower bound l found like this '$[l; u]$' or 'UNSAT' if the model is unsatisfiable. For CP, if a solution could not be found in under 20 minutes, we tested it again, by calling the MiniZinc model directly from the command line and printing out intermediate solutions. Those results are marked as CP* and show us the upper bounds. CP* contains the time in which the upper bound was found in 'Time single' and the overall running time in 'Time all'. The final column 'Solution in' states if the solution was found using ASP, ILP or CP.

Table 1 shows the results for the Minimum Dominating Set. We can see, that CP performs slower than ILP and faster than ASP on the *asymmetric* leaf. Besides that leaf, CP requires more time than ILP and ASP and can only find all optimal solutions for the smallest instance *small-leaf*.

	Solution in	Threads	Time single	Time all	Optimum	Models
(a) small-leaf	ASP	1	0.000	0.000	4	47
	ILP	1	0.0	–	4	–
	CP	1	0.359	1.015	4	47
(b) left-right-leaf	ASP	2	0.007	0.018	12	1091
	ILP	4	0.016	–	12	–
	CP	1	0.586	–	12	*Unknown*
(c) middle-leaf	ASP	2	0.007	0.024	12	1091
	ILP	4	0.016	–	12	–
	CP	1	0.531	–	12	*Unknown*
(d) bigger-leaf	ASP	2	0.007	0.010	13	43
	ILP	4	0.031	–	13	–
	CP	2	0.5	–	13	*Unknown*
(e) rippedleaf	ASP	2	0.007	0.011	14	441
	ILP	4	0.016	–	14	–
	CP	1	0.437	–	14	*Unknown*
(f) maple	ASP	2	0.031	0.016	20	1
	ILP	4	0.016	–	20	–
	CP	2	0.548	–	20	*Unknown*
(g) asymmetric	ASP	4	1252.886	–	$[59; 75]$	–
	ILP	4	17.009	–	62	–
	CP	3	599.573	–	62	*Unknown*

Table 1: Minimum Dominating Set.

There are minor differences for the Minimum 2-hop Dominating Set results, as can be seen in Table 2. For *small-leaf*, *left-right-leaf* and *middle-leaf* the number of optimal solutions decreased a lot. This causes ASP and CP to find all solutions faster. Contrary to this, *bigger-leaf* and *rippedleaf* increase their number of optimal solutions and therefore ASP requires more time to find them all. CP can find all optimal solutions for *bigger-leaf* on 1 thread faster than for the Minimum Dominating Set on 2 threads. ILP solves the problem for *rippedleaf* in the same time, but now using 1 instead of 4 threads. To find one optimal solution for the graphs required more time, except ILP on *middle-leaf* and *bigger-leaf*, ASP on *middle-leaf* and CP on *middle-leaf*. ASP, ILP and CP were able to find one optimal solution for *asymmetric* a lot faster.

	Solution in	Threads	Time single	Time all	Optimum	Models
(a) small-leaf	ASP	1	0.000	0.000	1	1
	ILP	1	0.0	–	1	–
	CP	1	0.375	0.75	1	1
(b) left-right-leaf	ASP	2	0.012	0.014	5	6
	ILP	4	0.062	–	5	–
	CP	1	0.672	2.937	5	6
(c) middle-leaf	ASP	2	0.014	0.013	5	6
	ILP	4	0.016	–	5	–
	CP	2	0.516	3.405	5	6
(d) bigger-leaf	ASP	2	0.015	0.021	6	1177
	ILP	4	0.016	–	6	–
	CP	1	0.562	146.902	6	1177
(e) rippedleaf	ASP	2	0.013	0.057	7	20500
	ILP	1	0.016	–	7	–
	CP	1	0.531	–	7	–
(f) maple	ASP	2	0.016	0.031	9	806
	ILP	4	0.031	–	9	–
	CP	2	0.818	–	9	*Unknown*
(g) asymmetric	ASP	4	264.726	1231.504	25	2020+
	ILP	4	0.577	–	25	–
	CP	2	26.495	–	25	*Unknown*

Table 2: Minimum 2-hop Dominating Set.

The results for the Minimum 3-hop Dominating Set are displayed in Table 3.

Compared with the Dominating Set variants from Table 1 and Table 2, CP finds all optimal solutions now faster and is only not able to find all optimal solutions for *asymmetric*. ASP's runtime increased slightly for finding all solutions except for *rippedleaf*. Solving also takes longer for ASP now, contrary to ILP, where solving gets faster for every graph or stays the same but uses fewer threads. For CP, solving takes almost the same time as for the 2-hop Dominating Set. Only *asymmetric* was solved noticeably faster.

