2D Bin packer

Collection of insights/ideas gained by working on 2D bin packer. A lot of details and small but logical improvements are not included here.
Everything is evaluated on the dataset for code challenge.

Sections: Shapes, Open-Spaces and Packer are mostly for reference, the interesting stuff happens in the section about the score heuristic

Shapes (Super-items)

At initialisation, I calculate every possible rectangular ordering of a product.

For example a product of which there need to be packed 8 items will have

(1+2+2+3+2+4+2+4) * 2 (orientations) = 20 possible super items

Quantity					
1	(1,1)				
2	(1,2)	(2,1)			
3	(1,3)	(3,1)			
4	(1,4)	(2,2)	(4,1)		
5	(1,5)	(5,1)			
6	(1,6)	(2,3)	(3,2)	(6,1)	
7	(1,7)	(7,1)			
8	(1,8)	(2,4)	(4,2)	(8,1)	

The algorithm will only work on these super-items and not on individual items.
Note that individual items can still be packed but they will be contained in a super-item consisting of 1 item.
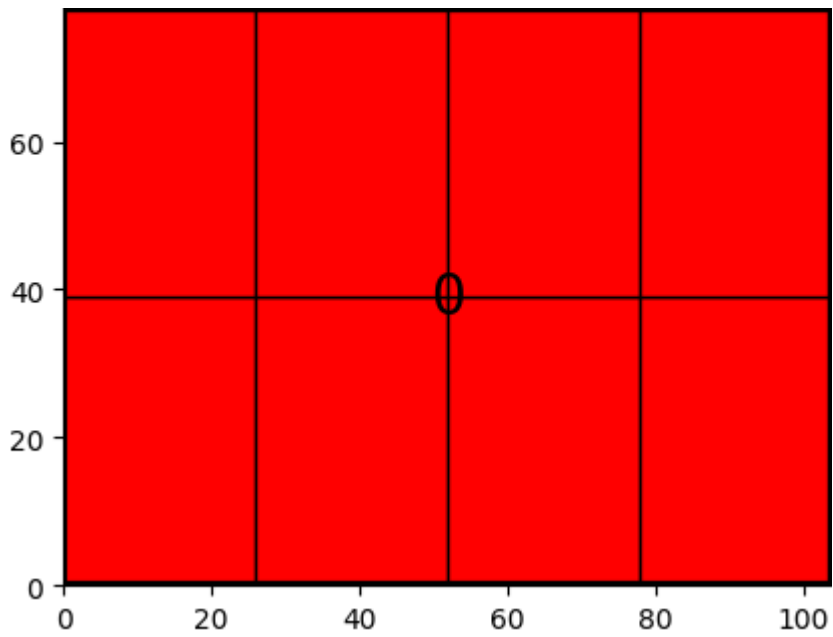
Super-item consisting of 8 items

Pros:
- More possibilities for the packer -> better packing
- More items will be packed in one iteration -> fewer iterations better performance

Cons:
- More possibilities -> more calculations

Open-Spaces

Every super-item is not placed into a bin or layer but in an available open space.

Every item that is placed into an open-space will split that space into 2.

