In this exercise I was to solve the Present Wrapping Problem (PWP) with SAT and SMT optimization.

SAT

Propositional satisfiability (SAT) optimization is based on the idea that one has a propositional formula composed by Boolean decision values that together form the constraints of the optimization problem.

Constraints

The constraints needed for the PWP were 1) A placement constraint to ensure the gridpoints of a present were next to each other and placed legally on the paper; 2) a contraint that ensured no present overlap; 3) a constraint that ensured that each present was placed at most once, and lastly; 4) a constraint that ensured that each present was placed at least once.

Implementation

To solve the problem I started by defining the grid on the paper where each gridpoint represent one length dimension for each present. The grid was thus a 3D-array named `grid` and each gridpoint was defined as $x - y - p$ where x defined the x-value, y defined the y-value, and p defined the present we were currently trying to place. This grid was to represent the Boolean values, e.g. if present 1 was to be placed in the gridpoint (0,0) then the value $0 - 0 - 1$ was to be set as true.

Then I wished to add the constraints to the solver. Thus, I formulated the placement constraint as

$$(x_1 y_1 p_1 \wedge x_2 y_1 p_1 \wedge \dots) \vee (x_2 y_1 p_1 \wedge x_3 y_1 p_1 \wedge \dots) \vee \dots, \tag{1}$$

where x_1 and y_1 denote the first possible x-position and y-position respectively and p_1 represents the first present we wish to place. Together $(x_1 y_1 p_1 \wedge x_2 y_1 p_1 \wedge \dots)$ will denote the space needed for the present of placed with the bottom left corner of present 1 in (x_1, y_1). This is illustrated in Figure 1.

The constraint that ensures that no presents overlap is formulated as such

$$\big((x_1 y_1 p_2) \vee (x_1 y_1 p_1)\big) \wedge \big((x_1 y_1 p_3) \vee (x_1 y_1 p_1)\big) \wedge \dots \wedge \big((x_i y_j p_k) \vee (x_i y_j p_{k-1})\big), \tag{2}$$

where again x_n, y_n denote positions in x- and y-direction and k denotes the total number of presents. This constraint goes through each gridpoint on the paper and says that no two presents can exist in the same gridpoint. This is illustrated in Figure 2.

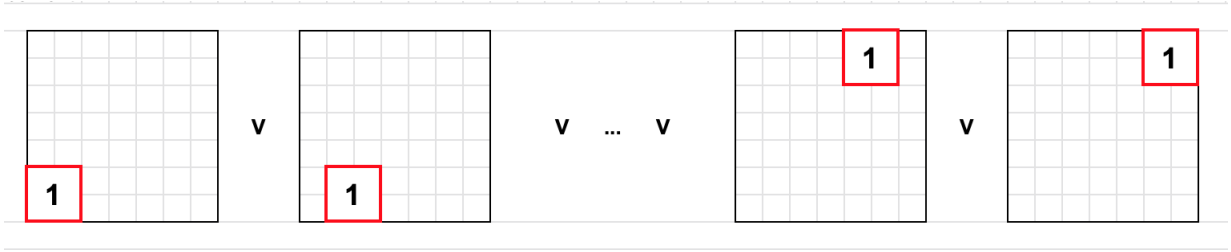


Figure 1: Illustration of how the placement constraint works.

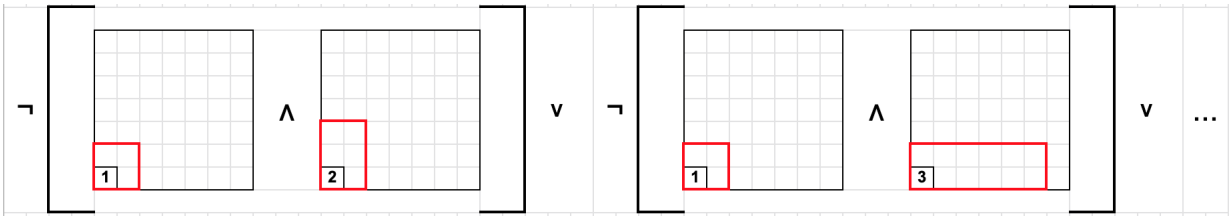


Figure 2: Illustration of how the overlap constraint works.