	Solution in	Threads	Time single	Time all	Optimum	Models
(a) small-leaf	ASP	1	0.000	0.000	1	7
	ILP	1	0.0	–	1	–
	CP	1	0.37	0.786	1	7
(b) left-right-leaf	ASP	2	0.019	0.021	3	18
	ILP	4	0.016	–	3	–
	CP	1	0.639	1.52	3	18
(c) middle-leaf	ASP	2	0.020	0.020	3	18
	ILP	4	0.016	–	3	–
	CP	1	0.655	1.536	3	18
(d) bigger-leaf	ASP	2	0.023	0.039	4	2659
	ILP	1	0.016	–	4	–
	CP	1	0.671	56.214	4	2659
(e) rippedleaf	ASP	2	0.019	0.020	4	81
	ILP	1	0.0	–	4	–
	CP	1	0.671	4.249	4	81
(f) maple	ASP	2	0.031	0.047	5	569
	ILP	4	0.0	–	5	–
	CP	1	1.382	347.513	5	569
(g) asymmetric	ASP	4	1320.739	–	[13; 17]	*Unknown*
	ILP	4	0.188	–	14	–
	CP	4	8.184	–	14	*Unknown*

Table 3: Minimum 3-hop Dominating Set.

Now in Table 4, we have a look at the first connected variant: the Minimum Connected Dominating Set. Compared to the Minimum Dominating Set from Table 1, the runtimes increase drastically. CP is only able to solve *small-leaf* in under 20 minutes and states, that there only exist 5 optimal solutions, while ASP finds 24. Besides for *rippedleaf*, which is unsatisfiable, ASP's runtime increased a lot.

	Solution in	Threads	Time single	Time all	Optimum	Models
(a) small-leaf	ASP	2	0.006	0.005	5	24
	CP	1	0.703	19.574	5	5
(b) left-right-leaf	ASP	4	18.445	141.970	21	20088
	CP	4	1251.863	–	*Unknown*	*Unknown*
	CP*	1	268.31	–	[; 21]	–
(c) middle-leaf	ASP	4	313.022	678.339	21	20088
	CP	1	1264.8562	–	*Unknown*	*Unknown*
	CP*	1	59.93	3600	[; 21]	–
(d) bigger-leaf	ASP	24	561.827	4135.029	24	69482
	CP	1	1222.320	–	*Unknown*	*Unknown*
	CP*	24	66.65	3600	[; 24]	–
(e) rippedleaf	ASP	1	0.012	–	*UNSAT*	0
	CP	1	1582.000	–	*Unknown*	–
	CP*	1	–	3600	*Unknown*	*Unknown*
(f) maple	ASP	24	1202.277	–	[33; 40]	*Unknown*
	CP	4	1213.421	–	*Unknown*	*Unknown*
	CP*	24	54.58	3600	[; 39]	–
(g) asymmetric	ASP	4	1379.182	–	[28; 145]	*Unknown*
	CP	4	1258.207	–	*Unknown*	*Unknown*
	CP*	24	3426.46	3600	[; 134]	–

Table 4: Minimum Connected Dominating Set.

The canonical connected method decreases the number of optimal solutions, which leads to a reduced runtime, as can be seen in Table 5. Again CP is only able to get all optimal solutions for *small-leaf* and the number of those is different from the one ASP finds. With the exception of *rippedleaf* and *asymmetric*, the runtimes for solving and finding all optimal solutions of ASP got much faster. It even gets all solutions for *maple* in under 20 seconds, when before, it was not solvable with the non-canonical connected method in 20 minutes. Although CP could not find all optimal solutions, it took less time for solving and it detects the unsatisfiability of *rippedleaf*. Now it solves all test graphs in under 10 seconds, excluding *asymmetric*, where an optimal solution could not be found in 20 minutes. Also remarkable is, that the canonical method for CP states that a minimum dominating set for *maple* contains 40 nodes, while ASP states that it contains 41. Important to notice is that the canonical connected method does not return the optimal solution for every graph. We see this if we look at *middle-leaf*, which has 21 nodes with the connected, but 22 with the canonical connected method. Because *middle-leaf* only differs in node numeration from *left-right-leaf* (where the canonical method gets the correct optimum), we can assume that specific node numerations cause symmetry breaking to limit the search space too much.

	Solution in	Threads	Time single	Time all	Optimum	Models
(a) small-leaf	ASP	2	0.004	0.004	5	7
	CP	1	0.686	3.609	5	4
(b) left-right-leaf	ASP	4	0.047	0.197	21	117
	CP	1	3.841	–	21	*Unknown*
(c) middle-leaf	ASP	4	0.103	0.424	22	1152
	CP	1	3.477	–	22	*Unknown*
(d) bigger-leaf	ASP	4	0.112	0.221	24	156
	CP	1	4.212	–	24	*Unknown*
(e) rippedleaf	ASP	1	0.011	–	*UNSAT*	0
	CP	2	1.75	–	*UNSAT*	0
(f) maple	ASP	4	6.807	19.777	41	374848
	CP	1	7.853	–	40	*Unknown*
(g) asymmetric	ASP	4	1200.975	–	[72; 134]	*Unknown*
	CP	1	1261.083	–	*Unknown*	*Unknown*
	CP*	1	284.45	3600	[; 133]	–

Table 5: Minimum Canonical Connected Dominating Set.

