ALMA MATER STUDIORUM UNIVERSITÀ DI BOLOGNA

DISI

Master in Artificial Intelligence

Giorgio Buzzanca - giorgio.buzzanca@studio.unibo.it

Very Large Scale Integration Design

Combinatorial Decision Making And Optimization - Project Work Report

---

Academic Year 2021 - 2022

Chapter 1

Introduction

1.1 The two dimensional strip packing problem

Very large scale integration involves integrating multiple transistors in a single microchip.

In the context of this project I had to work on a simplified version of the problem, where, given N blocks of width w_i and height h_i, and a strip of width W, the objective was to find a placement of the blocks on the strip such that the resulting height H of the strip was minimized, under the constraint that the blocks cannot overlap, i.e.

$$H^* = \min_H \quad s.t.$$

$$\forall i \in \{1, \ldots, N\} \quad \exists x_i, y_i$$

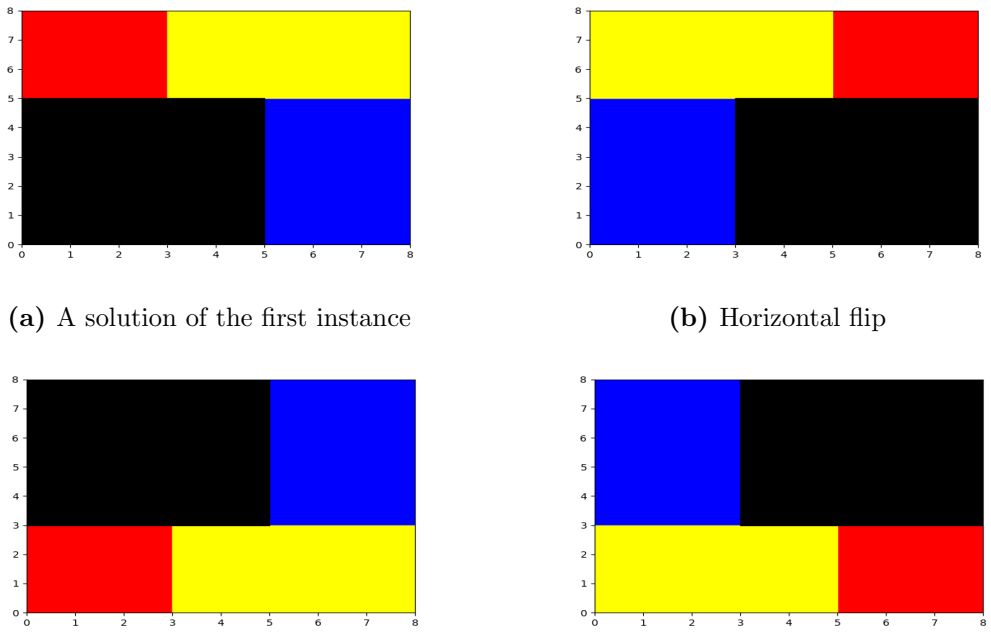
$$x_i \geq 0 \wedge x_i + w_i \leq W, \forall i \in \{1, \ldots, N\} \wedge$$

$$y_i \geq 0 \wedge y_i + h_i \leq H, \forall i \in \{1, \ldots, N\} \wedge$$

$$x_i + w_i \leq x_j \vee x_j + w_j \leq x_i \vee y_i + h_i \leq y_j \vee y_j + h_j \leq y_i, \forall i, j \in \{1, \ldots, N\}, i \neq j.$$

Two dimensional strip packing, as it is called, is a well known problem in the field of combinatorial optimization, falling under the category of bin packing problems. Bin packing problems are NP-hard, which means they can be solved in polynomial time by a non-deterministic Turing machine, with solutions being verifiable in polinomyal time, and they are at least as hard as any other problem in NP. Currently no polynomial time algorithm is known to solve bin packing, and we must either resort to polynomial-time algorithms computing approximate solutions, or to exact algorithms with exponential running time.

1.2 Symmetries in 2DSP



(a) A solution of the first instance

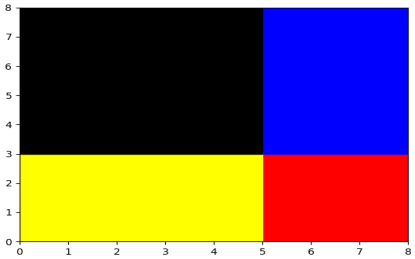
(b) Horizontal flip

(c) Vertical flip

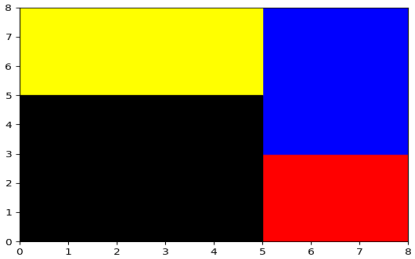
(d) Diagonal flip

Figure 1.1: Reflectional symmetries that are always present in a solution.

2DSP is a problem that presents many geometric symmetries. Starting from a

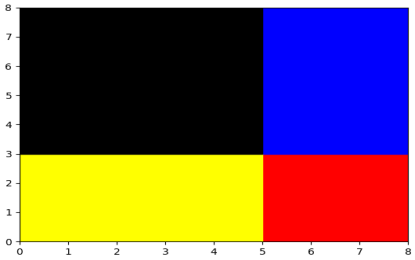


(a) A solution of the first instance

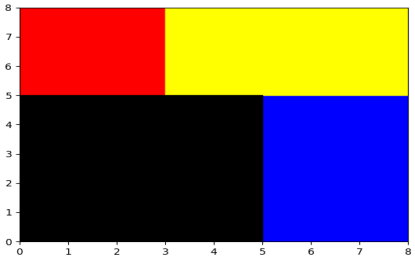


(b) Swapping of two aligned elements defining a vertical level

Figure 1.2: A different kind of symmetries, where two (or more) perfectly aligned circuits forming a macro-rectangle, can be swapped to form a new solution. In turn this new solution is subject to all reflectional symmetries seen before.



(a) Solution A



(b) Solution B

Figure 1.3: The two unique solutions for the first instance after all symmetries are broken.

solution, we can always apply horizontal, vertical or diagonal flip to obtain a new valid solution (as it can be seen in Figure 1.1). In addition, whenever some of the blocks form a regular macro-rectangle inside the board, we can move around these blocks without breaking the non-overlapping constraint (1.2), and thus obtaining new solutions.

These new solution can themselves be flipped, making the number of admissible solutions grow expontentially. For example, the first instance I was given, does

actually admit only two unique solutions, once all symmetries are broken (those shown in Figure 1.3), but the total number of solutions is 12.

1.3 Bounds for the height of the board

A sound lower bound for the height of the board is given by:

$$H_{LB} = \max\{\frac{\sum_{i\in\{1,\dots,N\}} w_i * h_i}{W}, \max_{i\in\{1,\dots,N\}}\{h_i\}\}$$

The absolute worst-case behaviour of this lower bound is $H_{LB} \leq \frac{1}{4}Opt(I)$ [1]. Tighter lower bounds can be found, but they are more expensive to compute, and not always applicable to the case where the blocks can be rotated. In addition, since all instances I was given can be solved exactly with no holes between circuits in the board, the WCPR of this lower bound is, in this case, equal to 1.

I implemented different heuristics to compute an upper bound for the problem, to speed up the search as much as possible, activating also warm start, providing solvers with an initial solution to start from, where it was possible.

- **Next Fit Decreasing Height**: Circuits are sort in decreasing order of height and a first level is initialized with the first circuit. Then, iterating through the circuits, as long as they fit, they are packed left justified on the current level. If the current level cannot accomodate the next circuit, a new level is initialized.

- **First Fit Decreasing Height**: Circuits are sort in decreasing order of height and a first level is initialized with the first circuit. Then each circuit is packed left justified on the first level that can accomodate it. If no level can accomodate it, a new level is initialized.

- **Best Fit Decreasing Height**: Circuits are sort in decreasing order of height and a first level is initialized with the first circuit. Then each circuit

is packed left justified on the level that can accomodate it with the least amount of space left. If no level can accomodate it, a new level is initialized.

- $\boldsymbol{KP_{OG}}$: This heuristic has been proposed by Lodi, Martello, and Vigo in [2], for the two dimensional strip packing problem, with oriented items and guillotine cutting. It is based on solving a series of 0-1 Knapsack sub-problems, where the objective is to find the subset of circuits which can be packed on the current level so that the total area of the placed circuits is maximized.

  The circuits are first sorted according to non-increasing value of height and then at each step the tallest element among those which have not yet been packed is selected to initialize a new level, and the corresponding sub-problem is solved.

Eventually I decided to use the KP_{OG} heuristic as the upper bound, since it outperformed all the others, as I was expecting.

1.4 A summary of the models implemented and the solvers used

For this project I was asked to model and solve 2D strip packing in different ways: as a CP, SAT, MIP, and SMT optimization problem.

For the CP part I used the Minizinc constraint modeling language, and the solvers chuffed and gecode, trying out three versions of one model obtained applying different symmetry breaking techniques, and comparing them.

For the SAT part I implemented the model described in [3], and used the solver z3.

For the MIP part I implemented two different models from the literature with their respective variations. The first model is the one described in [4], while the

second is the one described in [5], through which I also came to know about "Generalized Disjunctive Programming". I implemented these models making use of the PulP library, which is a Python library for linear programming, implementing many APIs for different solvers, and I tested them with the solvers CPLEX and Gurobi.

For the SMT part I implemented a procedure to translate the best performing variant of the CP model into the SMT-LIBv2 format, and automatically solve the instance with a solver of choice. I tested the procedure with the solvers z3 and CVC5.

I also implemented a variant of each of the aforementioned models to handle rotation of the blocks, and to my knowledge, as far as the SAT model is concerned, this represent a new contribution to the literature.

1.5 Utils implemented

Since I was given instances in a text file format unsuitable for Minizinc, I also implemented a parser to read the instances and convert them into json and dzn formats. It is during this conversion that the lower bounds and the upper bounds for the board height are computed. In addition, I implemented different scripts for automated solving of the instances, for plotting the solutions obtained, and to validate their correctness.

Contents

1 Introduction **1**

1.1 The two dimensional strip packing problem 1

1.2 Symmetries in 2DSP 2

1.3 Bounds for the height of the board 4

1.4 A summary of the models implemented and the solvers used 5

1.5 Utils implemented 6

2 Constrained Programming **9**

2.1 Decision Variables 9

2.2 Constraints 10

2.3 Objective Function 14

2.4 Rotation 14

2.5 Validation 15

3 SAT **23**

3.1 Decision Variables 25

3.2 Constraints 26

3.3 Objective Function 28

3.4 Rotation 28

3.5 Validation 29

4 MIP Models - Positions and Covering Methodology 34

4.1 Decision Variables 36

4.2 Constraints 36

4.3 Objective Function 37

4.4 Rotation 37

4.5 Validation 39

5 MIP Models - GDP to MIP with Big-M Reformulation 43

5.1 Decision Variables 45

5.2 Constraints 46

5.3 Objective Function 47

5.4 Rotation 48

6 SMT Model 52

6.1 Decision Variables 53

6.2 Constraints 53

6.3 Objective Function 55

6.4 Rotation 56

6.5 Validation 56

7 Conclusion 61

Bibliography 64

Chapter 2

Constrained Programming

Constraint programming is a potent declarative programming paradigm that leverages concepts from fields like artificial intelligence and operations research to tackle combinatorial optimization challenges. A constraint satisfaction problem (CSP) involves a collection of variables, each with its own set of possible values (the domain), and a set of relationships between subsets of these variables. The user defines the constraints, while an underlying solver perform a search in the state-space of the problem to find a solution, i.e. an assignment of values in the domains to the variables that satisfies all the constraints. All CP models were implemented with MiniZinc.

