In this exercise I was to solve the Present Wrapping Problem (PWP) with a CP solver made in Minizinc and Python.

Constraints

For the CP solver I chose to use two types of constraints: 1) a placement constraint, and 2) a constraint that ensured no overlap of the presents. In the Minizinc solver, the placement constraint could be formulated as

$$(x_p + d_{xp} \leq w \wedge x_p \geq 0) \wedge (y_p + d_{yp} \leq h \wedge y_p \geq 0) \ \forall \ p,$$

where x_p adn y_p represents the bottom left corner of present p and and d_{xp} and d_{yp} represent the dimensions of present p in x and y direction respectively. This constraint will ensure that the presents are placed correctly on the paper, and will not exceed the boundaries of the paper.

The overlap constraint can be set by the global Minizinc function `diffn()`. This constraint ensures that given rectangles with given sizes are non-overlapping [1].

Search Heuristics

In MiniZinc one can use different given search heuristics. The ones I've chosen to investigate are

1. `int_search(place, input_order, indomain_min, complete)`
2. `int_search(place, input_order, indomain_median, complete)`
3. `int_search(place, first_fail, indomain_min, complete)`
4. `int_search(place, first_fail, indomain_median, complete)`
5. `int_search(place, input_order, indomain_random, complete)`

as they are given in the MiniZinc documentation. I've only looked at `int_search` as it is the only pre-made search heuristic that return an integer as its result. The first option will try to insert the presents in the given order to the minimal value. This is similar to the strategies deployed by option 2 and 5, as they instead try to assign the presents in input-order to the median value and a random value respectively. These work fine, but may often become unsatisfiable and the heuristic has to backtrack. For options 3 and 4 however this happens less frequently as they try to place the most difficult presents first to the lowest and the median value respectively. The median version will work best, as this will the most quickly find unsatisfiable solutions and be able to fix them, so this is the search heuristic I've deployed.

Results

The solver seems to be able to solve all the problems, it's just very slow. The results were are in the format of a list giving all the bottom left corners of the presents. I was not able to transfer my MiniZinc into Python, so I've unfortunatly not been able to plot the solutions. Therefore, I don't have an *out* folder for this part of the project, but the code will give correct results when run. I have modified all the instances to work for MiniZinc.

Rotation

To allow for rotation of the presents one would have to implement unique constraints as I've not found a ready.made global constraint to allow for this. This would therefore be a lot more work, and it would be easier to do so for SAT/SMT where one already has made local constraints.

Multiple dimensions

To make the code more efficient one could implement a function that would treat all presents of the same dimension as one, as explained in the report for SAT/SMT solving. This again would be quite difficult for MiniZInc, as again one would have to implement local constraints instead of using global ones.

References

[1] Global Constraints, *MiniZinc Documentation:* https://www.minizinc.org/doc-2.4.3/en/lib-globals.html