Solutions for the Minimum 2-hop Connected Dominating Set problem are listed in Table 6. Although the 2-hop Connected Dominating Set method reduces the maximum solving runtime of ASP on *left-right-leaf*, *middle-leaf* and *bigger-leaf* from under 10 minutes to under 20 seconds, CP can only solve *small-leaf*, *middle-leaf* and *left-right-leaf*. ASP also can not solve *asymmetric* and *maple*, but gets tighter bounds compared to the non-k-hop method. The only other remarkable difference to the CP runtimes from Table 4 is that now due to a from 5 to 1 reduced amount of optimal solutions, the runtime for finding all solutions on *small-leaf* also reduces from under 20 to 1.5 seconds.

	Solution in	Threads	Time single	Time all	Optimum	Models
(a) small-leaf	ASP	2	0.006	0.006	1	1
	CP	1	0.734	1.406	1	1
(b) left-right-leaf	ASP	4	0.621	2.111	12	118
	CP	1	225.972	–	12	*Unknown*
(c) middle-leaf	ASP	4	1.054	3.698	12	118
	CP	1	93.182	–	12	*Unknown*
(d) bigger-leaf	ASP	4	17.451	29.086	13	4
	CP	1	1302.454	–	*Unknown*	*Unknown*
	CP*	1	147.73	3600	[; 13]	–
(e) rippedleaf	ASP	1	0.018	–	*UNSAT*	0
	CP	2	1319.521	–	*Unknown*	*Unknown*
	CP*	1	–	3600	*Unknown*	–
(f) maple	ASP	24	1205.141	–	[19; 24]	*Unknown*
	CP	1	1256.488	–	*Unknown*	*Unknown*
	CP*	1	93.92	3600	[; 24]	–
(g) asymmetric	ASP	24	1356.188	–	[20; 95]	*Unknown*
	CP	1	1286.573	–	*Unknown*	*Unknown*
	CP*	1	2437.63	3600	[; 90]	–

Table 6: Minimum 2-hop Connected Dominating Set.

As the 2-hop method got faster solutions than the non-k-hop method, this also applies to the 3-hop compared with the 2-hop method. This can be seen in Table 7. This time CP gets an optimal solution for *small-leaf*, *left-right-leaf* and *middle-leaf* in under 4 seconds and one for *bigger-leaf* in under 9 seconds. ASP's runtime further decreases on almost any test graph, but *asymmetric* and *maple* still can not be solved in 20 minutes.

	Solution in	Threads	Time single	Time all	Optimum	Models
(a) small-leaf	ASP	2	0.007	0.006	1	7
	CP	1	0.901	2	1	7
(b) left-right-leaf	ASP	4	0.099	0.171	7	124
	CP	1	3.874	–	7	*Unknown*
(c) middle-leaf	ASP	4	0.140	0.205	7	128
	CP	1	3.609	–	7	*Unknown*
(d) bigger-leaf	ASP	4	0.828	3.292	8	74
	CP	1	8.889	–	8	*Unknown*
(e) rippedleaf	ASP	1	0.021	–	*UNSAT*	0
	CP	1	1258.673	–	*Unknown*	*Unknown*
	CP*	1	–	3600	*Unknown*	–
(f) maple	ASP	24	1413.329	–	[15; 18]	*Unknown*
	CP	1	1212.473	–	*Unknown*	*Unknown*
	CP*	1	191.27	3600	[; 17]	–
(g) asymmetric	ASP	24	1480.644	–	[14; 63]	*Unknown*
	CP	1	1244.935	–	*Unknown*	*Unknown*
	CP*	1	3090.99	3600	[; 66]	–

Table 7: Minimum 3-hop Connected Dominating Set.

In Table 8, it is noticeable, that adding the k-hop constraint to the canonical dominating set results in a further search space limitation and therefore improves the runtimes slightly. ASP solves the 2-hop faster than the non-k-hop dominating set, but it still is not fast enough to solve *asymmetric* in 20 minutes. Even though the bounds got tighter this time. A slight improvement regarding runtime can also be noticed at the CP solutions, but not enough to solve *maple* or *asymmetric*.