2.1 Decision Variables

Following the Minizinc notation, I will call all the entities whose values are to be determined by the solver *variables*, while I will call constant terms parameters. The parameters of the model are:

- N: The number of variables,
- L: The width of the board,

- w_i: The width of the circuit at index i,
- h_i: The height of the circuit at index i,
- H_{LB}: The lower bounds on the height of the board,
- H_{UB}: The upper bounds on the height of the board,
- *order*: A permutation of the index set of the variables, such that the area of the circuit at index i is greater than or equal to the area of the circuit at index j if $i < j$. This is introduced to set up a symmetry breaking constraint as later discussed.

The decision variables are:

- x_i: The x-coordinate of the bottom-left corner of the circuit at index i. This variable has domain $[0, L - w_i]$.
- y_i: The y-coordinate of the bottom-left corner of the circuit at index i. This variable has domain $[0, H_{UB} - h_i]$.
- H: The height of the board. This variable has domain $[H_{LB}, H_{UB}]$.

2.2 Constraints

The basic constraints which are needed to correctly model a 2DSP instance are the non-overlapping constraint, and the actual bounds on the x and y coordinates of the circuits. The non-overlapping constraint has already been introduced in 1.1, and it is implemented in Minizinc via the global constraint "diffn"[6]. The actual bound on the coordinates of the circuits is implemented via the following

constraints:

$$\max_{i \in [1,N]} y_i + h_i \leq H$$

$$\max_{i \in [1,N]} x_i + w_i \leq L$$

Aside from these basic constraints, the model can be augmented with additional constraints to improve the performance of the solver. The first set of additional constraints is the *implied constraints*, which are constraints that are logically entailed by the basic ones. The second set of additional constraints is the *symmetry breaking constraints*, which address the symmetries discussed on 1.2.

Implied Constraints

The first implied constraint imposes that pair of circuits whose dimension would exceed taken together the board width (height), cannot be placed one next to the other horizontally (vertically):

$$\forall_{i,j \in [1,N]} (w_i + w_j > L) \rightarrow (y_i + h_i \leq y_j \vee y_i >= y_j + h_j)$$

$$\forall_{i,j \in [1,N]} (h_i + h_j > H) \rightarrow (x_i + w_i \leq x_j \vee x_i >= x_j + w_j)$$

The second implied constraint can be obtained reasoning on the parallelism between strip packing and scheduling. We can also think of strip packing in the horizontal (vertical) dimension as a scheduling problem where the resource is the height (width) of the board, the tasks are the circuits, the start time of the tasks is the x (y) coordinate, the duration is the width (height), the resource usage is the height (width), and the global resource bound is the height (width) of the board. This translates into:

$$\forall_{t\in[0,L+max_{i\in[1,N]}w_i]}\left(H \geq \sum_{i\in[1,N]}(1_{x_i\leq t<x_i+w_i}\cdot h_i)\right)$$
$$\forall_{t\in[0,H+max_{i\in[1,N]}h_i]}\left(L \geq \sum_{i\in[1,N]}(1_{y_i\leq t<y_i+h_i}\cdot w_i)\right).$$

This constraint is implemented in the MiniZinc model via the global constraint "cumulative" [6].

Symmetry Breaking Constraints

The reflectional symmetries are broken by imposing a lexicographic ordering on the x and y coordinates of the circuits wrt the reflected version. Two arrays a and b of length n are said to be lexicographically ordered iff:

$$\forall_{k\in[1,n]}((\forall_{i\in[1,k)}a_i = b_i) \implies a_k \leq b_k)$$

This constraint has been the subject of much research and various methods of decomposition have been proposed. By default, MiniZinc employs the "Alpha M Lex Encoding", which utilizes a Boolean array to keep track of relationships between values in both vectors.

$$\alpha_1$$
$$\forall_{i\in[1,N]}\left(\alpha_i \iff (((a_i < b_i)\vee\alpha_{i+1})\wedge(a_i\leq b_i))\right)$$

This encoding produces $4n + 1$ number of atom occurrences.

I decided to implement a different decomposition of the lexicographic ordering, namely the "Recursive OR Decomposition"[7], because when using chuffed as a solver I got better results in terms of time needed to find a solution. In this work, the authors decompose the lex relation into a set of nested ORs and ANDs:

$$a_1 < b_1 \vee (a_1 = b_1 \wedge \ldots \wedge (a_{n-1} = b_{n-1} \wedge a_n \leq b_n))$$

An array of booleans x, of size n, can be introduced to eliminate the common sub-expressions and simulate recursion.

$$x_1$$
$$x_n \iff a_n \leq b_n$$
$$x_{n-i} \iff (a_{n-i} < b_{n-i} \vee (a_{n-i} = b_{n-i} \wedge x_{n-i+1}))$$

This decomposition produces $2n + 1$ number of atom occurrences.

So, bearing in mind that the lex_lesseq relation is decomposed with the "ROR" decomposition, the symmetry breaking constraints for reflectional symmetry are:

$$\forall_{i \in [1,N]} \left(lex\_lesseq(x_i, L - x_i - w_i)\right)$$
$$\forall_{i \in [1,N]} \left(lex\_lesseq(y_i, H - y_i - h_i)\right)$$

The spatial relationship between two arbitrary circuits can be always fixed, to reduce the search space, by imposing the lexicographic ordering on the coordinates of the two circuits. This translates into the first rectangle being placed on the

bottom-left half of the board with respect to the second one. In particular, I applied this constraint to the two largest circuits.

$$lex\_lesseq([y_{order_1}, x_{order_1}], [y_{order_2}, x_{order_2}])$$

In addition, since circuits of the same size can be swapped in a solution, another spatial relationship like **??** can be introduced.

$$\forall_{i \in [1,N]} \text{ let } J = \{j \in [1,N] \text{ s.t. } w_j = w_i \wedge h_j = h_i \wedge j \geq i\} \text{ in}$$
$$\forall_{j \in J | j > 1} \left(lex\_lesseq([y_{order_{j-1}}, x_{order_{j-1}}], [y_{order_j}, x_{order_j}]) \right)$$

2.3 Objective Function

The objective function is simply the height of the board, which shall be minimized.

2.4 Rotation

To handle rotation of the circuits I introduced a boolean array r of size N, where r_i is true if the circuit i is rotated, and false otherwise, and new variables rw_i and rh_i, which are the effective width and height of the circuit i when taking the rotation state into account.

These new basic constraints are added to the model:

$$\forall_{i \in [1,N]} \left(rw_i = \begin{cases} h_i, & \text{if } r_i \\ w_i, & \text{otherwise} \end{cases}, rh_i = \begin{cases} w_i, & \text{if } r_i \\ h_i, & \text{otherwise} \end{cases} \right)$$

The model is the same as the one described in the previous section, except for the fact that the width and height of the circuits are now replaced with rw_i

and rh_i. In addition, a new symmetry breaking constraint is now added, i.e. the rotation state of square-shaped circuits is fixed to false.

$$\forall_{i \in [1,N]} \left(w_i = h_i \implies r_i = 0 \right)$$

2.5 Validation

I run the model with chuffed and gecode solvers, evaluating different search strategies. MiniZinc implements search search annotations to control the search process. Not every solver implements all the search annotations MiniZinc provides, so I used different annotations for gecode and chuffed.

In particular, we can control the order with which the variables are searched, the domain exploration strategy, and the restarting behaviour. Restarting is a powerful technique to avoid getting stucked in a configuration which is not feasible, and whose exploration in depth would take too much time and lead possibly to a dead end.

Restarting in general can happen after the search-tree has reached a certain depth, when a limit number of nodes has been explored, or when a certain amount of time has passed. MiniZinc annotations allow us to set a limit on the number of nodes explored, in different ways. Geometric restarting for example impose that the k-h restart has a node limit of $scale * base^k$, where $scale$ and $base$ are parameters of the annotation. Luby restarting instead impose that the k-h restart has a node limit of $scale * L[k]$, where $scale$ is a parameter of the annotation, and L is the Luby sequence.

The search strategies I tried are:

- **Sequential Simple Input_Order and Indomain_Split**: in this case the search is completely deterministic and sequential, x and y possible assignments are searched in the circuits' order, and the values are assigned bisecting

the domain, excluding the upper half. This strategy is the fastest for most of the instances, but completely fails in a couple of cases. For instance, when solving the instance 37, 38, and 39, it takes respectively 445msec, 21s and 230msec, 711msec, while, when solving with the default search with luby restart, it takes respectively 50s and 74msec, 154s and 488msec, 46s and 419msec. In contrast, it cannot even find a sub-optimal solution for instance 32, or instance 40, while this is not the case for the default search with luby restart.

- **Sequential Area Input_Order and Indomain_Split**: as the previous one, this is a deterministic search, but in this case the input variables are sort according to the area of the corresponding circuit, in descending order. This strategy does not suffer from hanging on some instances, but it is slower than the previous one, and the default search with luby restart.

- **Dom_W_Deg with Luby_Restart(100) and Indomain_Min**: in this case the search is not deterministic, indeed the dom_w_deg search annotation make the solver choose the "variable with the smallest value of domain size divided by weighted degree which is the number of times it has been in a constraint that caused failure earlier in the search", and luby restart is enforced. This annotation is the best-performing when using the gecode solver, according to my experimental results.

- **Default Search with Luby_Restart(100)**: this is the best-performing, overall, strategy when using the chuffed solver.

In this report I only report results for the experiments with the best-performing configurations.

Experimental Design

Aside from the solution (i.e. the assigned coordinates), I collected the following information:

- The final board height,
- Whether the solution obtained was proven to be optimal or not,
- The time elapsed.

To automatically perform a huge number of experiments without human-intervention needed, I implemented a python script which let the user choose the mzn model and the solver and run through the instances, writing solutions to a directory, plotting them if required, and collecting all the statistics needed for the evaluation of the models.

I also compared the performances of the models when using symmetry breaking constraints or not, and when allowing rotations of the circuits. I used Gecode only for the symmetry breaking enabled case, since otherwise it would have taken too much time to obtain the metrics.

All experiments were runned on a machine with an Intel Core i7-7700HQ CPU @ 2.80GHz, 16GB of RAM, and a 256GB SSD, with MiniZinc version Version 2.6.4, with chuffed version 0.10.4, and gecode version 6.3.0.

Experimental Results

In table 2.1 I report the results of the experiments relative to the standard version of the problem, while in 2.2 those for the version with rotation allowed. In addition, I showed a comparison of the execution times for chuffed and gecode for the standard version with symmetry breaking enabled 2.1. As it can be noticed,

CP performs extremely well in this problem, and symmetry breaking does not substantially improve performanes.

Id	Chuffed + SB	Chuffed w/out SB	Gecode + SB
ins-1	8	8	8
ins-2	9	9	9
ins-3	10	10	10
ins-4	11	11	11
ins-5	12	12	12
ins-6	13	13	13
ins-7	14	14	14
ins-8	15	15	15
ins-9	16	16	16
ins-10	17	17	17
ins-11	18	18	19
ins-12	19	19	19
ins-13	20	20	20
ins-14	21	21	21
ins-15	22	22	22
ins-16	23	23	24
ins-17	24	24	24
ins-18	25	25	27
ins-19	26	26	27
ins-20	27	27	28
ins-21	28	28	29
ins-22	29	29	30
ins-23	30	30	30

ins-24	31	31	32
ins-25	32	32	34
ins-26	33	33	33
ins-27	34	34	34
ins-28	35	35	36
ins-29	36	36	50
ins-30	37	37	39
ins-31	38	38	38
ins-32	39	39	N/A
ins-33	40	40	40
ins-34	40	40	44
ins-35	40	40	N/A
ins-36	40	40	49
ins-37	60	60	N/A
ins-38	60	61	72
ins-39	60	60	74
ins-40	92	92	N/A

Table 2.1: Minimal height found for which 2DSP was satisfiable without rotation allowed.

