COMPX556 Metaheuristic Algorithms Assignment One
Variable Neighbourhood Search

Hannah Trebes, Connor Fergusson

***Abstract*—This report details our approach to solving the 2D Rectangle Packing Problem using Variable Neighbourhood search. Our main focus is on producing different Neighbourhood functions.**

I. Introduction

THIS report covers the methodology taken to implement a Variable Neighbourhood Search (VNS) for the purpose of producing an optimal solution to the 2D Rectangle Packing Problem. The results garnered from this process are also presented.

The 2D Rectangle Packing Problem is an NP hard problem. An algorithm to solve this must take a set of items (boxes in our case) and fit them onto an object of fixed width in such a way that the total height of the items is minimized and none of the items are colliding/overlapping each other. The use of some form of Metaheuristic Algorithm to produce solutions to this problem increases the chance that the solution produced will be a global optimum i.e. the best possible solution, rather than a local optimum as may be generated by a Local Search Algorithm. We decided to use VNS to produce solutions to this problem.

VNS works by using multiple neighbourhoods to improve the performance of Local Search. A neighbourhood refers to a set of solutions reached through a single change to the current solution. The neighbourhoods are produced by neighbourhood functions. By switching between neighbourhoods VNS can reveal a local optimal solution that exists in one/some neighbourhoods but not in others.

II. Methodology

A. Problem Representation

Initially we decided to use a 2D array to represent the items (boxes hereafter). The 2nd array would hold the information related to each box that could be required in a calculation. However we then changed to creating a Box class. This enabled us to have an ArrayList in our VNS class which stored the boxes directly.

Each box is initialized with a width, height and calculated area, and given x, y coordinates of (0,0). These properties of each Box are public and directly accessible from the VNS class. Methods in the Box class allow the box to be rotated, to check if the Box fits within another given Box, to check if the Box is colliding with any other boxes and which box it is colliding with.

To communicate the solution two methods are used. The results method in VNS outputs the ArrayList index of each box and its final x, y coordinates. The drawBoxes method takes an ArrayList of boxes and draws them as they have been placed on the object in a scaled pop-up window. The graphical representation does not label the boxes.

B. The Initial Solution

In researching potential neighbourhood functions to help solve this problem, we came across a great deal of algorithms[1] that detailed ways to achieve a potentially minimal initial solution. From this, we gathered that it would be quite easy to focus entirely on generating a very good initial solution. However for the purposes of the VNS neighbourhood functions we intended to implement, a slightly less optimal initial solution is preferred. This gives the metaheuristic algorithm more wiggle room and allows it to make a greater imprvenemt to the initial solution. Accordingly, after examining existing initial algorithms and carrying out our own testing we implemented a very basic initial solution.

Our initial solution orders the given set of boxes by area with the greatest area at the first index of the ArrayList, and the least area at the last index. Every Box is then rotated so that its height is minimized. This is done in getData as the boxes are read in from the given CSV file. The newSolution method then places each box on the object in index order starting at (0,0), increasing the x until the box no longer fits on that row, then resetting x to 0 and increasing the y to the height of the tallest box on the previous row. Once all the boxes have been placed, the findGaps method is used to attempt to move boxes into the gaps. The boxes are first ordered by area because the ordering of the boxes produced better final solutions in every test run than placing the boxes in the order they are read in from the file.

C. The Neighbourhood Functions

1) Initial Ideas

Initially we had several ideas for potential neighbourhood functions that did not make it into the final VNS. These included Tetris, Random, RightDownLeft and the method pair FindGaps/FillGaps.

Tetris was a function that would try to fill a row at the bottom of the object and if this was achieved, would then no-longer move any boxes from that filled row. Effectively locking the boxes to prevent breaking up a perfect row.

Random consisted of taking all the boxes and placing them anywhere on the object, then shuffling them down and left until no more movement was possible. This was intended to really break away from any local minimum.

RightDownLeft would take the top box, move it to the bottom right of the object (max x value, y = 0), increment y until there were no collisions, then shift the box as far down and left as possible. This would repeat until no change to the total height was observed.

Find/FillGaps are two methods that work together to move boxes into gaps on the object. FindGaps locates the gaps and passes them as an ArrayList of Boxes to FillGaps. FillGaps then makes a list of boxes that would fit in each gap and randomly selects from these one box. It then moves this box into the gap and if this does not cause a collision, width overflow, or solution height increase, leaves it there.