	Solution in	Threads	Time single	Time all	Optimum	Models
(a) small-leaf	ASP	2	0.005	0.005	1	1
	CP	1	0.797	1.609	1	1
(b) left-right-leaf	ASP	4	0.046	0.088	12	13
	CP	1	3.468	–	12	*Unknown*
(c) middle-leaf	ASP	4	0.026	0.049	13	13
	CP	1	2.284	–	13	*Unknown*
(d) bigger-leaf	ASP	4	0.084	0.101	13	1
	CP	1	3.968	965.612	13	1
(e) rippedleaf	ASP	1	0.015	–	*UNSAT*	0
	CP	2	1.234	–	*UNSAT*	0
(f) maple	ASP	4	3.796	17.017	25	9784
	CP	1	6.934	–	25	*Unknown*
(g) asymmetric	ASP	24	1273.791	–	[44; 81]	*Unknown*
	CP	1	1446.638	–	*Unknown*	*Unknown*
	CP*	1	455.09	3600	[; 74]	–

Table 8: Minimum 2-hop Canonical Connected Dominating Set.

Table 9 contains the results for the Minimum 2-hop Rooted Connected Dominating Set problem. These results imply, that the rootedness limits the search space further except for *middle-leaf*. The runtimes of ASP improved compared to those of Table 6, or in the case of the two graphs *maple* and *asymmetric*, where ASP does not find a solution, the bounds get tighter. Even CP is able to solve *left-right-leaf*, *bigger-leaf* and detect *rippedleafs* unsatisfiability, additionally to solving *small-leaf*, which it was also able to in Table 6.

	Solution in	Threads	Time single	Time all	Optimum	Models
(a) small-leaf	ASP	2	0.006	0.006	3	2
	CP	1	0.728	1.4	3	1
	ILP	–	0.007	–	3	–
(b) left-right-leaf	ASP	4	0.419	1.595	14	132
	CP	1	109.416	–	14	*Unknown*
(c) middle-leaf	ASP	4	0.331	2.037	14	132
	CP	1	1215.514	–	*Unknown*	*Unknown*
	CP*	1	9.57	3600	[; 14]	–
	ILP	–	7.006	–	14	–
(d) bigger-leaf	ASP	4	0.677	1.538	15	4
	CP	1	331.049	–	15	*Unknown*
	ILP	–	18.178	–	15	–
(e) rippedleaf	ASP	1	0.012	–	*UNSAT*	0
	CP	4	1.405	–	*UNSAT*	0
(f) maple	ASP	24	1262.068	–	[25; 27]	*Unknown*
	CP	1	1220.406	–	*Unknown*	*Unknown*
	CP*	1	12.30	3600	[; 27]	–
	ILP	–	1074.401	–	[20, 26]	–
(g) asymmetric	ASP	24	1187.724	–	[21; 89]	*Unknown*
	CP	1	1212.456	–	*Unknown*	*Unknown*
	CP*	24	1759.51	3600	[; 82]	–
	ILP	–	1065.585	–	[38, 161]	–

Table 9: Minimum 2-hop Rooted Connected Dominating Set.

Though not every solution is optimal, the Minimum 2-hop Rooted Canonical Dominating Set results shown in Table 10, have much lower runtimes. Here we can again confirm, that the canonical method does not give us the optimal solution for every graph (*middle-leaf* has 15 nodes but optimal are 14) and that CP is only able to compete with ASP on the biggest instances. CP solves *maple* in about 7 seconds, while ASP needs about 1365 seconds. Because of the search space limitation caused by symmetry breaking, CP can find a solution for every graph except *asymmetric* now.

	Solution in	Threads	Time single	Time all	Optimum	Models
(a) small-leaf	ASP	2	0.005	0.005	3	1
	CP	1	0.701	1.367	3	1
(b) left-right-leaf	ASP	4	0.013	0.015	14	13
	CP	1	3.291	–	14	*Unknown*
(c) middle-leaf	ASP	4	0.015	0.023	15	70
	CP	1	2.152	–	15	*Unknown*
(d) bigger-leaf	ASP	4	0.016	0.017	15	1
	CP	1	3.869	–	15	*Unknown*
(e) rippedleaf	ASP	1	0.011	–	*UNSAT*	0
	CP	1	1.269	–	*UNSAT*	0
(f) maple	ASP	4	1365.163	0.307	27	10272
	CP	1	6.702	–	27	*Unknown*
(g) asymmetric	ASP	24	1365.163	–	[53; 83]	*Unknown*
	CP	1	1259.791	–	*Unknown*	*Unknown*
	CP*	24	761.43	3600	[; 74]	–

Table 10: Minimum 2-hop Rooted Canonical Connected Dominating Set.

Like mentioned before, the non-optimal solutions we get when we use the canonical methods on *middle-leaf* can be explained through node numeration. In Figure 1, we see *middle-leaf* with a line dividing the leaf in two. Every Node $v \in V$ outermost to the line has only edges $(v, w) \in E$ back to the innermost so that $w < v$. This interferes with the canonical methods because they ignore all edges $(u_1, u_2) \in E$, where $u_2 < u_1$. If we compare the results for *maple* in Table 10 and Table 6, with the upper bounds in Table 9 and Table 8, we notice that the results do not fit the upper bounds. This is probably because of an error in the *maple* data file. The right side of the leaf lacks a connection between the nodes 115 and 116 and therefore it is not mirror-symmetrical.