Id	Chuffed + SB	Chuffed w/out SB
ins-1	8	8
ins-2	9	9
ins-3	10	10
ins-4	11	11

ins-5	12	12
ins-6	13	13
ins-7	14	14
ins-8	15	15
ins-9	16	16
ins-10	17	17
ins-11	18	18
ins-12	19	19
ins-13	20	20
ins-14	21	21
ins-15	22	22
ins-16	23	23
ins-17	24	24
ins-18	25	25
ins-19	26	26
ins-20	27	27
ins-21	28	28
ins-22	29	29
ins-23	30	30
ins-24	31	31
ins-25	33	33
ins-26	33	33
ins-27	34	34
ins-28	35	35
ins-29	36	36
ins-30	38	38
ins-31	38	38

ins-32	40	40
ins-33	40	40
ins-34	40	40
ins-35	40	40
ins-36	40	40
ins-37	61	61
ins-38	60	61
ins-39	61	61
ins-40	92	92

Table 2.2: Minimal height found for which 2DSP was satisfiable with rotation allowed.

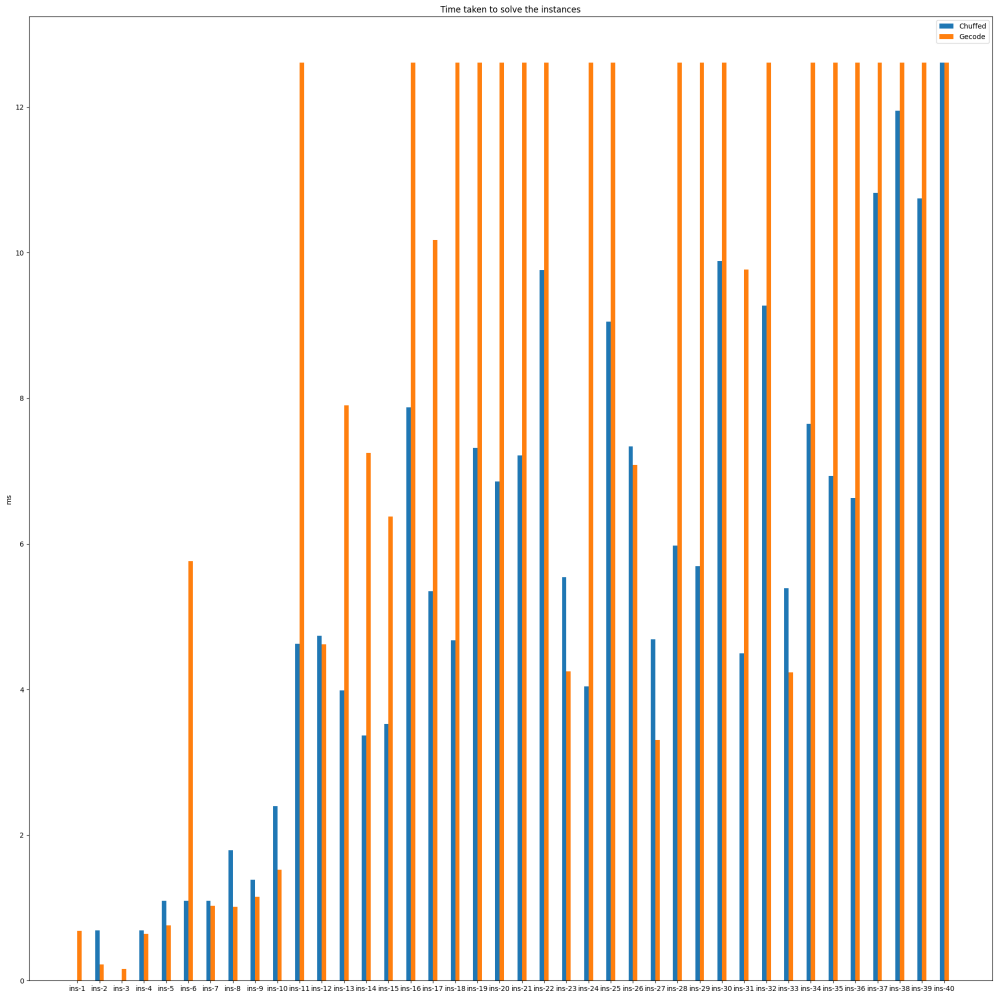


Figure 2.1: A comparison of the execution times of Chuffed and Gurobi for the instances with no rotation allowed and symetry breaking enabled. A logarithmic transformation has been applied to rescale the bars.

Chapter 3

SAT

Boolean satisfiability is the problem of determining whether there exists an assignment for a given propositional formula which makes the formula evaluate to true. Propositional formulas are constructed using propositional variables, logical connectives (such as AND, OR, NOT, etc.), and logical operators (such as parentheses) to represent statements that can be either true or false inside a theory.

Constrained satisfaction problems can be mapped into SAT problems by encoding the constraint of the problem in propositional logic.

An encoding of a constraint C into SAT is a CNF (conjunctive normal form) F that expresses C, so that any solution to C is a model of F. Examples of encodings from CSP to SAT are:

- The direct encoding, where for each variable $v \in V$, and domain value $i \in D_v$, a SAT variable $x_{v,i}$ is introduced, to represent the assignment $v = i$. In addition, the at-least-one ($\forall_{v \in V} \bigvee_{i \in D_v} x_{v,i}$) and at-most-one ($\forall_v \forall_{i,j \in D_v | i \neq j} \neg x_{v,i} \vee \neg x_{v,j}$) clauses are added, and to handle conflicts (incompatible assignments), the clauses $\neg x_{v,i} \vee \neg x_{v',i'}$ are added for each pair of variables v, v' and values i, i' that are incompatible.

- The logarithmic encoding, which allows to exponentially reduce the number of variables used for the encoding with respect to the direct encoding. Here instead of having one boolean variable for each CSP variable and for each value in the domain, we introduce a boolean variable for each CSP variable and bit of the base 2 representation of the domain. So, supposing variable v has domain $D_v = \{i | i \in \mathbb{N}, i <= 15\}$, and it could take any value in the domain (i.e. from 0 to 15), 16 introduced variables would be needed with the direct encoding, while it would only require 4 variables with the logarithmic encoding. Indeed, only 4 bits are required to represent numbers from 0 to 15. This way no at-least-one or at-most-one clauses are required. It suffices to introduce conflict causes for values not in the domain.

Following the work of Soh et al. in [3], I used the order encoding, firstly introduced by Tamura et al. in [8], to model the 2DSP problem.

The idea at the base of order encoding is to encode comparisons $v \leq a, a \in D_v$ by introducing a boolean variable, $p_{v,a}$, for each integer variable v and domain value a.

This encoding allows a natural representation of the order relation on integers; for example it it's particularly easy to represent the constraint $x <= y$: $\{x \leq a \vee \neg(y \leq a)\}$ for each integer a in the domain.

It is required to introduce a set of so-called axiom clauses to obtain a sound encoding. We have a set of axiom clauses for each variable v in the CSP. Assuming v ranges from $lb \in \mathbb{Z}$ to $ub \in \mathbb{Z}$, the axiom clauses for v are: $\bigwedge_{a \in [lb,ub)} (p_{v_a} \vee \neg p_{v_{a+1}})$.

A linear inequality constraint between two or more variables (i.e. $\sum_{i=1}^{n} a_i * x_i \leq$

c, such that $c \geq lower\_bound(\sum_{i=1}^{n} a_i * x_i))$, can be encoded as it follows:

$$\bigwedge_{\sum_{i=1}^{n} b_i = c-n+1} \bigvee_i (a_i * x_i \leq b_i)^{\#} \text{ with}$$

$b_i \in \mathbf{Z}$ such that $\sum_{i=1}^{n} b_i = c - n + 1 \wedge lower\_bound(a_i * x_i) - 1 \leq b_i \leq upper\_bound(a_i * x_i)$ and

$$(a * x \leq b)^{\#} \equiv \begin{cases} x \leq \lfloor \frac{b}{a} \rfloor, & \text{if } a > 0 \\ \neg(x \leq \lceil \frac{b}{a} \rceil - 1), & \text{if } a < 0 \end{cases}$$

In my implementation I extended the domain of each variable v with the value $lb_v - 1$, and added the constraints $\neg p_{v_{lb-1}} \wedge p_{v_{ub}}$.

To perform optimization in a SAT setting, we can turn the COP into a series of SAT instances, with progressively reducing (or increasing) objective. Solvers as z3 allow the user to "push" the state of a model, add the new constraint and check for satisfiability. Using this mechanism, I implemented a binary search over the domain of the board's height.

3.1 Decision Variables

The constant parameters of the model are the same as in CP. The decision variables are:

- $p_{x_i,v}, i \in [1, N], v \in [-1, L - w_i]$, the order encoding variables for the x coordinate of each circuit,
- $p_{y_i,v}, i \in [1, N], v \in [-1, H_{UB} - h_i]$, the order encoding variables for the y coordinate of each circuit,
- $lr_{i,j}, i \in [1, N], j \in [1, N]$, whether the i-th circuit is on the left of the j-th circuit,

- $ud_{i,j}, i \in [1, N], j \in [1, N]$, whether the i-th circuit is on the bottom of the j-th circuit,
- $p_{h,v}, v \in [H_{LB} - 1, H_{UB}]$, the actual height of the board.

3.2 Constraints

First of all we have the axiom-clauses for the order encoding of the variables:

$$\begin{gathered}\forall_{i\in[1,N]}(\neg p_{x_i,-1} \wedge p_{x_i,L-w_i} \wedge \forall_{v\in[1,L-w_i]}(\neg p_{x_i,v-1} \vee p_{x_i,v})) \\ \forall_{i\in[1,N]}(\neg p_{y_i,-1} \wedge p_{y_i,H_{UB}-h_i} \wedge \forall_{v\in[1,H_{UB}-h_i]}(\neg p_{x_i,v-1} \vee p_{x_i,v})) \\ \neg p_{h,H_{LB}-1} \wedge p_{h,H_{UB}]} \wedge \forall v \in [H_{LB}, H_{UB}](\neg p_{h,v-1} \vee p_{h,v})\end{gathered} \tag{3.1}$$

Then, we must actually impose that the y coordinates of the circuits are less than or equal to h:

$$\forall_{i\in[1,N]}(\forall_{v\in[H_{LB},H_{UB}-1]}(\neg p_{h,v} \vee p_{y_i,v-h_i})) \tag{3.2}$$

The non-overlapping constraints introduced in [3] are the following:

$$\begin{gathered}\forall_{i,j\in[1,N],i<j}(\forall_{e,0\le e<L-w_i}((\neg lr_{i,j} \vee p_{x_i,e} \vee \neg p_{x_j,e+w_i}))) \\ \forall_{i,j\in[1,N],i<j}(\forall_{e,0\le e<L-w_j}((\neg lr_{j,i} \vee p_{x_j,e} \vee \neg p_{x_i,e+w_j}))) \\ \forall_{i,j\in[1,N],i<j}(\forall_{f,0\le f<H_{UB}-h_i}((\neg ud_{i,j} \vee p_{y_i,f} \vee \neg p_{y_j,f+h_i}))) \\ \forall_{i,j\in[1,N],i<j}(\forall_{f,0\le f<H_{UB}-h_j}((\neg ud_{j,i} \vee p_{y_j,f} \vee \neg p_{y_i,f+h_j}))) \\ lr_{i,j} \vee lr_{j,i} \vee ud_{i,j} \vee ud_{j,i}\end{gathered} \tag{3.3}$$

Clearly, in 3.3 when it comes to the practical implementation, it must be take into account that x_i and x_j, as well as y_i and y_j, have different domains, so an additional check must be enforced.

Another possibility is to implement the non-overlapping constraints using the encoding of the linear inequality constraint discussed before, which is what I did in

my project, since I had already coded it, and I wanted to test it thoroughly. In addition, it has some advantadges when it comes to the model which allowes rotation of the circuits. Theoretically the set of clauses produced should be equivalent.

The last fundamental constraint which must be enforced is the actual value of the board height $\hat{h}$: $p_{h,\hat{h}}$. This constraint is the last one which is added before checking for satisfiability.