Right now I use non-overlapping open-spaces. The open-space is split as follows
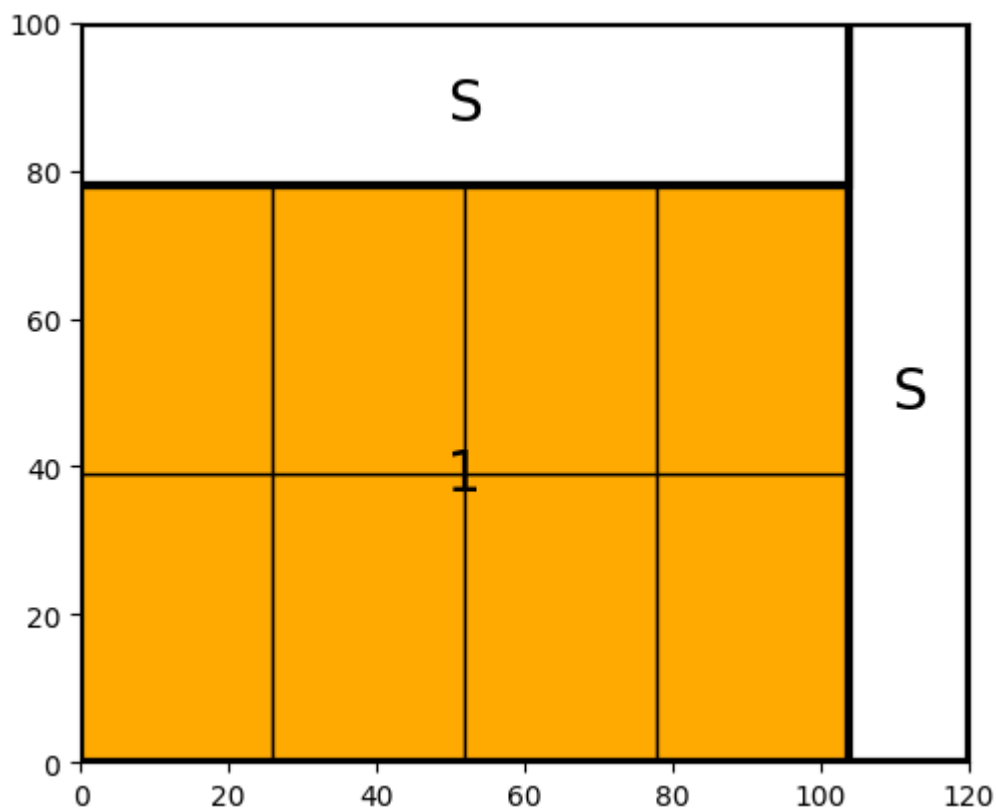
Placing an item (orange) splits the open-space into two new open-spaces (S)

This is obviously not perfect, in the figure an item of dimensions (120,20) could still be packed, but the way the open-spaces are defined do not allow this.
Overlapping open spaces would allow for this item to be packed.

Compared to the normal way of defining open-spaces (overlapping), non-overlapping open-spaces is most likely worse but way faster, as not only there are fewer open-spaces to consider but managing and splitting the open-spaces is way more efficient and easier.

Changing to overlapping open-spaces will likely improve packing performance at the cost of execution time. Will do this someday

Space Merging

In order to somewhat combat the disadvantage of non-overlapping spaces. I implemented space merging. When two spaces have 2 perfectly aligning corners they can be merged into one space. In the following figure the open-spaces above item 15 and item 14 were merged making the packing of item 17 possible.
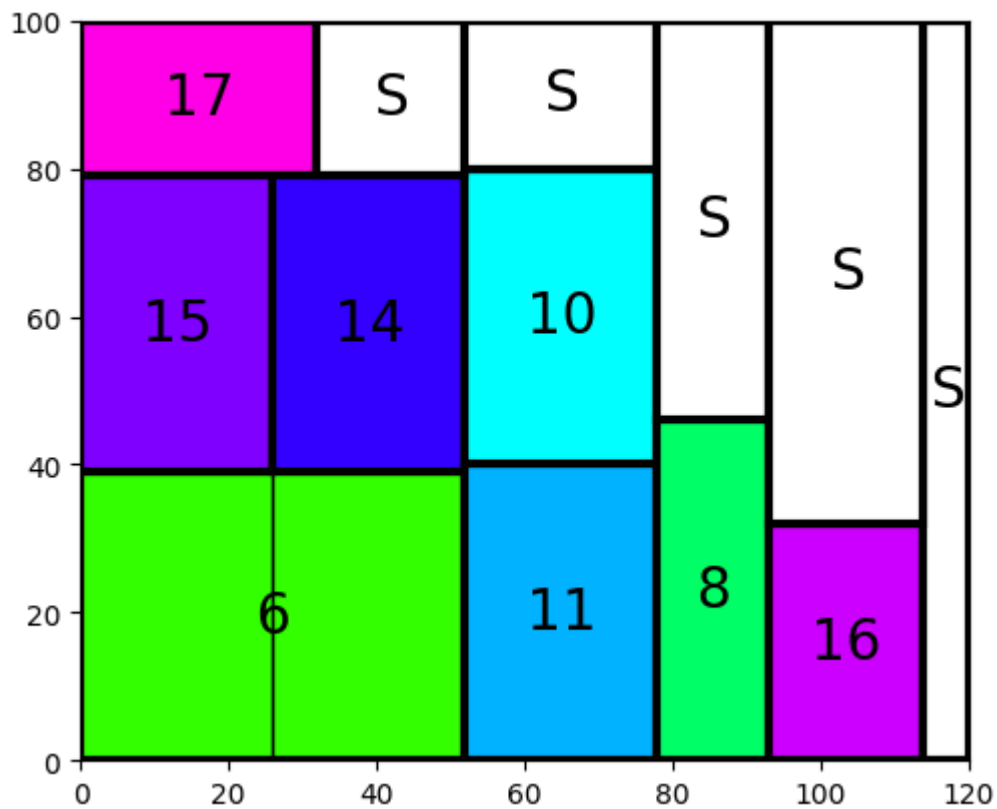
Packer