While the first two ideas were eventually discarded – they produced too much change too often for a minimal solution to be reached – RightDownLeft and Find/FillGaps were retained as methods. Find/FillGaps is called after each newSolution is generated to compact the boxes and RightDownLeft is called after each FillGaps.

2) Retained Ideas

A basic LocalSearch method is implemented in the VNS and used in conjunction with the Shake method.

The 3 neighbourhood functions retained into the final VNS are the 3 permutations of the Shake method.

- The 1st permutation of Shake takes a random box and moves it to the end of the ArrayList.
- The 2nd permutation takes a random box, rotates it, and moves it to the end of the ArrayList.
- The 3rd permutation rotates a random box.

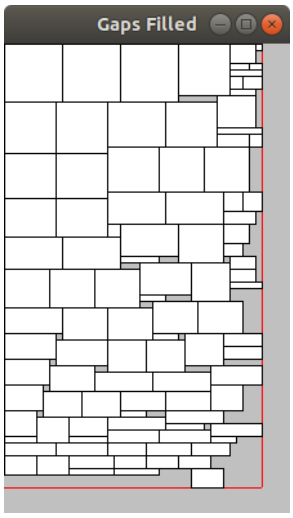
After all permutations of Shake, newSolution is run again (which calls Find/FillGaps and subsequently RightDownLeft).

III. Results

For the sake of the results, unless specified otherwise, the input file used was M1A.csv. With a *tmax* of 100 and an object width of 40.

A. Initial Solution

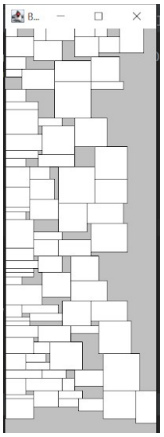
With the initial solution that we implemented, the placing of the boxes by newSolution is the same each time. However at the end of newSolution when FindGaps is called, this adds an element of random placement to the solution so the initial solution is different each time the code is run. An example output of the initial solution is shown below with height 69.




```
Box 96 is at position: 38, 14
Box 97 is at position: 32, 61
Box 98 is at position: 35, 3
Box 99 is at position: 38, 3
Box 100 is at position: 39, 0
TotalHeight: 69
Number of Boxes: 100
```

B. Tetris

The below image shows the result of running the discarded Tetris method on an area ordered initial solution. This particular instance had a height of 97. Due to the large number of changes that would have to be made to make this a viable neighbourhood for a minimal solution, this method was discarded entirely. When implemented as a Neighbourhood function, there was very little decrease in height, and more often than not height would increase.



C. BVNS

Our VNS works on the Basic Variable Neighbourhood Search described in lecture. The results for M2C and M3D were on an object with width 100. The best results obtained from this total program are shown in the following table.

Input File	tmax = 10	tmax = 100	tmax = 500
M1A.csv	65 ()	64	63 (1hr)
M2C.csv	262 (5min)	264(10min)	-
M3D.csv	420 (6min)	-	-

D. Run-Time

When run for long enough it is apparent that our program can compute the “theoretical minimum” that some of our classmates identified (63). However the program is quite slow in general and running it for long periods of time often produces disproportionately small increases in the best solution. This long runtime is mainly due to the number of for-loops and nested for-loops that cycle through all of the boxes in the solution for every neighbourhood iteration.

Because the stoping criteria is an iteration counter, the time taken for one input-file under the same parameters may vary due to the randomization elements in the methods used to calculate the solution.

IV. Conclusion

Our implementation of the VNS algorithm seems to work quite well in finding optimal or near optimal solutions to the 2D Rectangle Packing Problem. The only issue with our implementation is the time it takes to calculate any large number of solutions, and the slightly variable nature of this run-time. The initial change in decision to implement a Box class has made the code much easier to write and understand as well as making the displaying of results better.

Developing a good initial solution through the use of newSolution, Find/FillGaps and the movement methods allowed us to have a slower overall algorithm as the improvements being made will be less drastic than if a poor initial solution was found. This allowed our neighbourhood functions to be quite simple in themselves and so easier to expand upon were we to continue this project. In conclusion, we have produced an inplementation of the metaheuristic VNS algorithm that produces near optimal solutions, with a cost paid in runtime.

V. Video and Code Links

Video:

https://drive.google.com/file/d/12FMTomj0w8TuPnNnUpWldS010Gg-eSsj/view?usp=sharing

Code:

https://github.com/trebesh/556-Assignment-1

VI. References

[1] Survey on two-dimensional packing

https://cgi.csc.liv.ac.uk/~epa/surveyhtml.html#toc.2.1