Implied Constraints

The first implied constraint I decided to add is the one regarding large circuits, which cannot be place side-by-side either in the horizontal or vertical dimension:

$$\begin{aligned} &\forall_{i,j\in[1,N] \text{ s.t. } j>i \wedge w_i+w_j>L}(\neg lr_{i,j} \wedge \neg lr_{j,i}) \\ &\forall_{i,j\in[1,N] \text{ s.t. } j>i \wedge h_i+h_j>\hat{h}}(\neg ud_{i,j} \wedge \neg ud_{j,i}) \end{aligned} \tag{3.4}$$

The other one is the following, which translates the concept that one circuit cannot be on the left and on the right of another one at the same time:

$$\begin{aligned} &\forall_{i,j\in[1,N] \text{ s.t. } j>i}(\neg lr_{i,j} \vee \neg lr_{j,i}) \\ &\forall_{i,j\in[1,N] \text{ s.t. } j>i}(\neg ud_{i,j} \vee \neg ud_{j,i}) \end{aligned} \tag{3.5}$$

Symmetry Breaking Constraints

I implemented all the symmetry breaking constraints discussed in the paper, but I eventually decided to use only the following ones:

- Fixing the spatial relation between circuits with the same size:

$$\forall_{i,j\in[1,N] \text{ s.t. } j>i \wedge w_i=w_j \wedge h_i=h_j}(\neg lr_{j,i} \wedge (lr_{i,j} \vee \neg ud_{j,i})) \tag{3.6}$$

- Fixing the spatial relation between two arbitrary circuits i and j (in particular between the two largest circuits):

$$\neg lr_{i,j} \wedge \neg lr_{j,i} \tag{3.7}$$

3.3 Objective Function

The objective function is the board height as usual.

3.4 Rotation

To handle rotation of the circuits I had to introduce the following changes:

- I had to extend the model with a new set of decision variables $r_i | i \in [1, N]$, where r_i is true when the i-th circuit is placed on the board rotated,
- The domains of the variables x_i and y_i for a given circuit i were redefined as $D_{x_i} \equiv [0, L - \min\{w_i, h_i\}]$ and $D_{y_i} \equiv [0, H_{UB} - \min\{w_i, h_i\}]$,
- I had to introduce a set of clauses to enforce the true range of admissible values for x_i and y_i taking into account r_i.

$$\begin{aligned}
\forall_{i\in[1,N]}(r_i \implies (& \bigwedge_{j\in[L-h_i, L-\min\{w_i,h_i\}]} p_{x_i,j}, \bigwedge_{j\in[H_{UB}-w_i, H_{UB}-\min\{w_i,h_i\}]} p_{y_i,j})) \\
\forall_{i\in[1,N]}(\neg r_i \implies (& \bigwedge_{j\in[L-w_i, L-\min\{w_i,h_i\}]} p_{x_i,j}, \bigwedge_{j\in[H_{UB}-h_i, H_{UB}-\min\{w_i,h_i\}]} p_{y_i,j}))
\end{aligned} \tag{3.8}$$

- I had to modify the constraint needed to enforce the actual board height:

$$\begin{aligned}
\forall_{i\in[1,N]}(\forall_{v\in[H_{LB}, H_{UB}-1]}(\neg r_i \implies (\neg p_{h,v} \vee p_{y_i, v-h_i}))) \\
\forall_{i\in[1,N]}(\forall_{v\in[H_{LB}, H_{UB}-1]}(r_i \implies (\neg p_{h,v} \vee p_{y_i, v-w_i})))
\end{aligned} \tag{3.9}$$

- I had to consider the four possible cases when enforcing the non-overlapping constraint between i and j (i.e. $r_i \wedge r_j$, $\neg r_i \wedge r_j$, $r_i \wedge \neg r_j$, $\neg r_i \wedge \neg r_j$). Fortunately having implemented the general linear inequality constraint for order-encoded integer variables, it was easy. Indeed, given a function

  $non\_overlapping\_i\_j(x_i, x_j, y_i, y_j, w_i, h_i, w_j, h_j)$ (in the Python code it has a different signature, for software engineering practical reasons) returning the set of clauses of the non-overlapping constraint, it suffices to run through all the cases and enclose this clauses inside an implication of the form case considered $\implies$ set of clauses . As an example, when both i and j are rotated, the constraint would be:

$$\begin{aligned} &\forall_{i,j\in[1,N],\text{ s.t. } i<j}((r_i \wedge r_j) \implies \\ &(non\_overlapping\_i\_j(x_i, x_j, y_i, y_j, h_i, w_i, h_j, w_j) \wedge \\ &non\_overlapping\_i\_j(x_j, x_i, y_j, y_i, h_j, w_j, h_i, w_i)) \end{aligned} \tag{3.10}$$

- I had to modify the large circuits constraints to handle rotation, and, as for the non-overlapping ones, I did it considering all the possible cases.

In addition, I introduced a new symmetry breaking constraint to fix the rotation state of the square-shaped circuits:

$$\forall_{i\in[1,N]\text{ s.t. } w_i=h_i}(\neg r_i) \tag{3.11}$$

3.5 Validation

Experimental Design

I solved the instances using "z3". I compiled this tool from source enabling architecture-dependant optimization, as the usage of CPU-specific set of instructions, and object code optimization. The version is 4.11.2 (64-bit).

As already mentioned, I performed a binary search over the domain of the board height, and I implemented several utils to check whether the solution obtained was valid, optimal, and computed before the expiration of a user-defined timeout. All the instances were run automatically (through a script I implemented) with a timeout of 300s, with and without the activation of symmetry breaking, to compare results.

Experimental Results

In 3.1 You can see the results for the SAT-based approach as far as the objective value is concerned, while in 3.1 and 3.2 you can see the time needed to solve the instances. As it can be noticed, symmetry breaking does not play a big role in this case, even slowing down the solving process in some cases.

Id	No Rotation w/out SB	Rotation w/out SB	No Rotation w SB	Rotation w SB
ins-1	8	8	8	8
ins-2	9	9	9	9
ins-3	10	10	10	10
ins-4	11	11	11	11
ins-5	12	12	12	12
ins-6	13	13	13	13
ins-7	14	14	14	14
ins-8	15	15	15	15
ins-9	16	16	16	16
ins-10	17	17	17	17
ins-11	18	18	18	18
ins-12	19	19	19	19
ins-13	20	20	20	20

ins-14	21	21	21	21
ins-15	22	22	22	22
ins-16	23	23	23	23
ins-17	24	24	24	24
ins-18	25	25	25	25
ins-19	26	27	26	26
ins-20	27	27	27	27
ins-21	28	30	28	30
ins-22	29	29	29	30
ins-23	30	30	30	30
ins-24	31	31	31	31
ins-25	32	33	32	33
ins-26	33	33	33	33
ins-27	34	34	34	34
ins-28	35	35	35	35
ins-29	36	36	36	36
ins-30	38	41	37	41
ins-31	38	38	38	38
ins-32	40	43	40	43
ins-33	40	40	40	40
ins-34	40	41	40	41
ins-35	40	40	40	40
ins-36	40	40	40	40
ins-37	60	64	60	64
ins-38	62	62	62	62
ins-39	60	61	60	61
ins-40	96	N/A	92	N/A

Table 3.1: Minimal height found for which 2DSP was satisfiable.

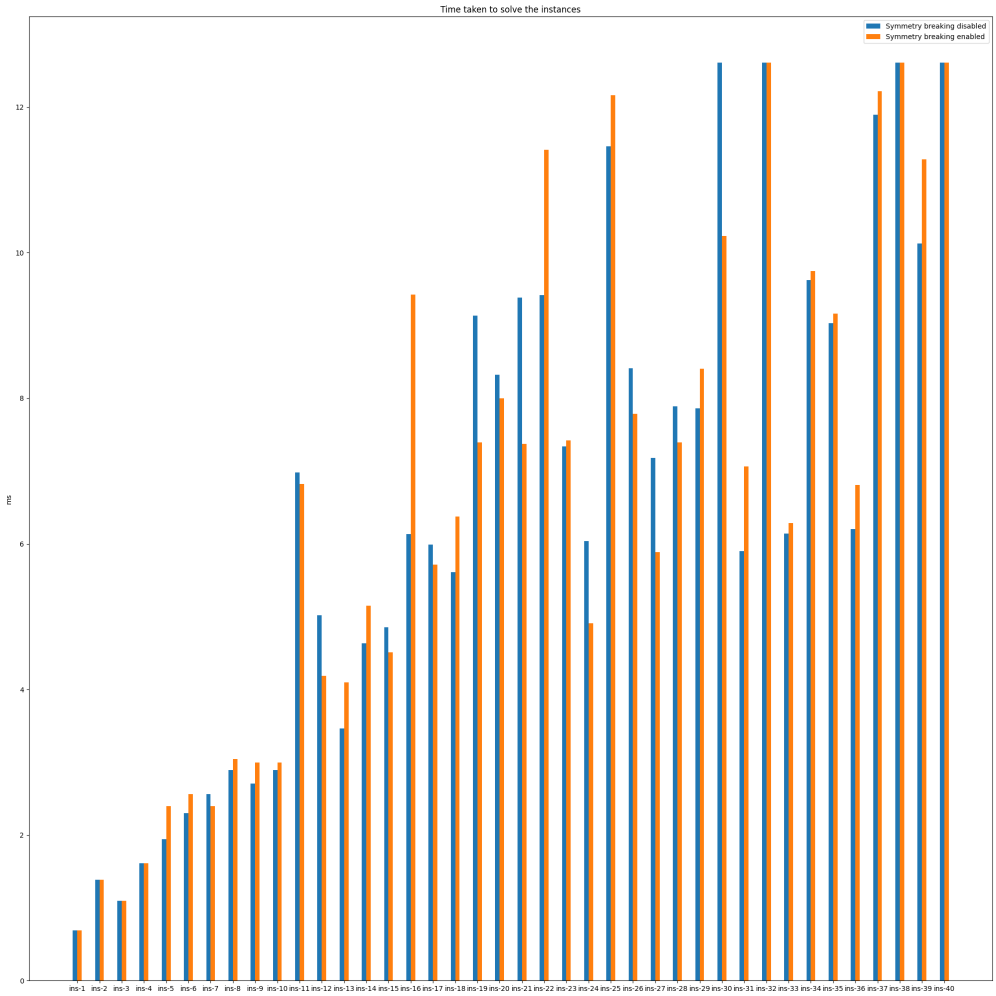


Figure 3.1: A comparison of the execution times for the standard version of the problem with and without symmetry breaking enabled. A logarithmic transformation has been applied to rescale the bars.