At each iteration of the packer for each possible combination of shape and open-space, a score is calculated. The highest score then determines which shape is packed into which open-space

	shape1	shape2	shape3
space1	0	.3	.3
space2	.4	.7	0
space3	.9	.1	.1

In this case shape1 will be packed into space3, space3 will then as a result be splitted.

Score heuristic

Here comes the actual interesting stuff

Determining which shape and space to select is probably the most important aspect of the entire algorithm, and where gains are made. As a reminder this score is calculated for each possible shape, space combination. Invalid combinations are masked to zero at the end (Shape doesn't fit in space or not enough items left to make shape)

Fill rate

Simple idea, the more the shape fills the open-space the better -> fill rate

Fixing the open-space dimensions and increasing the dimensions of a to be packed shape, produces the following scoring plots in the 1D and 2D case.
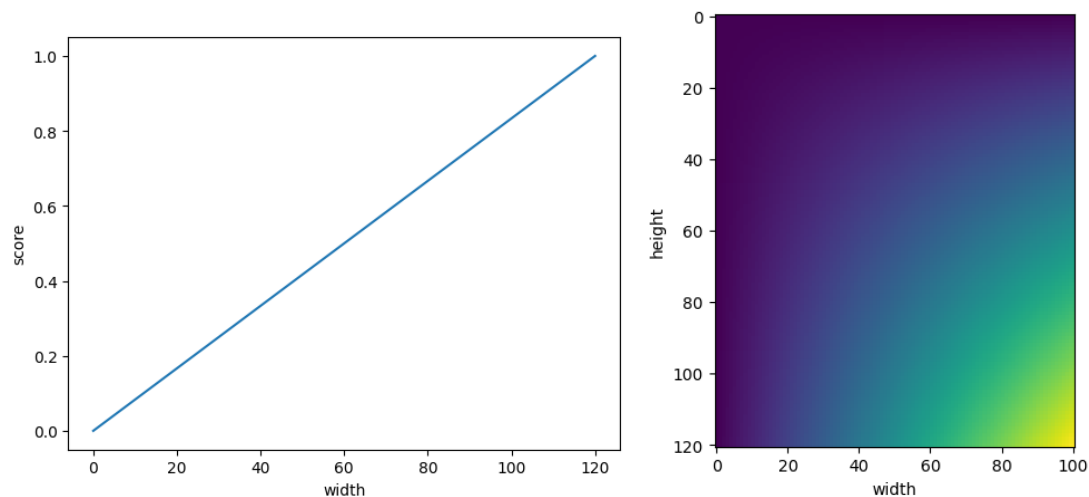
1D: $score(x) = x/w_s$

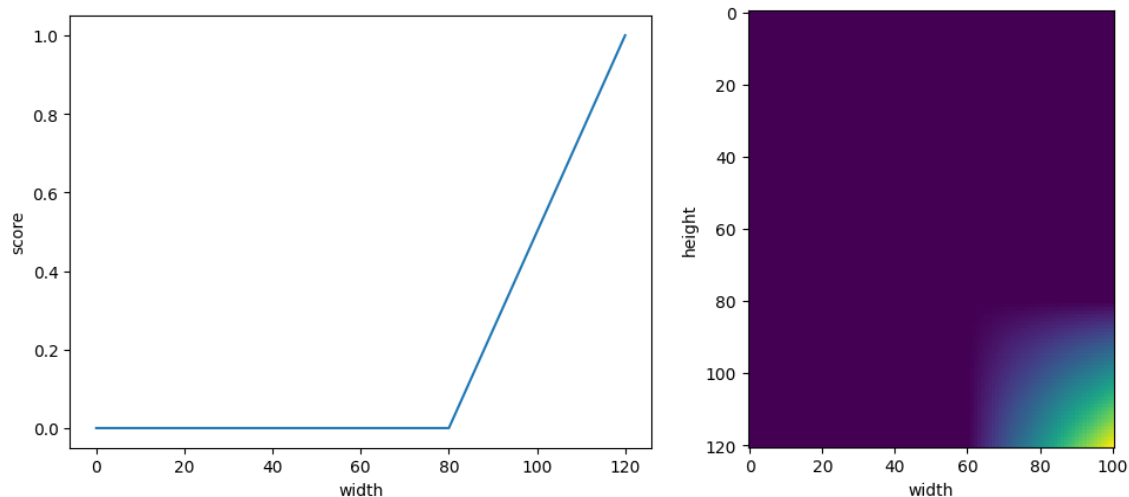2D: $score(x, y) = area_{shape}/area_{space} = (x.y)/(w_s.h_s)$