Furthermore, the constraint that ensures that each present is only placed once can be formulated as

$$\begin{aligned}
&\neg\big[(x_1y_1p_1 \wedge x_2y_1p_1 \wedge \dots) \wedge (x_2y_1p_1 \wedge x_3y_1p_1 \wedge \dots)\big] \wedge \\
&\neg\big[(x_1y_1p_1 \wedge x_2y_1p_1 \wedge \dots) \wedge (x_3y_1p_1 \wedge x_4y_1p_1 \wedge \dots)\big] \wedge \dots \wedge \\
&\neg\big[(x_1y_1p_2 \wedge x_2y_1p_2 \wedge \dots) \wedge (x_2y_1p_2 \wedge x_3y_1p_2 \wedge \dots)\big] \wedge \\
&\neg\big[(x_1y_1p_2 \wedge x_2y_1p_2 \wedge \dots) \wedge (x_3y_1p_2 \wedge x_4y_1p_2 \wedge \dots)\big] \wedge \dots
\end{aligned} \tag{3}$$

This equation compares two potential positions of each present and if the present is only placed in one or is not placed in any of them it will return true. This must hold for all packages in all positions. This is illustrated in Figure 3.

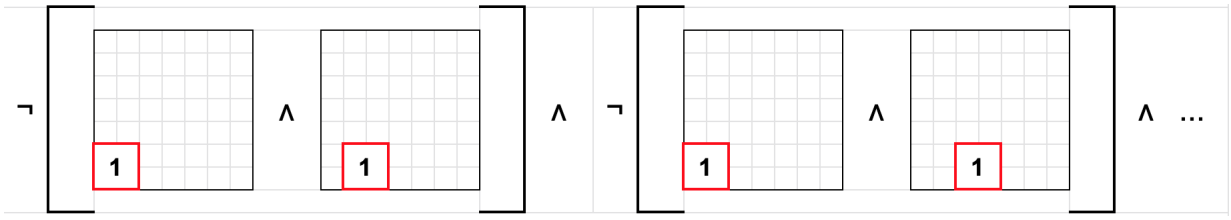


Figure 3: Illustration of how the constraint that ensures each present is only placed once works. After all placements for present 1 are investigated, one continues with investigating possible placements for the preceding presents.

Lastly, we have the constraint that ensures that each present is placed at least once.

This is formulated as

$$\begin{aligned}
&\neg\big[(x_1y_1p_1 \wedge x_2y_1p_1 \wedge \dots) \wedge (x_2y_1p_1 \wedge x_3y_1p_1 \wedge \dots)\big] \wedge \\
&\neg\big[(x_1y_1p_1 \wedge x_2y_1p_1 \wedge \dots) \wedge (x_3y_1p_1 \wedge x_4y_1p_1 \wedge \dots)\big] \wedge \dots \wedge \\
&\neg\big[(x_1y_1p_2 \wedge x_2y_1p_2 \wedge \dots) \wedge (x_2y_1p_2 \wedge x_3y_1p_2 \wedge \dots)\big] \wedge \\
&\neg\big[(x_1y_1p_2 \wedge x_2y_1p_2 \wedge \dots) \wedge (x_3y_1p_2 \wedge x_4y_1p_2 \wedge \dots)\big] \wedge \dots
\end{aligned} \tag{4}$$

This will compare all possible placements for each present and as long as each present is placed at least once it will return true. This is illustrated in Figure 4.