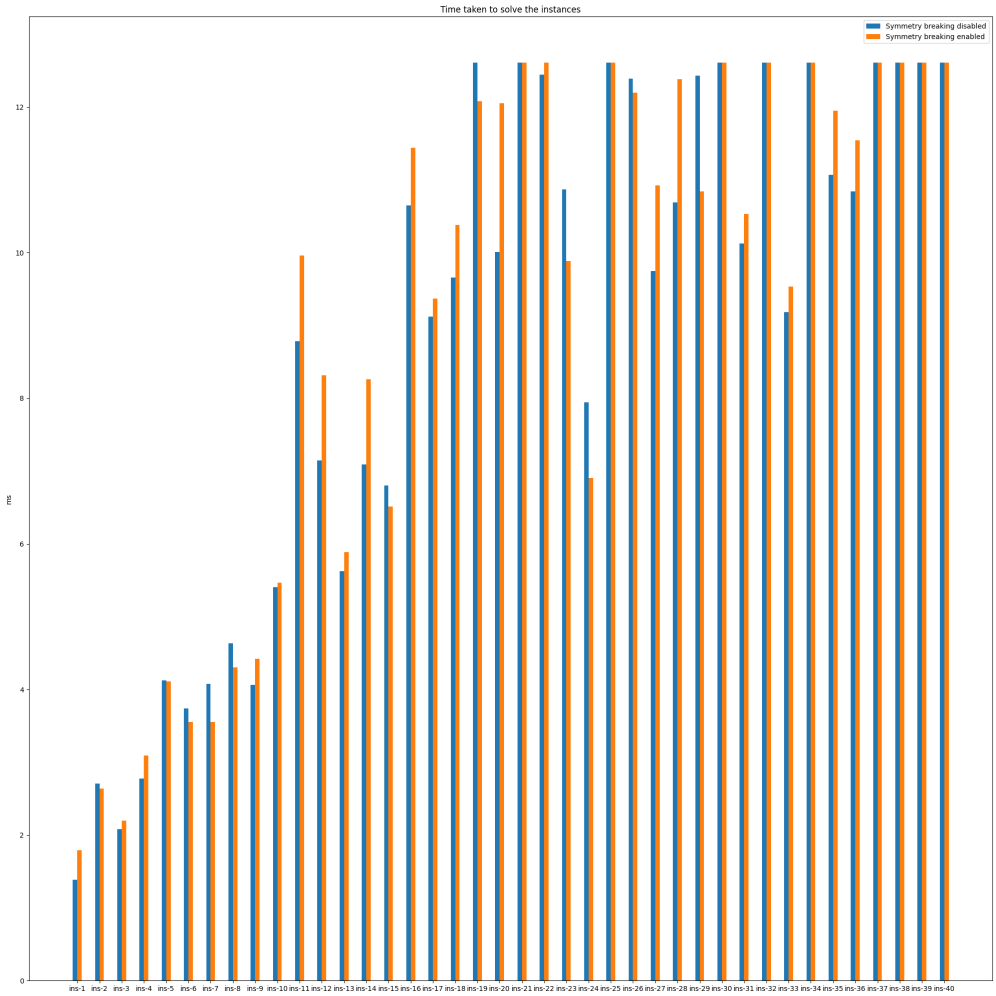


Figure 3.2: A comparison of the execution times for the rotation version of the problem with and without symmetry breaking enabled. A logarithmic transformation has been applied to rescale the bars.

Chapter 4

MIP Models - Positions and Covering Methodology

Adapting the work of Cid-Garcia and Rios-Solis in [4] to the 2DSP problem, I implemented a two-stage algorithm which employs a preprocessing step and an actual MIP solving step.

In particular, the P&C methodology uses a set-covering model to solve the decision version of the 2DBPP (two dimensional bin packing problem), that is, checking whether it is feasible to pack n items into K bins without overlapping, where each bin is an identical strip with capacity equal to the number of cells (i.e. the area of the strip) $W \times H$. The algorithm starts trying to prove feasibility for a $K = \frac{\sum_{i \in [1,N]} w_i * h_i}{W * H}$, and then (performing a linear search), if the problem is infeasible, it tries to prove feasibility for $K = H_{LB} + 1$, and so on, incrementing K by one at each step.

In the 2DSP case we only have one strip, and we want to pack the items into it without overlapping, so the problem is equivalent to the 2DBPP with $K = 1$.

The first stage of the methodology consists in generating a set of valid positions where each circuit could be placed inside the strip. Circuits are grouped by size

and demands, and the set of valid positions (which has pseudo-polynomial size, complessively $O(W^2H^2)$) is generated with the following procedure (valid positions for a single circuit with width w and height h in a strip of height H and width W):

Algorithm 4.1: Valid positions generation

```
1   input: int W, int H, int w, int h
2   output: list[list[int]]
3   begin
4       for i ← 0 to W − w
5           for j ← 0 to H − h
6               points ← []
7               for k ← 0 to h − 1
8                   s ← (j + k) ∗ W + i
9                   e ← s + w
10                  for l ← s to e − 1
11                      points ← points + [l]
12                  end
13              res ← res + [points]
14          end
15      end
16    return res
17  end
```

The result of this step (applied to each circuit) is mapped into a correspondence matrix C, of size $|J| \times (W \times H)$, where J is the set of valid positions for all circuits, and $W \times H$ is the number of cells in the strip, which are the "bins" in the paper. The matrix C is a binary matrix, where $C_{ij} = 1$ if the i-th valid position covers the j-th cell, and $C_{ij} = 0$ otherwise. In addition, we keep track of the subset of valid positions for each circuit i called $T(i) = \{j|\gamma(j) = i\}$ where function $\gamma(j)$ indicates if position j is a valid position for the circuit i, with $T(i) \subseteq J$, and compute upper bounds M_i for the number of times a circuit i can be placed in the strip, which is the number of valid positions for that circuit.

In addition to this, we also have the computed demands for each circuit i: d_i.

The second stage of this methodology is the covering step, which is the actual decision problem modeled as a MIP.

4.1 Decision Variables

Bearing in mind that the precomputed parameters are the following:

- C: the correspondence matrix,
- $T(i)$: the set of valid positions for circuit i.
- d_i: the demand for circuit i,
- M_i: the upper bound for the number of times circuit i can be placed in the strip.

The decision variables are the following:

- $x_{i,j}$: binary variable indicating that circuit i is placed in the valid position j, which itself will cover a certain number points as described earlier.

4.2 Constraints

The basic constraints for the problem are the following:

- $\forall_{p \in P}(\sum_{i \in [1,N]} \sum_{j \in T(i)} x_{i,j} * c_{j,p} \leq 1)$: each point in the strip is covered by only one circuit or none,
- $\forall_{i \in [1,N]}(\sum j \in T(i) x_{i,j} \geq d_i)$: the demand for each circuit i is satisfied,
- $\forall_{i \in [1,N]} \forall_{j \in T(i)} x_{i,j} \in 0, 1$: the variables are binary.

Implied Constraints

What in the paper are called "valid inequalities", which "make the polyheadron of the solution space to be closer to the convex hull of the discreate feasible solutions of the problem", can be considered as implied constraints. In particular, the following inequalities can always be enforced:

- $\forall i \in [1, N](\sum_{j \in T(i)} x_{i,j} \geq 1)$, which enforces that each circuit is placed at least once in the strip.
- $\forall_{i \in [1,N]}(\sum_{j \in T(i)} x_{i,j} \leq M_i)$, which enorces that each circuit is placed at most M_i times in the strip,
- $\sum_{i \in [1,N]} \sum_{j \in T(i)} \sum_{p \in P} c_{j,p} * x_{i,j} \leq W \times H$, which enforces that the area of the bin is not exceeded.

Symmetry Breaking Constraints

For this formulation of the 2DSP I did not consider any symmetry breaking constraint.

4.3 Objective Function

The problem is modeled as a series of satisfaction ones, so no objective function is enforced when the covering problem is modeled.

4.4 Rotation

To handle rotation of the circuits, we must:

- Extend the first stage to compute also the valid positios for the rotated version of the circuits. The correspondece matrix will than have new columns $\bar{c}_{j,p}$, such that $\bar{T}(i) = \{j|\gamma(j) = i\}$ is the subset of the new valid positions j for the rotated version of the circuit i.

- Add the new decision variables $y_{i,j}$, which are binary and indicate that the rotated version of the circuit i is placed in the valid position $j \in \bar{T}(i)$.

- Modify the constraints of the model as it follows:

  - $\forall_{p \in P}(\sum_{i \in [1,N]} \sum_{j \in T(i)} x_{i,j} * c_{j,p} + \sum_{i \in [1,N]} \sum_{j \in \bar{T}(i)} y_{i,j} * \bar{c}_{j,p} \leq 1)$: each point in the strip is covered by only one circuit, either in the original or rotated version, or none,

  - $\forall_{i \in [1,N]}(\sum_{j \in T(i)} x_{i,j} + \sum_{j \in \bar{T}(i)} y_{i,j} \geq d_i)$: the demand for each circuit i is satisfied, using either the original or the rotated version of the circuit,

  - $\forall_{i \in [1,N]} \forall_{j \in T(i)} x_{i,j} \in 0, 1$ and $\forall_{i \in [1,N]} \forall_{j \in \bar{T}(i)} y_{i,j} \in 0, 1$: the variables are binary,

  - $\forall_{i \in [1,N]}(\sum_{j \in T(i)} x_{i,j} + \sum_{j \in \bar{T}(i)} y_{i,j} \geq 1)$, which enforces that each circuit is placed at least once in the strip, either in the original or rotated version,

  - $\forall_{i \in [1,N]}(\sum_{j \in T(i)} x_{i,j} + \sum_{j \in \bar{T}(i)} y_{i,j} \leq M_i)$, which enforces that each circuit is placed at most M_i times in the strip, either in the original or rotated version,

  - $\sum_{i \in [1,N]} \sum_{j \in T(i)} \sum_{p \in P} c_{j,p} * x_{i,j} + \sum_{i \in [1,N]} \sum_{j \in \bar{T}(i)} \sum_{p \in P} \bar{c}_{j,p} * y_{i,j} \leq W \times H$, which enforces that the area of the strip is not exceeded.

4.5 Validation

Experimental Design

I solved the version of the problem with no rotation allowed using both Gurobi (version 10.0.0 build v10.0.0rc2 (linux64)), and Cplex (version 22.1.0.0), to compare their performances. A timeout of 5 minutes was set for each instance. For the version of the problem with rotation allowed, I used only Gurobi. As already mentioned, to obtain the highest flexibility in the choice of the solver, I used the PulP library. As for all the other models, I implemented a script to automatically solve the instances, and to collect the results.

Experimental Results

Table 4.1 shows the minimal height for which it was possible to prove feasibility for the different configurations. In this case, whenever a value is reported, it is optimal, because the solving process starts with $H = H_{LB}$. Whenever the solver was not able to find a solution within the time limit, the value N/A is reported. While in figure 4.1 the time needed to solve the instances is reported.

Id	No-Rotation with Gurobi	No-Rotation with Cplex	Rotation with Gurobi
ins-1	8	8	8
ins-2	9	9	9
ins-3	10	10	10
ins-4	11	11	N/A
ins-5	12	12	N/A
ins-6	13	13	N/A
ins-7	14	14	N/A
ins-8	15	15	N/A

ins-9	16	16	N/A
ins-10	17	17	17
ins-11	18	18	18
ins-12	19	19	19
ins-13	20	20	20
ins-14	21	21	21
ins-15	22	22	22
ins-16	23	23	23
ins-17	24	24	24
ins-18	25	25	25
ins-19	26	26	26
ins-20	27	27	27
ins-21	28	28	28
ins-22	29	29	29
ins-23	30	30	30
ins-24	31	31	31
ins-25	32	32	32
ins-26	33	33	33
ins-27	34	34	34
ins-28	35	35	35
ins-29	36	36	36
ins-30	37	37	N/A
ins-31	38	38	38
ins-32	39	39	N/A
ins-33	40	40	40
ins-34	40	40	N/A
ins-35	40	40	N/A

ins-36	40	40	N/A
ins-37	N/A	N/A	N/A
ins-38	N/A	N/A	N/A
ins-39	N/A	N/A	N/A
ins-40	N/A	N/A	N/A