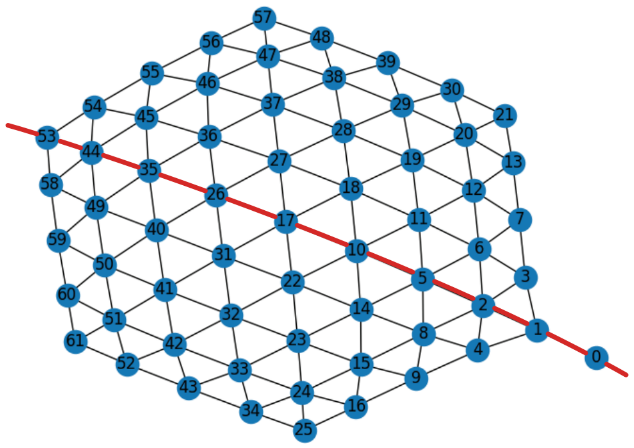
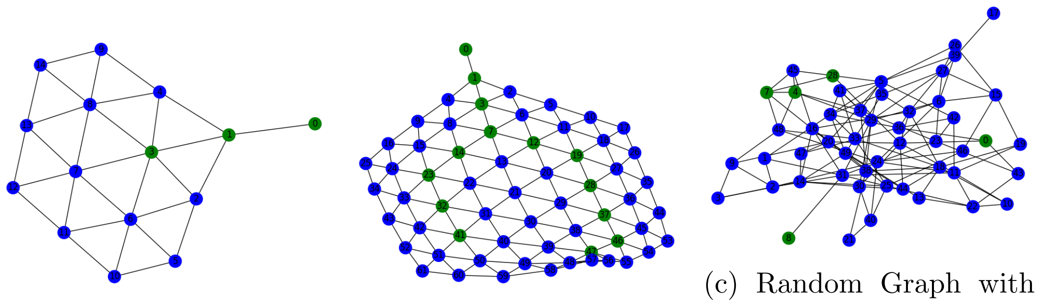


Figure 1: *middle-leaf* divided by a line.

In summary, CP is not able to solve Dominating Set variants with the same speed, that ILP and especially ASP can. Only on the biggest instances, CP can be faster than ASP. The same holds for ILP, it even outperforms CP on that heavily. Comparing the runtimes of Non-Connected and Connected Dominating Set variants of CP, it stands out that the Connected variants runtimes are higher than the Non-Connected ones. The numeration of *middle-leaf* was the cause of the non-optimal solutions we got when we used canonical methods. Another search space limitation is the increasement of the k parameter or ensuring that the dominating set contains the root node of the graph. As k increases, the runtimes get lower. But due to the restriction, that mesophyll cells can only be a few cells away from the next vein cell, we can not arbitrarily increase it. If we compare the results of CP in Table 9 and Table 6, we see again a slight improvement of speed caused by constraining the dominating set to contain the root node. But as these two only cause minor search space limitations, it does not suffice to find optimal solutions fast. The differences between the amount of optimal connected solutions found by ASP and CP occurs because ASP counts recurrent solutions [13]. The upper bounds that CP finds, are very often the optimal solution and were found relatively fast. This and that often CP could not detect the unsatisfiability of a graph implies that CP does not have a problem to get to a solution fast, but rather realizing in a feasible time that a found solution is optimal.

Figure 2: Minimum 2-hop Rooted Connected Dominating Set examples


(a) *small-leaf*

(b) *left-right-leaf*

(c) Random Graph with 50 nodes and a density of 0.1

Now we compare graphic representations of the k-hop Rooted Connected Dominating Set results on random graphs, in Figure 3 and Figure 4. The random graphs have a density of 0.1,0.5 or 0.9 and contain 50, 100, 250 or 500 nodes. The random graphs with 250 or more nodes and a density of 0.5 or higher could not be tested with $k = 3$ because the preprocessing of a 3-transitive closure could not be done, even when running for days.

Surprisingly, CP got a dominating set with 2 nodes, while ASP and ILP got one with 3 nodes on the random graph with 50 nodes and a density of 0.9. ASP and CP could not solve random graphs with 250 or 500 nodes and a density of 0.1 or 0.5 in under 20 minutes, while ILP only struggled with graphs that have 250 or 500 nodes and a density of 0.1. While increasing the density results in higher runtimes for graphs with 50 or 100 nodes, the opposite occurs, if a graph has 250 or 500 nodes. Then a density of 0.9 seems to work best. This only holds for $k = 1$, as increasing the density seems to increase the runtime if $k = 2$ or $k = 3$. Graphs with 250 nodes or more could be solved more efficiently with $k = 2$ or $k = 3$.