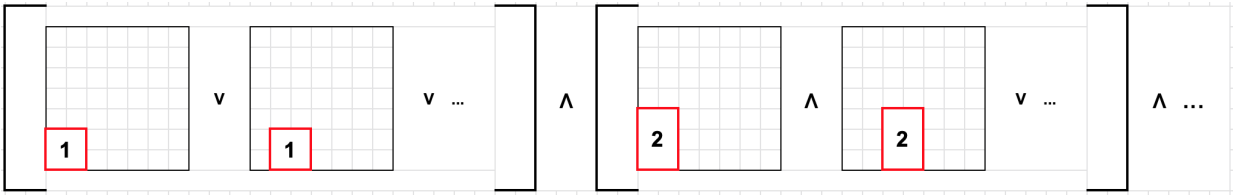


Figure 4: Illustration of how the constraint that ensures each present is placed at least once works.

Results

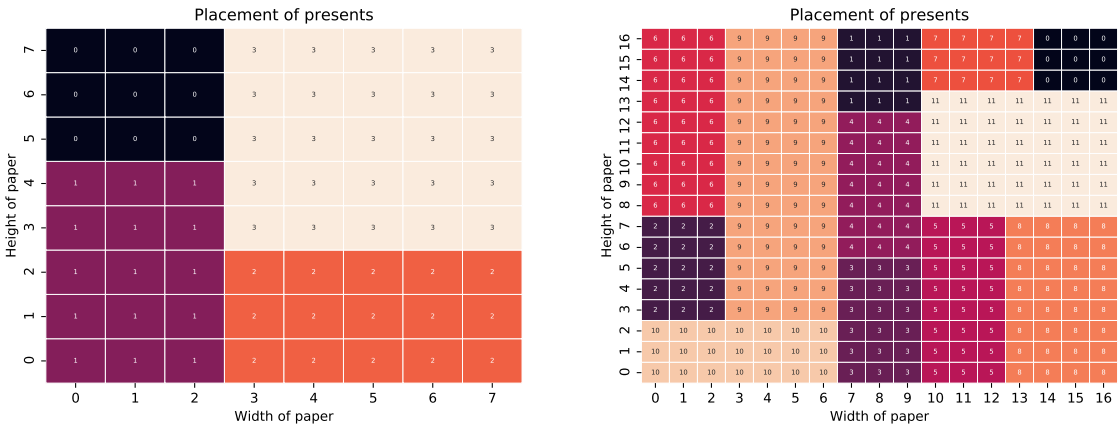
The SAT solver used a lot of time and needed a lot of capacity to complete, so unfortunatly my computer was not able to run for larger grids than 17x17 before crashing. However, I get no errors so I will assume it also works for the larger grids. The solution for the smallest and largest grid I was able to solve is shown in Figure 5. So as seen, the code works quite well, if not for it taking a lot of time.

SMT

Satisfiability Modulo Theories (SMT) optimization is quite similar to SAT optimization as it also uses propositional formulas to form the constraints of the optimization problem. However, these variables are integers instead of only Boolean values.

Constraints

The constraints needed for SMT optimization is 1) a placement constraint that ensures that the presents are placed in feasible gridpoints of the paper; and 2) a constraint that ensures that the presents do not overlap.



(a) (b)

Figure 5: Solution of PWP using SAT solver for a (a) 8x8 grid and a (b) 17x17 grid.

Implementation

To implement the PWP with SMT optimization I start with again defining a grid of integers named Axy where x and y denote the x- and y-values respectively of that gridpoint. Then, for each present I define a variable that will represent the bottom-left corner of the present as (xp, yp) where x and y denote the placement and p denotes the present we're currently investigating. With this I am now free to apply the placement constraints which I define as

$$\begin{aligned} 0 \leq x_p, y_p \\ x_p + d_{xp} \leq w \\ y_p + d_{yp} \leq h \end{aligned} \tag{5}$$

where x_p and y_p denotes the coordinate of the bottom left corner of the present we are trying to place, and d_{xp} and d_{yp} are the dimensions of said present. Lastly w is the width of the paper and h is the height. This will ensure that the bottom-left corners are placed such that the whole present will be placed correctly on the paper.

