Individual Design Journal 6 — **Basil R. Yap**
2018 – Capstone 7 — Journal Date: 2018/06/21
Project 61: Steelcase — Due Date: 2018/06/17

Problem Statement

All work regarding the *Container Loading Optimization (CLO)* problem is going according to plan and a functional prototype is expected to be complete in time for **Review 3**.

As covered in the previous Journal (**Journal 5**), the *CLO* Problem has been broken down with consideration to the usability of the algorithm with respect to the management team at Steelcase Malaysia as well as members of the public during the Capstone Showcase.

Therefore, the work completed by me over the last few weeks are as follows, in chronological order:

1. Benchmarking of existing 3D-bin packing algorithms
   (a) Literature review & Implementation (continued)
   (b) Algorithm scoring & Test results
2. Visualisation of algorithm solutions
   (a) Feasibility testing of conventional visualisation tools
   (b) Preliminary design of web application & UI *
   (c) Front-end development iterations

** not covered in this Journal.*

Benchmarking of existing 3D-bin packing algorithms

Literature review & Implementation

Upon further research into the problem of 3D-bin packing, a list of algorithms were compiled and implemented. The following are the algorithms we have finalised and completed, accompanied by a short description on their respective packing strategies:

1. S-Pack / H1 [1]

   S-Pack considers the problem by dividing the bin into layers. Each layer is filled in order from the back of the bin to the front using 2D-bin packing algorithms. Each layer's 2D-bin area depends on the 3D dimensions of the boxes arranged in prior layers.

2. MPV-BS / H2 [1]

   MPV-BS is a branch-and-bound method where only m branches were considered at each node. The tree search space exponentially increases with more classes of box sizes, therefore, a hyperparameter was set in order to limit the search space of the algorithm. Based on the paper's experimental results, $m = 4$ was selected.

3. Height first - Area second (HA) [3]

   Height first - Area second is a straight forward heuristic, where the bin was loaded from a corner with priority given to boxes with the tallest height and ties were broken with priority given to boxes with the largest area.

4. Extreme point best fit decreasing [2]

   Extreme point best fit decreasing is a combination of two heuristics, extreme point and best fit decreasing. Extreme point selects the locations to consider for box placement, while best fit decreasing gauges the *merit* of the box placed in that location. The bin is iteratively filled from back to front by assigning boxes (if any) with the highest *merit* score at every extreme point.

Algorithm scoring & Test results

For testing, the following parameters were used in the test algorithm, as shown in **Algorithm 1**:

- Bin size: 1190 x 228 x 219 (*SMM's 40ft Container Size*)
- Number of Boxes generated: 200
- Box size range (width, height, depth): [73, 119] (*min/max of current box size*)
- Number of Box size classes: [1, 6]
- Number of Repeat tests: 100

The results of the test can be seen graphically in **Figure 1.**

Experimentally, Extreme Point performed the best across all box size classes. It performed on average 7% better than S-Pack, which was the worst algorithm overall.

Algorithm 1 Test Algorithm

```
procedure TestProcedure(n)
    noBox $\leftarrow$ 200
    batch $\leftarrow \lfloor$noBox$/n\rfloor$
    bin $\leftarrow (1190, 228, 219)$
    dimMin $\leftarrow$ 73
    dimMax $\leftarrow$ 119
    repeat $\leftarrow$ 100
    boxList $\leftarrow$ Array[$\mathbb{R}$]
    solutionArray $\leftarrow$ Array[$\mathbb{R}^2$]
    for $i = 1$ to repeat do
        noBox $\leftarrow$ 200
        while noBox $> 0$ do
            if noBox $\geq (2 \times$ batch$)$ then
                noBox $\leftarrow$ noBox $-$ batch
                boxObj $\leftarrow$ randBox(dimMin, dimMax)
                for $j = 1$ to batch do
                    boxList append boxObj
            else
                boxObj $\leftarrow$ randBox(dimMin, dimMax)
                for $j = 1$ to noBox do
                    boxList append boxObj
                noBox $\leftarrow$ 0
        for $f(x, y) \in$ algorithmSet do
            arrangedSet $\leftarrow f$(bin, boxList)
            failedSet $\leftarrow \{$boxList$\}\setminus\{$arrangedSet$\}$
            solutionArray[$f(x, y)$][i] $\leftarrow$ Count(failedSet)
    for $f(x, y) \in$ algorithmSet do
        solutionArray[$f(x, y)$] $\leftarrow$ Average(solutionArray[$f(x, y)$])
    return solutionArray
```