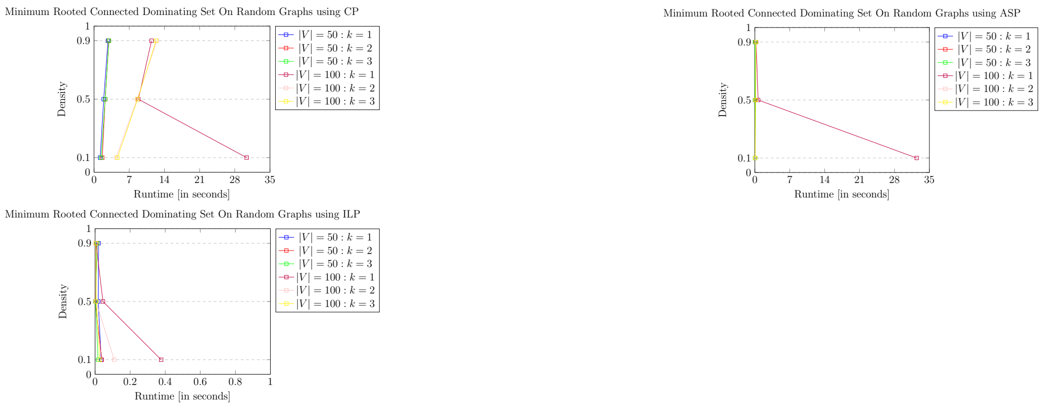


Figure 3: Minimum k-Hop Rooted Connected Dominating Set On Smaller Random Graphs

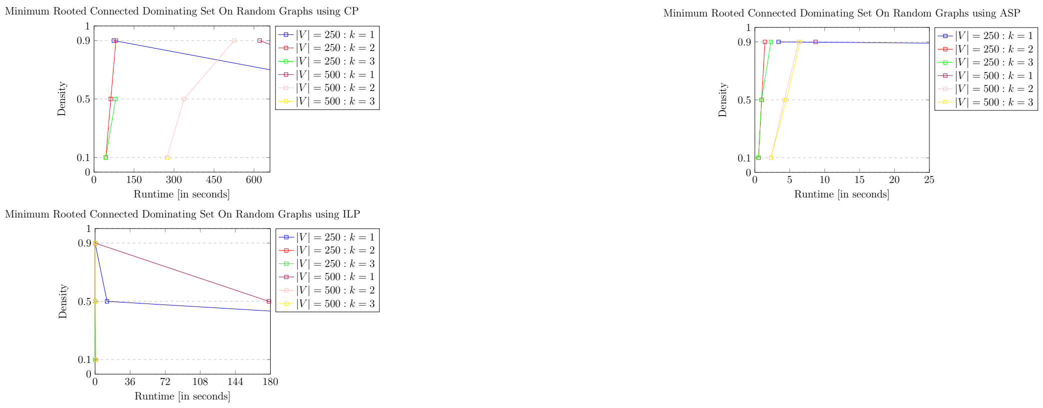


Figure 4: Minimum k-Hop Rooted Connected Dominating Set On Larger Random Graphs

Next, we look at results for the Minimum k-hop Rooted Connected Dominating Set on GRID graphs in Table 11. A GRID graph is numbered differently than the graphs before. Instead of giving every node $v \in V$ a number $n \in \mathbb{N}$, every node gets an x-coordinate and a y-coordinate on a 2-dimensional coordinate system. If two nodes $v, w \in V : v = (x_1, y_1) \land w = (x_2, y_2)$ share an edge: $(x_1 - x_2)^2 + (y_1 - y_2)^2 = 1$. Most strikingly, while ASP and ILP solve the 1-hop rooted connected dominating set in about 92 and 774 seconds for the quadratic graph GRID_8_8 and then improve much with increasing k (ASP is fastest with $k = 2$ and a bit slower than that with $k = 3$, while ILP gets faster with every k increasement), CP solves the 1-hop problem faster (in about 6 seconds) and then only minorly improves with increasing k. With this exception, ASP solves the GRID graphs the fastest. Besides GRID_8_8 and GRID_16_4 with $k = 1$, ILP's runtime lies between ASP's and CP's. Another conspicuousness is that increasing k to 3 on the GRID_16_4 graph, results in a much higher runtime for CP and ILP, while ASP has a better runtime than with $k = 1$.