Table 4.1: Minimal height found for which 2DSP was satisfiable

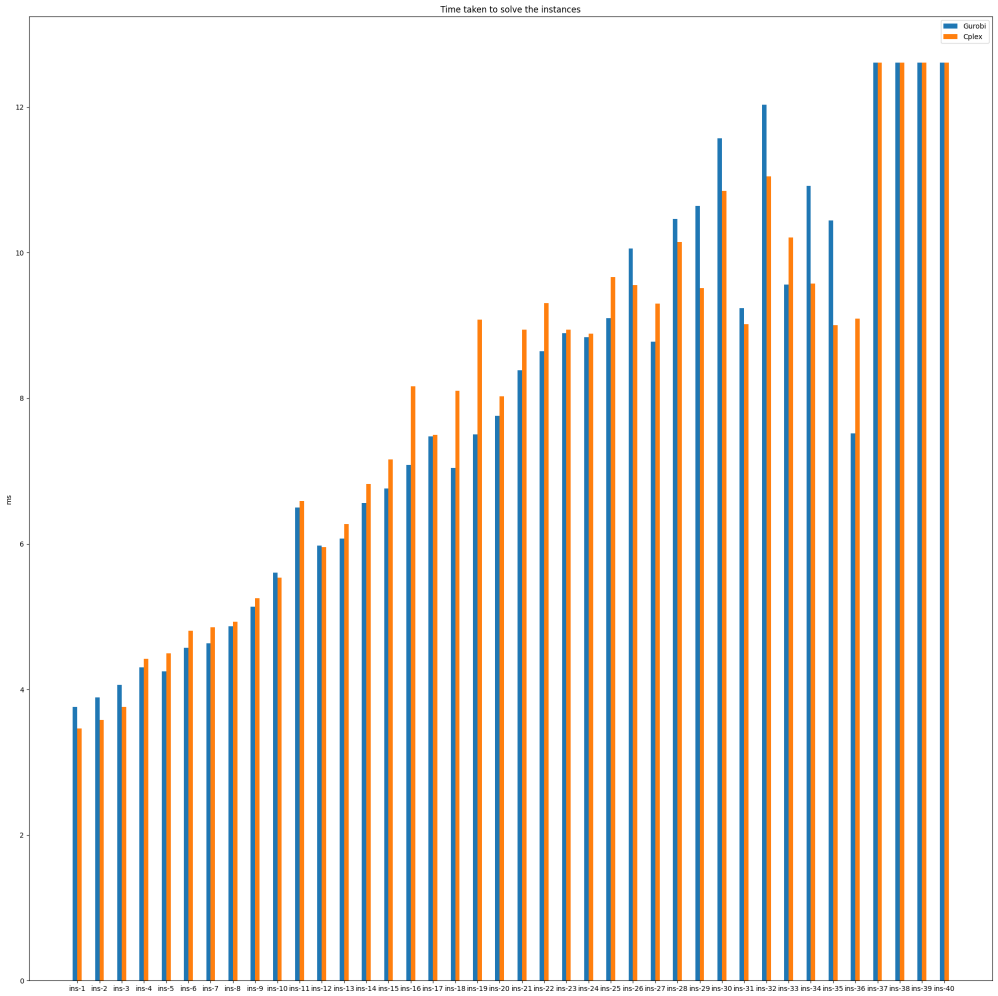


Figure 4.1: A comparison of the execution times of Gurobi and Cplex for the PC instances with no rotation allowed. A logarithmic transformation has been applied to rescale the bars.

Chapter 5

MIP Models - GDP to MIP with Big-M Reformulation

I decided to implement also the Big-M reformulation of the models described in [5], but it turned out that the variants called S1 and S2 (with additional symmetry constraints introduced by the authors) actually performed worse than the simpler model SG (originally discussed in [9]), so I decided to not include them in the final report.

The interesting idea to me behind the work of Trespalacios and Grossmann is that of reformulating GDP (Generalized Disjunctive Programming) [10] into mip, and to evaluate the impact on performances of different reformulations. In particular, in [5], they evaluate the impact of the Big-M [11], the improved Big-M [12], and the HR (Hull Reformulation) [13] reformulations.

In general a GDP can be represented as:

$$\min Z = f(x) + \sum_{k \in K} c_k \tag{5.1}$$

$$\text{s.t.} g(x) \leq 0 \tag{5.2}$$

$$\bigvee_{i \in D_k} \begin{bmatrix} Y_{ik} \\ r_{ik}(x) \leq 0 \\ c_k = \gamma_{ik} \end{bmatrix} \tag{5.3}$$

$$\Omega(Y) = True \tag{5.4}$$

$$x^{lo} \leq x \leq x^{up} \tag{5.5}$$

$$x \in \mathbb{R}^n,\ c_k \in \mathbb{R},\ Y_{ik} \in \{True, False\} \tag{5.6}$$

In this representation 5.1 is the objective function to be minimized, which takes into account additional costs associated to the disjunctive constraints 5.3 (the c_k variables). 5.2 is a set of global constraints, 5.3 is the set of disjunctive constraints, each of which is enforced when Y_{ik} is True, 5.4 is a set of logical properties that the Y_{ik} variables must satisfy, and 5.5 is the bound for x.

To reformulate a GDP problem as a MIP one with the Big-M reformulation, for each disjunction we must introduce a binary variable y_{ik}, which has a one-to-one correspondence with Y_{ik}, and find a value (often reffered to as the Big-M indeed), which is an absolute upper bound for the disjunction and which makes the disjunction redundant (i.e. inactive) when the corresponding Y_{ik} is False. For example if my original constraint was $Y_1 \implies x \leq 15$, and I had $0 \leq x \leq 30$ I would introduce a new integer variable $y_i \in \{0, 1\}$, and I would reformulate the constraint as: $x \leq (1 - y_i) \cdot M + 15$, where M would be the minimum value necessary to make the constraint redundant when $y_i = 0$, that is, in this case (since the upper bound for x is 30) $M = 15$.

The GDP formulation of the 2DSP I implemented is the following:

$$\min H \text{ s.t. } \forall_{i\in[1,N]} H \geq x_i + w_i \tag{5.7}$$

$$\forall_{i,j\in[1,N] \text{ s.t. } i<j}\left(\begin{bmatrix} Z^1_{ij} \\ x_i + w_i \leq x_j \end{bmatrix} \vee \begin{bmatrix} Z^1_{ji} \\ x_j + w_j \leq x_i \end{bmatrix} \vee \begin{bmatrix} Z^2_{ij} \\ y_i + h_i \geq y_j \end{bmatrix} \vee \begin{bmatrix} Z^2_{ji} \\ y_j + h_j \geq y_i \end{bmatrix}\right) \tag{5.8}$$

$$\forall_{i,j\in[1,N] \text{ s.t. } i<j}(Z^1_{ij} \veebar Z^1_{ji} \veebar Z^2_{ij} \veebar Z^2_{ji}) \tag{5.9}$$

$$\forall_{i\in[1,N]}(0 \leq y_i \leq H_{UB} - h_i) \tag{5.10}$$

$$\forall_{i\in[1,N]}(0 \leq x_i \leq W) \tag{5.11}$$

$$\forall i, j \in [1, N] \text{ s.t. } i \neq j (Z^1_{ij}, Z^2_{ij} \in \{True, False\}) \tag{5.12}$$

In the original paper, the strip grew in the horizontal direction, and the (x, y) coordinates of a circuit were that of the left-top corner, but I decided to stick with the format of the other models and modify it.

5.1 Decision Variables

The decision variables of the MIP model are the following:

- x_i is the horizontal coordinate of the of the bottom-left corner of the i-th circuit
- y_i is the vertical coordinate of the bottom-left corner of the i-th circuit
- z^1_{ij} is a binary variable that is True if the i-th circuit is placed to the left of the j-th circuit
- z^2_{ij} is a binary variable that is True if the i-th circuit is placed above the j-th circuit
- H is the height of the strip

5.2 Constraints

When turning the disjunctions of the GDP formulation 5.8 into an equivalent set of constraints for the MIP problem, a Big-M value is needed. That set of disjunction, together with the exclusive or 5.9, is actually enforcing the non-overlapping constraints 1.1. A circuit can be either on the left, on the right, above or below another circuit.

For example we know that when the i-th circuit is on the left of the j-th circuit, $z^1_{ij} = True$, and therefore the constraint enforced is $x_i + w_i \leq x_j$. To make this constraint redundant when $z^1_{ij} = False$, we can use the Big-M value W, and translate the constraint in the MIP formulation as $x_i + w_i \leq (1 - z^1_{ij}) \cdot W + x_j$. Indeed it is always true that $x_i + w_i \leq W$.

In practice, the MIP constraints are the following:

- $0 \leq x_i + w_i \leq W \wedge 0 \leq y_i + h_i \leq H$ for all $i \in [1, N]$, which enforces the domain of the variables x and y,
- $z^1_{ij} \in \{0, 1\}$ and $z^2_{ij} \in \{0, 1\}$ for all $i, j \in [1, N]$ s.t. $i \neq j$, which enforces the domain of the variables z,
- $$\begin{aligned} x_i + w_i &\leq (1 - z^1_{ij}) \cdot W + x_j \\ x_j + w_j &\leq (1 - z^1_{ji}) \cdot W + x_i \\ y_i &\geq y_j + h_j - (1 - z^2_{ij}) \cdot H_{UB} \\ y_j &\geq y_i + h_i - (1 - z^2_{ji}) \cdot H_{UB} \end{aligned}$$

  for all $i, j \in [1, N]$ s.t. $i < j$, which enforces the non-overlapping constraints,
- $z^1_{ij} + z^1_{ji} + z^2_{ij} + z^2_{ji} = 1$ for all $i, j \in [1, N]$ s.t. $i < j$, which enforces the exclusive or constraint.

Implied Constraints

I added the large circuits implied constraints 2.2 to the MIP formulation, which were not present in the original paper:

$$\forall_{i,j\in[1,N] \text{ s.t. } i<j \text{ and } w_i+w_j>W}(z^1_{ij} = 0 \wedge z^1_{ji} = 0) \tag{5.13}$$

$$\forall_{i,j\in[1,N] \text{ s.t. } i<j \text{ and } h_i+h_j>H_{UB}}(z^2_{ij} = 0 \wedge z^2_{ji} = 0) \tag{5.14}$$

Symmetry Breaking Constraints

I also added two symmetry breaking constraints to the model: one fixing the spatial relation of same-sized circuits, and one fixing the spatial relation of the two biggest circuits. Lexicographic ordering between two pairs of two variables can be easily imposed on a MIP setting, indeed if (x, y) and $(\hat{x}, \hat{y})$ are the pairs of two variables, it suffices to impose that $y * ub(y) + x \leq \hat{y} * ub(\hat{y}) + \hat{x}$, where $ub(y)$ is the upper bound of the variable y. The same-sized circuits symmetry breaking constraint is the following:

$$\forall_{i,j\in[1,N] \text{ s.t. } i<j \text{ and } w_i=w_j \text{ and } h_i=h_j}(y_i * H_{UB} + x_i \leq y_j * H_{UB} + x_j)$$

The two biggest circuits symmetry breaking constraint is the following:

$$y_{order_0} * H_{UB} + x_{order_0} \leq y_{order_1} * H_{UB} + x_{order_1},$$

where *order* is a permutation of the indices of the circuits sorted by decreasing circuit area.

5.3 Objective Function

The objective function is simply H, the height of the board.

5.4 Rotation

To handle rotation I simply introduced the following new decision variables, and adapted the constraints accordingly:

- $r_i \in \{0, 1\}$, which is equal to 1 if the circuit i is rotated, and 0 otherwise,
- $rw_i = r_i \cdot h_i + (1 - r_i) \cdot w_i$, which is the rotated width of the circuit i,
- $rh_i = r_i \cdot w_i + (1 - r_i) \cdot h_i$, which is the rotated height of the circuit i.

The only non-trivial difference in the constraints was to modify the lexicographic orderings using $max(H_{UB}, W)$ instead of H_{UB}, since now the domain of rw and rh is $[0, max(H_{UB}, W)]$ instead of $[0, W - w_i]$ and $[0, H_{UB} - h_i]$ respectively.

Experimental Design

I run the experiments using the same setting as the one for the positioning and covering methodology. This time, I compared the results of Gurobi and CPLEX for the case with rotation.

Experimental Results

In table 5.1 it can be noticed that enabling rotation does not impact performances for this model. I think the reason why this happens is that by activating the warm start and giving the solver the initial configuration $r_i = 0$ for all i, the solver is able to find the optimal solution without having to rotate any circuit, but trying to keep r_i constant from the beginning thanks to some clever internal strategy. As it can be seen in 5.1, symmetry breaking does not help with this model.