With x,y the width/height of the shape (super-item) and w_s, h_s, the width/height of the space

Observation:
$score(x, y) = score(x) * score(y)$

Wasted space

Second idea: instead of the fill rate we look at the amount of space that is wasted. Wasted space is space in which no item can fit.

1D: $score(x) = (x > w_c): (x - w_c)/(w_s - w_c) \; else \; 0$

With x the width of the shape, w_s the width of the space and $w_c = w_s - w_m$, the cutoff-point

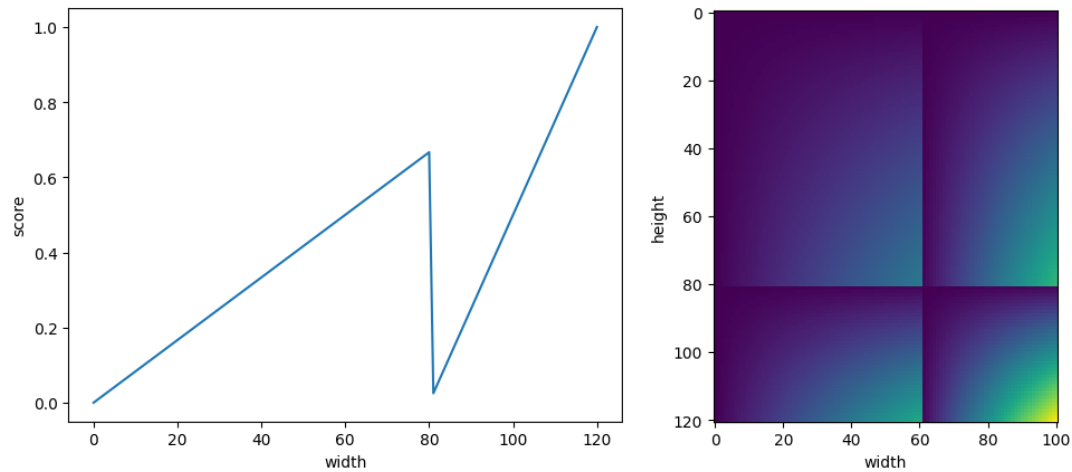w_m is the minimum width of all shapes.

$$score(x, y) \; ?= \; score(x) \; * \; score(y)$$

For the 2D score function I multiply two 1D score functions just as with the fill rate score function. This is (probably) NOT entirely correct, but hey, we're talking about a heuristic so perfection is not really the goal here. There is a dependency between the width and height in this case, specifically when determining w_m and h_m.

Combination

Current score function is a combination and combines the best of both worlds.
Score increases with a better fit up until a point where wasted space is introduced and the score drops to zero again. The score again reaches a maximum when wasted space is minimised and the item fits perfectly into the space.

1D: $score(x) = (x > w_c): (x - w_c)/(w_s - w_c)\ else\ x/w_s$

With x the width of the shape, w_s the width of the space and $w_c = w_s - w_m$, the cutoff-point. w_m is the minimum width of all shapes.

Again for the 2D score function I simply multiply two 1D score functions.