Next, I define the constraint that ensures that no two packages overlap as

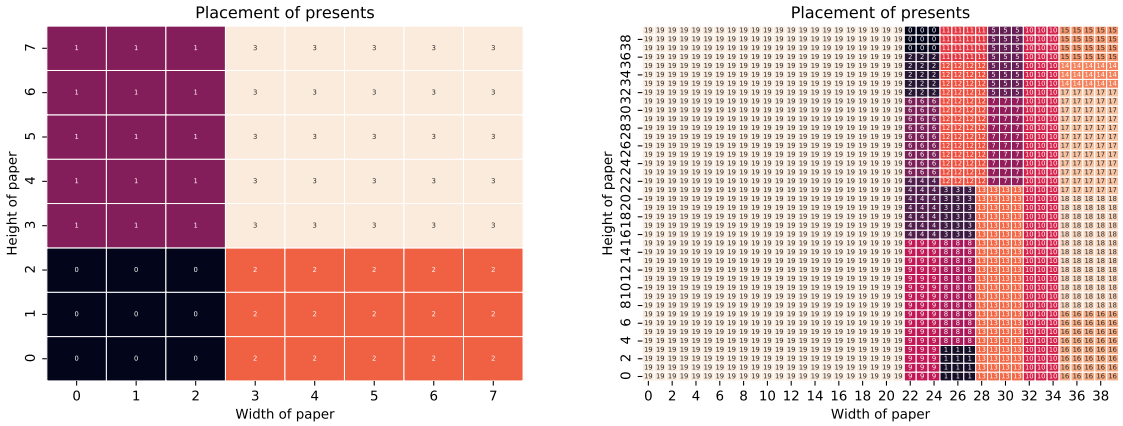
$$\big((x_p + d_{xp} \leq x_{p-k}) \vee (x_{p-k} + d_{xp} \leq x_p)\big) \vee \big((y_p + d_{yp} \leq y_{p-k}) \vee (y_{p-k} + d_{yp} \leq y_p)\big) \ \forall \ p, k, \tag{6}$$

where k also denotes the number of presents such that all presents get compared to each other and checked that none of them overlap.

Results

The SMT solver works very well. It is rather quick and I was able to solve all the PWP for all the gridsizes, with the grid that took the longest time was the 39x39 which used

approximately three hours to reach a solution. All the other grids took less than 30 min each, often much less. The result for the largest and smallest grid is shown in Figure 6.



(a) (b)

Figure 6: Solution of PWP using SMT solver for a (a) 8x8 grid and a (b) 40x40 grid.

Improvements

Rotation

One could modify the SAT and SMT solvers to allow rotation of the packages. To do so in the SAT one has to modify the placement constraint shown in Equation 1, to also include the rotated dimensions. This will not alter the equation per ce, but rather just include more x and y values. This is done in practice in the function called `SAT_rotation()`. For the SMT solver, one would have to modify both Equation 5 to satisfy rotation as such

$$\begin{gathered} 0 \leq x_p, y_p \\ \big((x_p + d_{xp} \leq w) \wedge (y_p + d_{yp} \leq h)\big) \vee \big((x_p + d_{yp} \leq w) \wedge (y_p + d_{xp} \leq h)\big). \end{gathered} \tag{7}$$

This is done in the function called `SMT_rotation` in the SMT-code. This is easier than for CP programming in MiniZinc as there you would have to implement local constraints instead of just using the aldreasy existing global constraints.

Multiple dimensions

Another improvement to the code is to make the code more efficient by treating all presents of the same dimensions as one. This could be done by having the code memorise which dimensions have already been treated, and then instead of calculating the new constraints

for a piece of the same size, instead just copy the constraints that were used earlier and add them again to the solver.