Id	Gurobi	Gurobi SB	Gurobi w.r.	Gurobi w.r. SB	CPLEX w.r. SB
ins-1	8	8	8	8	8
ins-2	9	9	9	9	9
ins-3	10	10	10	10	10
ins-4	11	11	11	11	11
ins-5	12	12	12	12	12
ins-6	13	13	13	13	13
ins-7	14	14	14	14	14
ins-8	15	15	15	15	15
ins-9	16	16	16	16	16
ins-10	17	17	17	17	17
ins-11	18	18	18	18	18
ins-12	19	19	19	19	19
ins-13	20	20	20	20	20
ins-14	21	21	21	21	21
ins-15	22	22	22	22	22
ins-16	23	23	23	23	23
ins-17	24	24	24	24	24
ins-18	25	25	25	25	25
ins-19	26	26	27	26	26
ins-20	27	27	27	28	28
ins-21	28	29	29	29	29
ins-22	29	29	29	30	30
ins-23	30	30	30	30	30
ins-24	31	31	31	31	31
ins-25	33	33	33	33	33
ins-26	33	34	33	33	33

ins-27	34	34	35	34	34
ins-28	35	35	35	35	35
ins-29	36	36	36	36	36
ins-30	39	38	38	38	38
ins-31	38	38	38	38	38
ins-32	41	41	41	40	40
ins-33	40	40	40	40	40
ins-34	40	40	41	41	41
ins-35	41	40	41	40	40
ins-36	40	40	40	41	41
ins-37	62	62	62	61	61
ins-38	62	62	61	61	61
ins-39	62	62	61	61	61
ins-40	N/A	101	101	N/A	N/A

Table 5.1: Best objective value for the SG MIP model with different configurations and solvers.

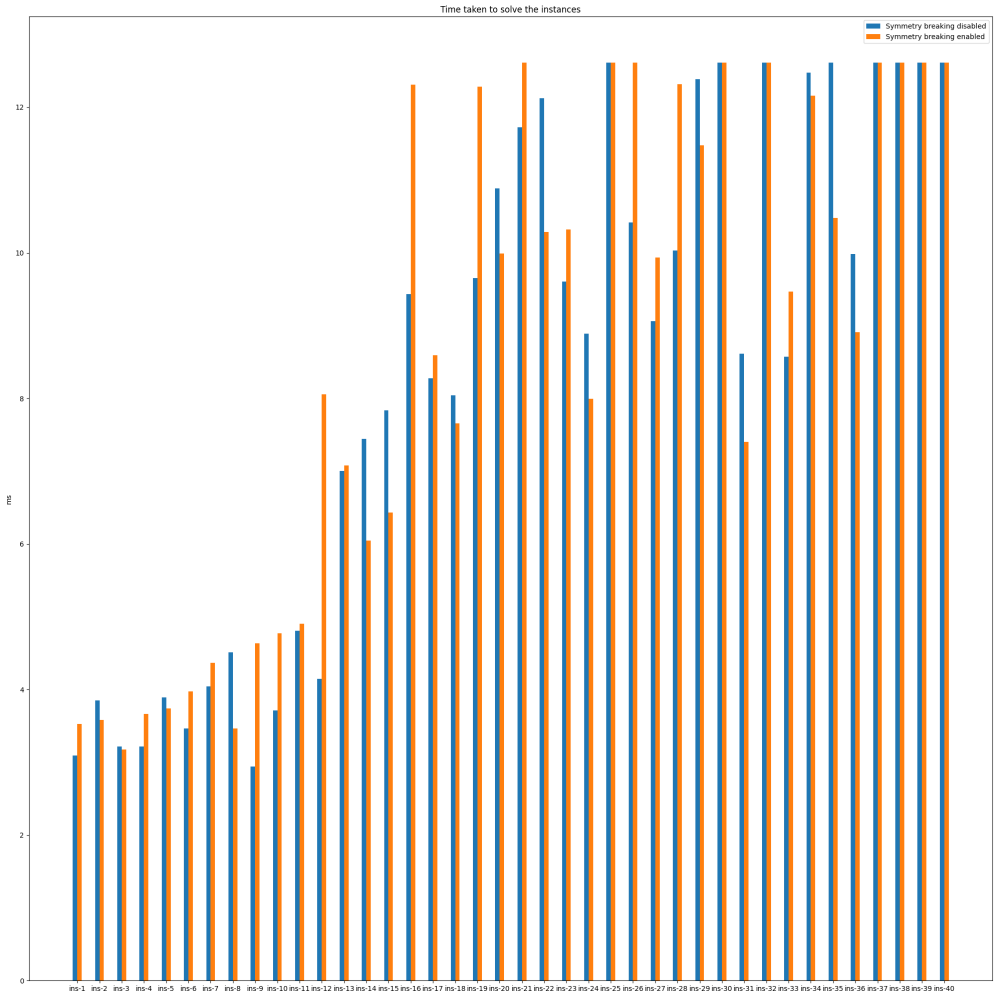


Figure 5.1: A comparison of the execution times for the standard version of the problem with and without symmetry breaking enabled, and using the Gurobi solver. A logarithmic transformation has been applied to rescale the bars.

Chapter 6

SMT Model

SMT (Satisfiability Modulo Theories) overcome the limited expressiveness of SAT by allowing us to model constraint satisfaction problems using first order logic (FOL) and restricting the interpretation of the formulas to a given background theory. Restricting to a single background theory allows for more efficient solving through the use of specialized decision procedures. We know that first order logic is semi-decidable, and undecidable with some theories as linear integer arithmetic, but we can restrict to decidable fragments of a theory, like quantified formulas with finite domains. Different SMT solvers support different theories, as integer aritmetic, bitvectors, arrays, and real numbers.

SMT (Satisfiability Modulo Theories) extend boolean satisfiability with more complex domains, such as arithmetic. In recent years, SMT solvers have gained increasing attention due to technological advancements and industrial applications, leading to progress in constraint-satisfaction problems that can be solved.

To model the 2DSP problem I used SMT-LIB, which provides a standardized format for defining theories and expressing problems, allowing for interoperability between different SMT solvers.

In particular, I translated the same model used in the CP formulation, with

small differences, to SMT-LIB, and used Z3 and CVC5 as solvers. I had to design a way to retrieve the solutions from the solvers, and to do so I employed some mechanism of interprocess communication (IPC) between the solver and the Python script.

In addition, I implemented binary search over the domain of the height variable, to find the minimum height which ensured satisfiability.

6.1 Decision Variables

The decision variables for the problem are:

- x_i: the horizontal coordinate of the i-th circuit,
- y_i: the vertical coordinate of the i-th circuit,
- h: the height of the board.

To decision variables are declared as it follows (x_i, and y_i where declared for each circuit, with $i \in [1, n]$):

```
(declare-fun x_i (Int) Int)
(declare-fun y_i (Int) Int)
(declare-fun h () Int)
```

6.2 Constraints

The first basic constraints are the ones that define the domain of the variables:

- $0 \leq x_i \leq W - w_i$,
- $0 \leq y_i \leq H - h_i$,

- $H_{LB} \leq h \leq H_{UB}$.

Those can be translated into SMT-LIB as follows:

```
(assert (and (>= (x_i 0) (<= x_i (- W w_i))))
(assert (and (>= y_i 0) (<= y_i (- H h_i))))
(assert (and (>= h H_LB) (<= h H_UB)))
```

Then we have the enforcement of the actual height of the board which can be translated into SMT-LIB as follows:

```
(assert (<= (+ y_i h_i) h))
```

The last basic constraint is the one that enforces the non-overlapping of the circuits 1.1 which can be translated into SMT-LIB as follows:

```
(assert (or (<= (+ x_i w_i) x_j)
            (<= (+ x_j w_j) x_i)
            (<= (+ y_i h_i) y_j)
            (<= (+ y_j h_j) y_i)))
```

Implied Constraints

As implied constraints, we have the cumulative constraints on x and y 2.2, and the ones on large circuits 2.2. The translation of the cumulative constraints is a bit complex, since it requires iterating through $t \in [early, late]$, where $early = 0$, and $late = W + max_{i \in [1,N]} w_i$ for the constraints on the x coordinate, and $late = H + max_{i \in [1,N]} h_i$ for the y coordinate. The cumulative constraints on x_i can be translated into SMT-LIB as follows:

```
(assert (>= h
  (+ (* ite ((and (<= x_i early) (< early (+ x_i w_i))) 1 0) h_i)
     (* ite ((and (<= x_i early+1) (< early+1 (+ x_i w_i))) 1 0) h_i)
```

```
[...]
(*ite((and (<= x_i late) (< late (+ x_i w_i))) 1 0) h_i))))
```

The cumulative constraints on y_i can be translated into SMT-LIB as follows:

```
(assert (>= h
  (+ (* ite((and (<= y_i early) (< early (+ y_i h_i))) 1 0) w_i)
     (* ite((and (<= y_i early+1) (< early+1 (+ y_i h_i))) 1 0) w_i)
     [...]
     (*ite((and (<= y_i late) (< late (+ y_i h_i))) 1 0) w_i))))
```

For the implied constraint on large circuits, for all the circuits $i, j \in [1, N]$, where $i < j$, and $w_i + w_j > W$, we have:

```
(assert (or (<= (+ y_i h_i) y_j) (>= y_i (+ y_j h_j))))
```

While, for all the circuits $i, j \in [1, N]$, where $i < j$, and $h_i + h_j > H_{UB}$, we have:

```
(assert (=> (>= h_i+h_j h) (or (<= (+ x_i w_i) x_j) (>= x_i (+ x_j w_j)))))
```

Symmetry Breaking Constraints

As symmetry breaking constraints, I used the lexicographic ordering on the coordinates of the circuits with respect to the reflected ones 2.2, and the fixation of the spatial relation between two circuits 2.2. For the lexicographic ordering I used the recursive OR encoding 2.2 2.2, as for CP.

6.3 Objective Function

The objective function is, as usual, the height of the board.

6.4 Rotation

To handle the rotation of the circuits, I had to introduce, as done for the CP part, new decision variables r_i, rw_i and rh_i, which are the rotation state, the rotated width and the rotated height of circuit i. The constraints discussed so far can be easily adapted for this version of the problem. An additional constraint which is needed is the one which actually define the value of rw_i and rh_i, which can be translated into SMT-LIB as follows:

```
(assert (ite r{} (= rw_i w_i) (= rw_i h_i)))
(assert (ite r{} (= rh_i h_i) (= rh_i w_i)))
```

In addition, while for solving the standard version of the problem it was enough to use the linear integer arithmetic theory, for the rotation version I had to use the non-linear integer arithmetic theory, since the rotated width and height of the circuits are not linear functions of the width and height of the circuits.

We can ask the solver to use the quantified non-linear integer arithmetic theory by adding the following line to the SMT-LIB file:

```
(set-logic QF_NIA)
```

6.5 Validation

Experimental Design

As for the other models I implemented a script for the automated solving of the instances, and I used z3 and cvc5 as solvers. I compiled from source both the solvers to enable architecture-dependant code optimization, and the versions were: Z3 version 4.12.2 - 64 bit and cvc5 version 1.0.3-dev.139.21e7e341c [git 21e7e341c on branch HEAD] compiled with GCC version 11.2.1 20220127 (Red Hat 11.2.1-9).

I compared performances on the standard version of the problem and the one with rotation allowed, using z3 and cvc5 for the former and only z3 for the latter. I did not run other experiments because I had already shown with other models how symmetry breaking could improve performances, and it was too costly in terms of time to solve instances for each different configuration.

Experimental Results

In table**??** I report the results of the experiments I ran, while on 6.1 and **??** you can have a look at the time needed to solve the instances for the standard version and the version with rotation of the problem respectively.