Variants of Combination

Both sections of the combination score function are linear. What would happen if we change to polynomial or even exponential functions. The only real thing that we actually change is the point when we accept a certain amount of wasted space.
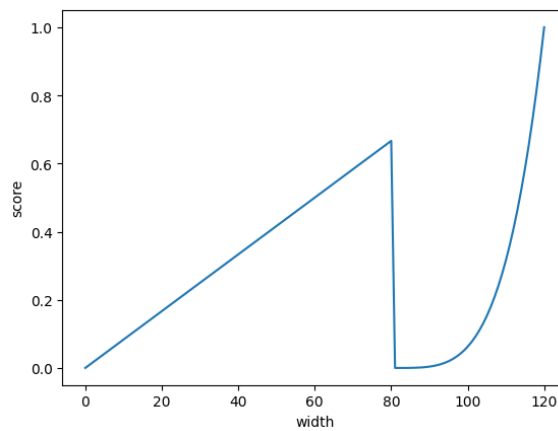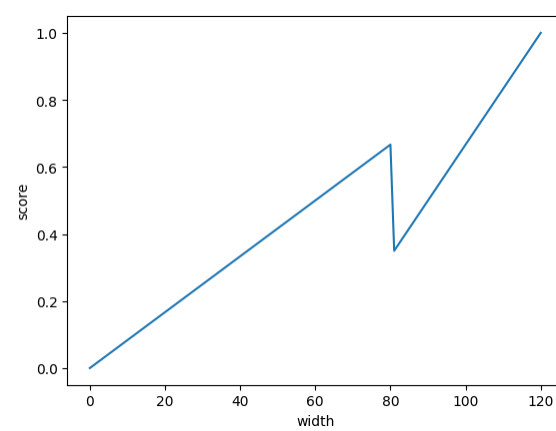
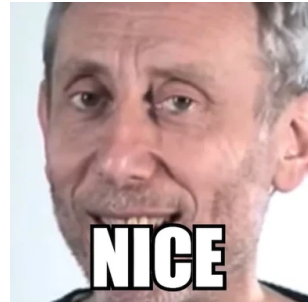The point where $score(w_c) == score(point)$
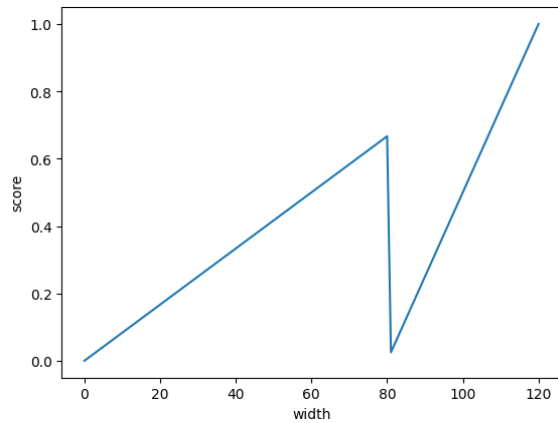
Linear
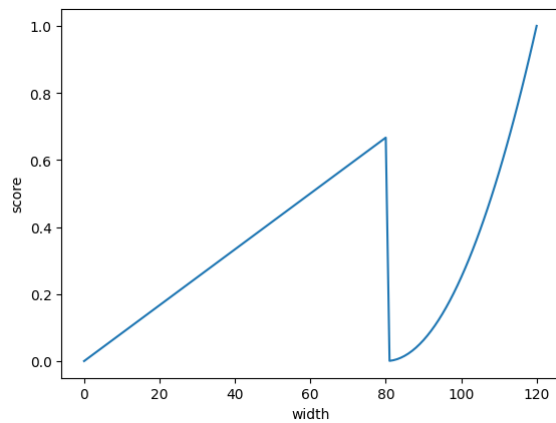
Polynomial p=2

Polynomial p=4

Sublinear a = .5

Results

Only changing the score heuristic here. All the rest stays constant.

Heuristic	Number of layers used	Comment
Fill rate	11464	
Wasted space	12155	Way worse yes, but should be combined with fill rate
Combination (Really need a better name for this)	10677	
Polynomial p=2	10761	
Polynomial p=4	11000	Anything superlinear likely worse
Linear a=0.2	10662	Parameter is tuned here. Negligible improvement
Linear a=0	10679	Should be equivalent to 'Combination'. strange?

Conclusion

The Combination heuristic is a simple but powerful way to score shape,space combinations. Everything is tested using non-overlapping open-spaces, with overlapping open-spaces things could change a lot.
This does require evaluating all shape, space combinations and making shapes (super-items) of all items as defined in the section about shapes. This is not a problem on the dataset I worked with (not at all actually). But might be a problem for other datasets who knows, I probably should calculate the time complexity