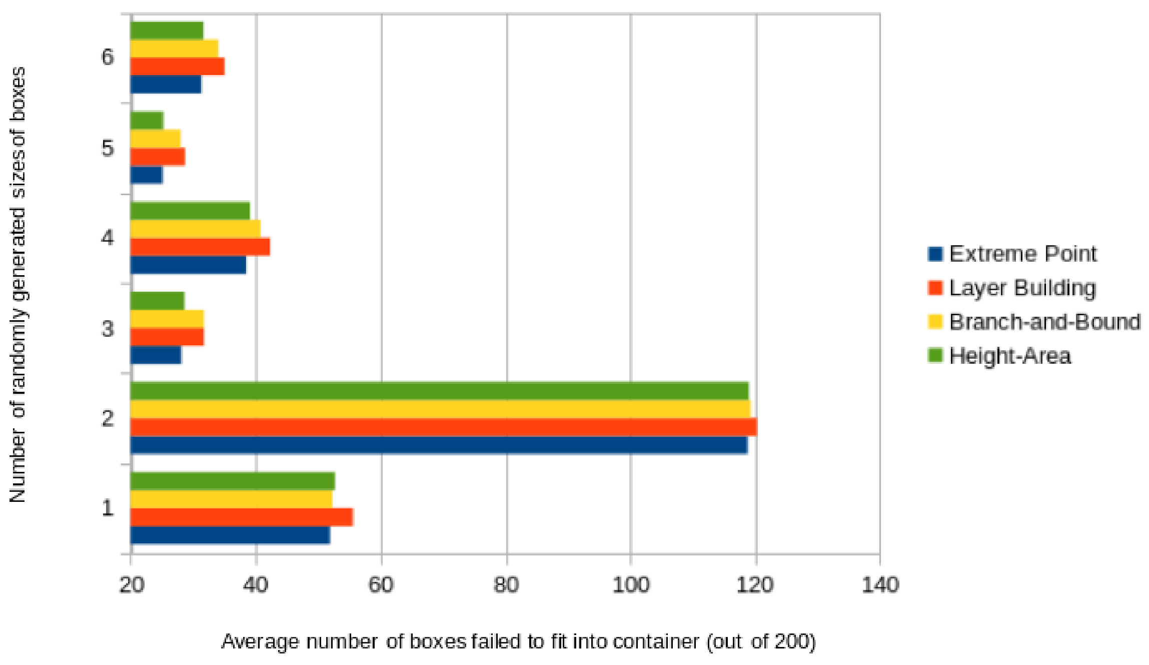

Figure 1: Benchmark Test Results

Visualisation of algorithm solutions

Feasibility testing of conventional visualisation tools

Upon completion of the algorithms, the focus turned towards finding a suitable way of visualizing the solutions generated. The solution consists of a "list" object, where each box has the following two attributes:

- reference id: a unique identifier to help keep track of the boxes
- coordinates: the arranged coordinates of the centroid of the box

Using a sample solution, we testing the functionality of the following software and implementation languages:

- Microsoft Excel
- R Language (3dplot, ggplot & plotly)
- Python (matplotlib, vtk & plotly)
- Tableau & PowerBI

Most of which resulted in the same problem, the visualisations do not convey the information provided by the solution sufficiently well. Due to the limitations of the 3D projection onto 2D space, the relative positions of the boxes are not retained. Also, in cases where the number of boxes/layers are numerous, the overlap of points make the whole visualisation completely infeasible.

It is, however, worth noting that a web application created using plotly in R or Python allows the user to turn the plot to provide sufficient navigation of the information. The web application created was still much to be desired as the solution was shown as points and customization was heavily limited by the extents of the Python package.

With the success of a simple web application, I decided that a full on web application would be better suited for the task and more fitting as a prototype to be shown to our industry partners.

Front-end development iterations

The following were some considerations when developing the GUI. All milestones were set using the software development principle of *version control.*

v1.xx Establish initial 3D environment

- **v1.03** Complete lighting and axis
- **v1.05** Spawn movable boxes
- **v1.11** Complete control panel for box generation

v2.xx Set-up controls for boxes and bins

- **v2.02** Complete controls for bulk spawn of boxes
- **v2.03** Create permanent 3D estimate of bin size

v3.xx Interface between algorithm and GUI

- **v3.01** Add control panel option for algorithm
- **v3.02** Test simple python API using Flask
- **v3.05** Functional prototype for one algorithm

References

[1] Martello, S., Pisinger, D., & Vigo, D. (2000). The three-dimensional bin packing problem. *Operations Research*, *48*(2), 256-267.

[2] Crainic,T. G, & Perboli, G., & Tadei, R. (2007) Extreme Point-Based Heuristics for Three-Dimensional Bin Packing. *INFORMS Journal on Computing*, *20*(3). 368 - 384.

[3] Lodi, A., Martello, S., & Vigo, D. (2002). Heuristic algorithms for the three-dimensional bin packing problem. *European Journal of Operational Research*, *141*(2), 410-420.