	Solution in	k	Time single	Optimum
GRID_6_4	ASP	1	0.009	11
	CP	1	0.811	11
	ILP	1	0.054	11
GRID_6_4	ASP	2	0.011	7
	CP	2	0.768	7
	ILP	2	0.045	7
GRID_6_4	ASP	3	0.013	6
	CP	3	0.789	6
	ILP	3	0.056	6
GRID_8_8	ASP	1	92.739	26
	CP	1	6.670	26
	ILP	1	774.59	26
GRID_8_8	ASP	2	1.534	18
	CP	2	6.436	18
	ILP	2	81.971	18
GRID_8_8	ASP	3	1.747	15
	CP	3	6.274	15
	ILP	3	15.546	15
GRID_16_4	ASP	1	0.281	28
	CP	1	9.316	28
	ILP	1	42.569	28
GRID_16_4	ASP	2	0.014	17
	CP	2	3.364	17
	ILP	2	1.405	17
GRID_16_4	ASP	3	0.023	16
	CP	3	42.861	16
	ILP	3	7.973	16
GRID_18_2	ASP	1	0.010	18
	CP	1	0.934	18
	ILP	1	0.117	18
GRID_18_2	ASP	2	0.011	17
	CP	2	0.940	17
	ILP	2	0.148	17
GRID_18_2	ASP	3	0.013	16
	CP	3	0.910	16
	ILP	3	0.173	16
GRID_32_2	ASP	1	0.015	32
	CP	1	3.547	32
	ILP	1	0.262	32
GRID_32_2	ASP	2	0.015	31
	CP	2	3.434	31
	ILP	2	0.342	31
GRID_32_2	ASP	3	0.023	30
	CP	3	3.206	30
	ILP	3	0.287	30

Table 11: Minimum k-hop Rooted Connected Dominating Set on grid graphs.

The random graphs revealed to us, that the density and the k parameter have a major influence on the runtime. E.g., A graph with many nodes is faster solvable, when it has a high density, while a small graph gets faster results for lower density. The same holds for the k parameter. So on smaller graphs, we get lower runtimes if $k = 1$ and on bigger graphs, we get higher runtimes if $k = 1$. From the GRID graphs, we learned, that quadratic graphs seem to be a bigger problem for ILP and ASP than for CP.

While our model follows the possible Pareto optimality of a leaf, it still disregards a lot of other important aspects and functions of a leaf venation pattern. Temperature [6], vein density [16] and the vein hierarchy are some factors also affecting the venation pattern.

5 Conclusions

Even though CP solves a few graphs faster, it seems to be not as fitting for solving Connected Dominating Set problems as ILP and ASP are. CP has a major focus on letting the user formulate his problem as easy as possible, by using a close to mathematical formulation of it. This hides from them the complexity involved, which makes it difficult to estimate the quality of formulations. Accordingly, it could be useful to find out if different CP formulations of this problem can perform better.

A more promising approach could be to find ASP solutions with added CP compatibilities because ASP is more efficient if inductive definitions are involved and could profit from CP's better handling of integer variables [12].

Even though we were not able to solve all test leaves, we improved the runtimes by adding symmetry breaking constraints. So, if one would try to optimize this model, it would be beneficial to have only data files with a node numbering, so that the canonical connected method does only find optimal solutions. This would interfere with our intent to find all optimal solutions, but that was only working for some graphs with low amounts of solutions anyway.

6 Acknowledgement

I would like to express my gratitude towards Prof. Gunnar Klau for giving me the opportunity to write about this topic. A special thanks, goes to Eline van Mantgem for the support in the weekly meetings and for giving me fast and extensive feedback on my writing.

References

[1] Petra Hofstedt, Armin Wolf: Einführung in die Constraint Programmierung. Springer-Verlag, Berlin Heidelberg, 2007.

[2] Francesca Rossi, Peter Van Beek, Toby Walsh: Handbook of constraint programming. Elsevier Science, 2006.

[3] Rina Dechter: Constraint processing. Morgan Kaufmann, 2003.

[4] Peter J. Stuckey, Kim Marriott, Guido Tack: The MiniZinc Handbook. `https://www.minizinc.org/doc-2.5.0/en/index.html`

[5] Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, Guido Tack: MiniZinc: Towards A Standard CP Modelling Language. Springer, Berlin Heidelberg 2007.

[6] Adam Conn, Ullas V. Pedmale, Joanne Chory, Saket Navlakha: High-resolution laser scanning reveals plant architectures that reflect universal network design principles. Cell Systems, 5:53–62.e1-e3, July, 2017.

[7] Charles B. Beck: An Introduction to Plant Structure and Development. Cambridge, 2010.

[8] Dieter Heß: Pflanzenphysiologie: Grundlagen der Physiologie und Biotechnologie der Pflanzen. Verlag Eugen Ulmer, 2008.

[9] Martin Hodson, John A. Bryant: Functional biology of plants. Wiley-Blackwell, 2012.

[10] Park S. Nobel: Physicochemical and environmental plant physiology. Acad. Press, 1999.

[11] Marvin van Aalst: Optimality principles of leafvenation patterns. November 11, 2019.

[12] Rehan Abdul Aziz, Peter J. Stuckey, Zoltan Somogyi: Inductive Definitions in Constraint Programming. Proceedings of the Thirty-Sixth Australasian Computer Science Conference, Adelaide, Australia, 2013.

[13] My Ky Huynh: Solving dominating set using answer set programming. February,2020.

[14] Mario Surlemont: Solving connected dominating set variants using integer linear programming. August,2020.

[15] Ding-Zhu Du, Peng-Jun Wan: Connected Dominating Set: Theory and Applications. Springer New York, 2013.