Id	No-Rotation with Z3	No-Rotation with CVC5	Rotation with Z3
ins-1	8	8	8
ins-2	9	9	9
ins-3	10	10	10
ins-4	11	11	11
ins-5	12	12	12
ins-6	13	13	13
ins-7	14	14	14
ins-8	15	15	15
ins-9	16	16	16
ins-10	17	17	17
ins-11	18	18	18
ins-12	19	19	19
ins-13	20	20	20
ins-14	21	21	21
ins-15	22	22	22

ins-16	23	23	23
ins-17	24	24	24
ins-18	25	25	26
ins-19	26	26	27
ins-20	27	27	27
ins-21	28	28	30
ins-22	29	30	30
ins-23	30	30	31
ins-24	31	31	31
ins-25	32	33	33
ins-26	33	33	34
ins-27	34	34	35
ins-28	35	35	36
ins-29	36	36	37
ins-30	38	38	38
ins-31	38	38	38
ins-32	43	43	40
ins-33	40	40	40
ins-34	41	41	40
ins-35	40	40	40
ins-36	40	40	40
ins-37	64	64	61
ins-38	62	62	62
ins-39	61	61	61
ins-40	N/A	N/A	N/A

Table 6.1: Best objective value obtained

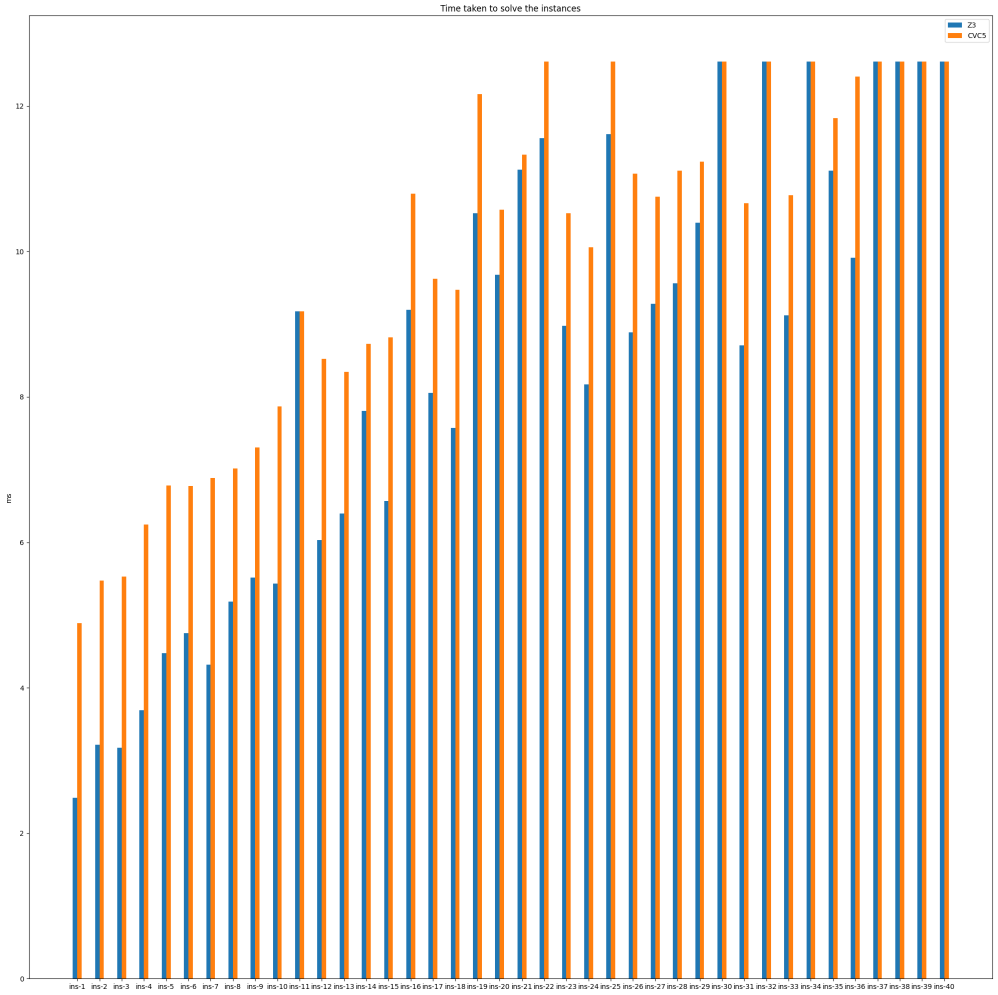


Figure 6.1: A comparison of the execution times of Z3 and CVC5 for the instances with no rotation allowed and symetry breaking enabled. A logarithmic transformation has been applied to rescale the bars.

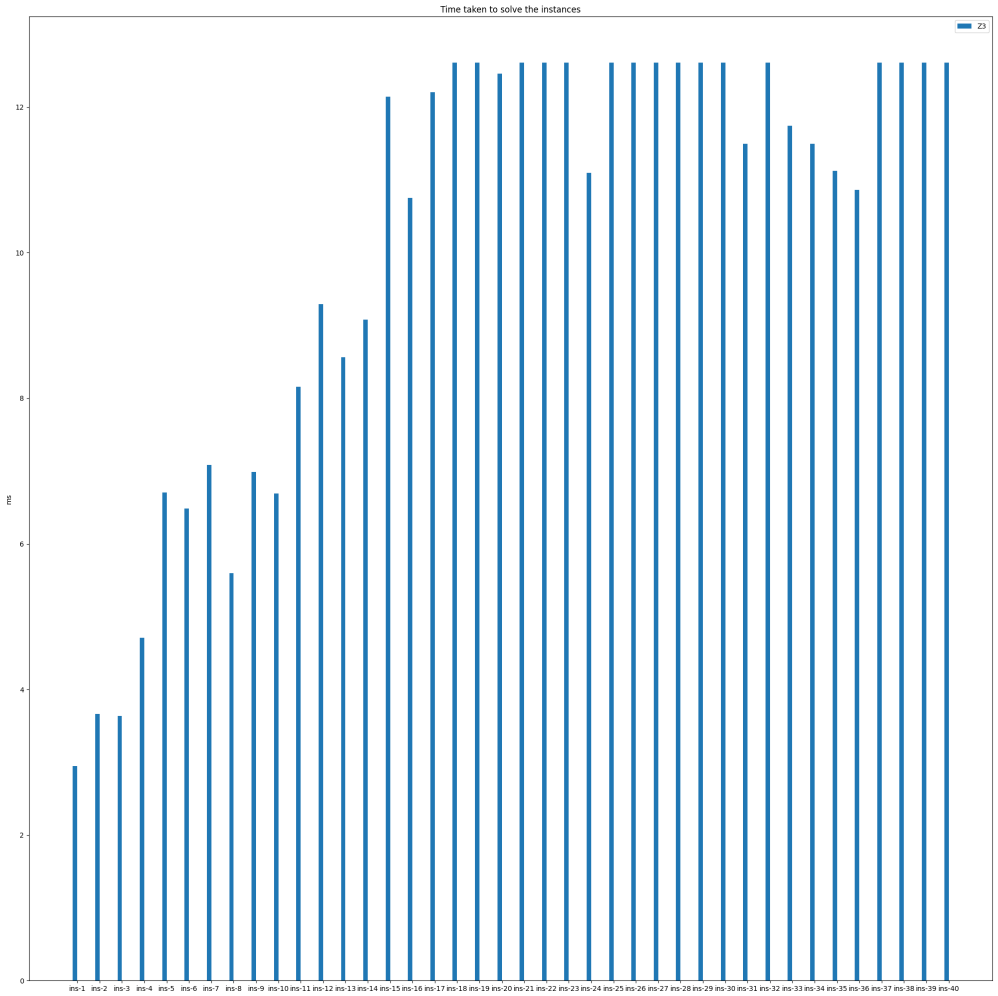


Figure 6.2: The execution times form the instances with rotation allowed and symmetry breaking enabled (Z3 solver). A logarithmic transformation has been applied to rescale the bars.

Chapter 7

Conclusion

Thanks to this project I had the opportunity to gain confidence with many tools in operations research. I came to the conclusion that the CP methodology is the best promising for this problem, together with the SAT and the SMT.

I obtained the worst results with the MIP for which, I think, it's most difficult to efficently reformulate the 2DSP problem. MIP does generally perform poor indeed on combinatorial problems that have a large number of symmetries or require search in large discrete spaces, as it is the case for the 2DSP.

In table 7.1 you can see a summary with the best objective value for each technology.

Id	CP	SAT	MIP	SMT
ins-1	8	8	8	8
ins-2	9	9	9	9
ins-3	10	10	10	10
ins-4	11	11	11	11
ins-5	12	12	12	12
ins-6	13	13	13	13

ins-7	14	14	14	14
ins-8	15	15	15	15
ins-9	16	16	16	16
ins-10	17	17	17	17
ins-11	18	18	18	18
ins-12	19	19	19	19
ins-13	20	20	20	20
ins-14	21	21	21	21
ins-15	22	22	22	22
ins-16	23	23	23	23
ins-17	24	24	24	24
ins-18	25	25	25	25
ins-19	26	26	26	26
ins-20	27	27	27	27
ins-21	28	28	28	28
ins-22	29	29	29	29
ins-23	30	30	30	30
ins-24	31	31	31	31
ins-25	32	32	32	32
ins-26	33	33	33	33
ins-27	34	34	34	34
ins-28	35	35	35	35
ins-29	36	36	36	36
ins-30	37	37	37	38
ins-31	38	38	38	38
ins-32	39	40	39	43
ins-33	40	40	40	40

ins-34	40	40	40	41
ins-35	40	40	40	40
ins-36	40	40	40	40
ins-37	60	60	N/A	64
ins-38	60	62	N/A	62
ins-39	60	60	N/A	61
ins-40	92	92	N/A	N/A

Table 7.1: Best objective value for each technology.

Bibliography

[1] Silvano Martello and Daniele Vigo. Exact solution of the two-dimensional finite bin packing problem. *Management Science*, 44(3):388–399, 1998.

[2] Daniele Vigo Andrea Lodi, Silvano Martello. Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. *Journal on Computing*, 11:329–431, 1999.

[3] Takehide Soh, Katsumi Inoue, Naoyuki Tamura, Mutsunori Banbara, and Hidetomo Nabeshima. A sat-based method for solving the two-dimensional strip packing problem. *Fundam. Inform.*, 102:467–487, 01 2010.

[4] Nestor M Cid-Garcia and Yasmin A Rios-Solis. Positions and covering: A two-stage methodology to obtain optimal solutions for the 2d-bin packing problem. *PLoS One*, 15(4):e0229358, April 2020.

[5] Francisco Trespalacios and Ignacio E. Grossmann. Symmetry breaking for generalized disjunctive programming formulation of the strip packing problem. *Annals of Operations Research*, 258(2):747–759, Nov 2017.

[6] Global constraints in the minizinc library. `https://www.minizinc.org/doc-2.5.3/en/lib-globals.html`.

[7] Ian Philip Gent, P Prosser, and BM Smith. A 0/1 encoding of the gaclex constraint for pairs of vectors. 2002. Proc.ECAI 2002 Workshop on Modelling and Solving Problems with Constraints.

[8] Naoyuki Tamura, Akiko Taga, Satoshi Kitagawa, and Mutsunori Banbara. Compiling finite linear csp into sat. volume 14, 09 2006.

[9] Nicolas Sawaya and Ignacio Grossmann. A cutting plane method for solving linear generalized disjunctive programming problems. *Computers & Chemical Engineering*, 29:1891–1913, 08 2005.

[10] Ignacio E. Grossmann and Juan P. Ruiz. Generalized Disjunctive Programming: A Framework for Formulation and Alternative Algorithms for MINLP Optimization. 1 2009.

[11] George L. Nemhauser and Laurence A. Wolsey. Integer and combinatorial optimization. In *Wiley interscience series in discrete mathematics and optimization*, 1988.

[12] Francisco Trespalacios and Ignacio E. Grossmann. Improved big-m reformulation for generalized disjunctive programs. *Computers & Chemical Engineering*, 76:98–103, 2015.

[13] Sangbum Lee and Ignacio E. Grossmann. New algorithms for nonlinear generalized disjunctive programming. *Computers & Chemical Engineering*, 24(9):2125–2141, 2000.