Brief Description of 2D Rectangular Bin Packing Algorithm

The problem statement given here is to solve the packing of rectangles into 2D finite bins that constitute a square. In combinatorial optimization, 2D Rectangle bin packing is a classical problem. The idea is to fit a sequence of Rectangles of variant sizes, let's say:

$$\{R_1, R_2, R_3, . . . , R_n \} ; \text{given } R_i = \{ W_i, H_i\}$$

Into a minimum number of bins of size (W, H). The constraints here are that no two rectangles must intersect or be contained one another. This problem can classified as an NP-Hard formulation by a reduction of 2-partition problem. A Packing is said to be orthogonal if all the sides of the placed rectangles are parallel with the bin edges. The packings that are orthogonal and allow each rectangle to be rotated by 90 degrees are considered. This is called Rotatable Rectangle bin packing. To solve this problem the maximal rectangles algorithm is chosen.

Maximal Rectangles algorithm

The Maximal Rectangles algorithm stores a list of free rectangles that represents the free area of the bin. Here in this scenarios, the bin is that of a square size. The Maximal rectangles algorithm implements an operation that corresponds to picking both split axes at the same time.

When an input Rectangle R is placed to bottom left of a free square F, we compute the two rectangles F_1 and F_2 that cover the L-shaped region of $F \cap R$ and update the F. The name "Maximal" refers to the property that these rectangles F_1 and F_2 are formed to be the maximal length in each direction. It means that at each side they touch either the bin edge or some rectangle already placed into the bin.

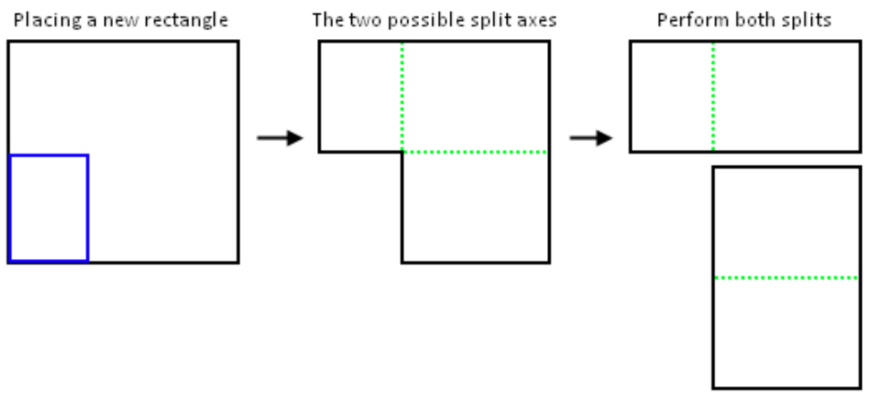


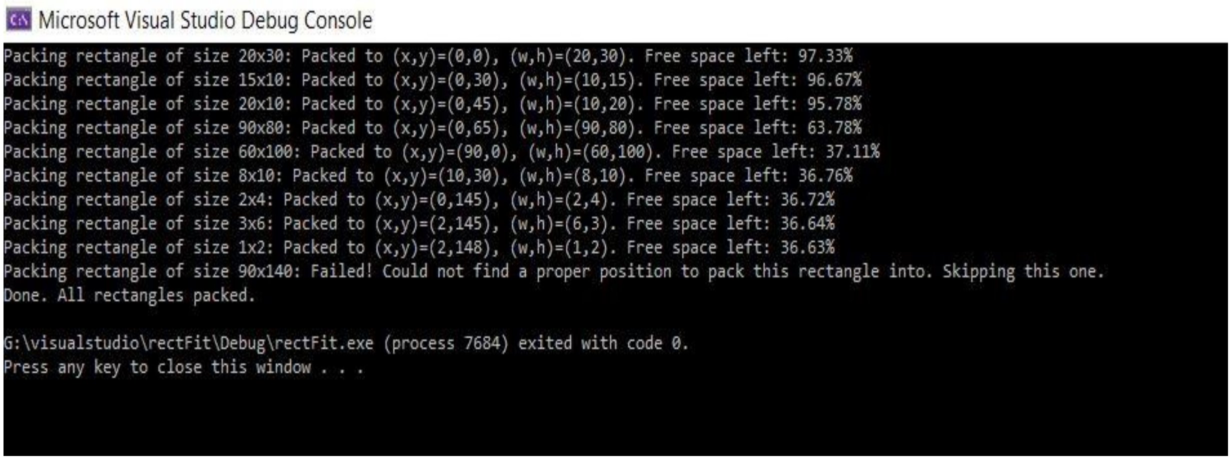
Fig 1: Illustration of Maximal Rectangle Algorithm

Algothirmic Solution:

- To achieve a solution to this, first a 2D bin of equal dimensions (Square!) is chosen.
- Further define whether the packing algorithm is allowed to rotate the input rectangles by 90 degrees so as to find a better placement.
- Now we define a set of heuristic rules to decide where to place a new rectangle.
- The heuristic rule chosen here is Best Short Side Fit.

- The idea here is that it positions the rectangle against the short side of a free rectangle into which it fits the best.
- Once the rules are set, the rectangles are inserted into bin the possibly rotated.
- Then the ratio of used surface area to total bin area is calculated.

Sample solution:



Microsoft Visual Studio Debug Console

```
Packing rectangle of size 20x30: Packed to (x,y)=(0,0), (w,h)=(20,30). Free space left: 97.33%
Packing rectangle of size 15x10: Packed to (x,y)=(0,30), (w,h)=(10,15). Free space left: 96.67%
Packing rectangle of size 20x10: Packed to (x,y)=(0,45), (w,h)=(10,20). Free space left: 95.78%
Packing rectangle of size 90x80: Packed to (x,y)=(0,65), (w,h)=(90,80). Free space left: 63.78%
Packing rectangle of size 60x100: Packed to (x,y)=(90,0), (w,h)=(60,100). Free space left: 37.11%
Packing rectangle of size 8x10: Packed to (x,y)=(10,30), (w,h)=(8,10). Free space left: 36.76%
Packing rectangle of size 2x4: Packed to (x,y)=(0,145), (w,h)=(2,4). Free space left: 36.72%
Packing rectangle of size 3x6: Packed to (x,y)=(2,145), (w,h)=(6,3). Free space left: 36.64%
Packing rectangle of size 1x2: Packed to (x,y)=(2,148), (w,h)=(1,2). Free space left: 36.63%
Packing rectangle of size 90x140: Failed! Could not find a proper position to pack this rectangle into. Skipping this one.
Done. All rectangles packed.

G:\visualstudio\rectFit\Debug\rectFit.exe (process 7684) exited with code 0.
Press any key to close this window . . .
```

Here we have defined a list of rectangles and a square grid of 150 x 150.

The rectangles are placed in this grid and the possible free space is calculated based on the occupancy. If a rectangle cannot be placed in the defined grid, it is skipped.