[16] Anita Roth-Nebelsick, Dieter Uhl, Volker Mosbrugger, Hans Kerp: Evolution and Function of Leaf Venation Architecture: A Review. Proceedings of the Thirty-Sixth Australasian Computer Science Conference, Annals of Botany 87(5):553–566, 2001.

Erklärung

Hiermit versichere ich, dass ich die Bachelorarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Düsseldorf, den 01. März 2021

(Vorname Nachname)

A Appendix

	Solution in	k	Time single	Optimum
GNM_50_122	CP	1	1.22	11
GNM_50_122	CP	2	1.635	5
GNM_50_122	CP	3	1.415	2
GNM_50_612	CP	1	1.924	4
GNM_50_612	CP	2	2.216	1
GNM_50_612	CP	3	2.209	1
GNM_50_1102	CP	1	2.778	2
GNM_50_1102	CP	2	2.944	1
GNM_50_1102	CP	3	2.964	1
GNM_100_495	CP	1	30.347	14
GNM_100_495	CP	2	4.394	4
GNM_100_495	CP	3	4.651	1
GNM_100_2475	CP	1	8.813	4
GNM_100_2475	CP	2	8.565	1
GNM_100_2475	CP	3	8.681	1
GNM_100_4455	CP	1	11.436	2
GNM_100_4455	CP	2	12.504	1
GNM_100_4455	CP	3	12.340	1
GNM_250_3112	CP	1	1109.712	*Unknown*
	CP*	1	217.98	[; 19]
GNM_250_3112	CP	2	44.299	2
GNM_250_3112	CP	3	44.271	1
GNM_250_15562	CP	1	1246.502	*Unknown*
	CP*	1	87.84	[; 5]
GNM_250_15562	CP	2	62.901	1
GNM_250_15562	CP	3	80.984	1
GNM_250_28012	CP	1	74.471	2
GNM_250_28012	CP	2	82.095	1
GNM_250_28012	CP	3	*Missing*	*Missing*
GNM_500_12475	CP	1	1297.835	*Unknown*
	CP*	1	279.96	[; 34]
GNM_500_12475	CP	2	273.657	2
GNM_500_12475	CP	3	276.346	1
GNM_500_62375	CP	1	1224.946	*Unknown*
	CP*	1	190.70	[; 6]
GNM_500_62375	CP	2	337.665	1
GNM_500_62375	CP	3	*Missing*	*Missing*
GNM_500_112275	CP	1	621.256	2
GNM_500_112275	CP	2	525.95	1
GNM_500_112275	CP	3	*Missing*	*Missing*

Table 12: Minimum k-hop Rooted Connected Dominating Set on random graphs using CP.

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

Table 13: Minimum k-hop Rooted Connected Dominating Set on random graphs using ASP.

	Solution in	k	Time single	Optimum
GNM_50_122	ILP	1	0.035	11
GNM_50_122	ILP	2	0.038	11 (?)
GNM_50_122	ILP	3	0.017	2
GNM_50_612	ILP	1	0.018	4
GNM_50_612	ILP	2	0.002	1
GNM_50_612	ILP	3	0.003	1
GNM_50_1102	ILP	1	0.020	3
GNM_50_1102	ILP	2	0.012	1
GNM_50_1102	ILP	3	0.012	1
GNM_100_495	ILP	1	0.377	14
GNM_100_495	ILP	2	0.109	4
GNM_100_495	ILP	3	0.027	1
GNM_100_2475	ILP	1	0.045	4
GNM_100_2475	ILP	2	0.005	1
GNM_100_2475	ILP	3	0.007	1
GNM_100_4455	ILP	1	0.005	2
GNM_100_4455	ILP	2	0.004	1
GNM_100_4455	ILP	3	0.004	1
GNM_250_3112	ILP	1	1017.304	$[15; 17]$
GNM_250_3112	ILP	2	0.271	2
GNM_250_3112	ILP	3	0.142	1
GNM_250_15562	ILP	1	12.29	5
GNM_250_15562	ILP	2	0.258	1
GNM_250_15562	ILP	3	0.267	1
GNM_250_28012	ILP	1	0.025	2
GNM_250_28012	ILP	2	0.019	1
GNM_250_28012	ILP	3	0.023	1
GNM_500_12475	ILP	1	1004.921	$[13; 21]$
GNM_500_12475	ILP	2	1.124	2
GNM_500_12475	ILP	3	0.635	1
GNM_500_62375	ILP	1	178.496	5
GNM_500_62375	ILP	2	0.290	1
GNM_500_62375	ILP	3	0.545	1
GNM_500_112275	ILP	1	0.189	2
GNM_500_112275	ILP	2	0.148	1
GNM_500_112275	ILP	3	0.205	1

Table 14: Minimum k-hop Rooted Connected Dominating Set on random graphs using ILP.