Introduction

Consider a set $\bar{J}$ of n rectangular items. Each item i has a width w_i and a height h_i, $i = \overline{1,n}$. The two-dimensional strip packing problem (2SP) consists in packing all the items in a strip of width W and infinite height. The objective is to minimize the total height used to pack the items without overlapping and going outside the strip. The orientation of items is fixed, that is, they cannot be rotated.

The input data is stored in "rectangles.csv" file. The width of the strip W and the number of items n are specified in the first line of the file. Each line starting with the second contains the width w_i and the height h_i of the rectangle i. For clarity, the data in the input file is organized as follows:

$$W, n$$
$$w_1, h_1$$
$$w_2, h_2$$
$$\vdots$$
$$w_n, h_n$$

The program shows the optimal packing for the listed items and a diagram of the solution. An example of output generated by the program is presented in Figure 1.

```
The execution time is: 0.4444587230682373 seconds
The strip width is: 15
The optimal packing height is: 10
The optimal packing:
1. rectangle of size (7, 2) is packed at position (0, 8)
2. rectangle of size (8, 4) is packed at position (7, 6)
3. rectangle of size (5, 3) is packed at position (2, 5)
4. rectangle of size (2, 4) is packed at position (0, 3)
5. rectangle of size (6, 6) is packed at position (8, 0)
6. rectangle of size (3, 5) is packed at position (5, 0)
7. rectangle of size (5, 3) is packed at position (0, 0)
```

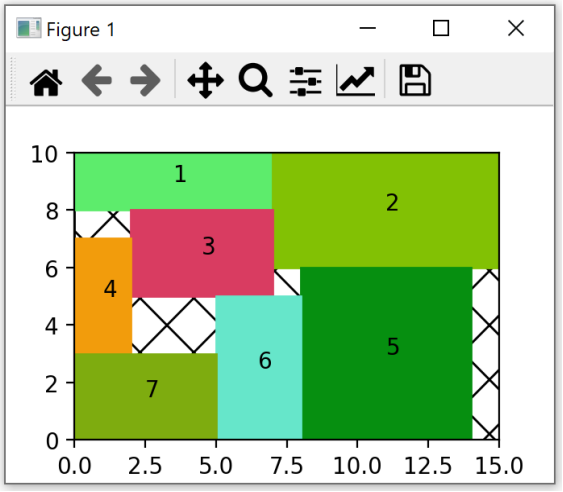

Figure 1.

Description of algorithm

This problem is NP-hard in the strong sense [1]. So, for obtaining the optimal solution of 2SP we will use the enumerative approach proposed in [2].

We designate the bottom left corner of the strip as the origin of the xy-plane, letting the x-axis be the direction of the width of the strip, and the y-axis be the direction of the height. We represent the location of each rectangle i in the strip by the coordinate (x_i, y_i) of its bottom left corner.

The following properties from [2] are important for the sequel:

Property 1. Any packing of a strip can be replaced by an equivalent packing where no item may be moved leftward or downward.

Property 1. An ordering of the items in an optimal solution exists such that, if $i < j$, then $x_i + w_i \leq x_j$ or $y_i + h_i \leq y_j$.

It follows that the enumeration scheme for solving 2SP can be limited to generate only solutions that satisfy Property 1 and Property 2.

An important consequence of these properties is that at any decision node, where the items of I are already packed and item $j \in \bar{J} \setminus I$ is selected for branching, j may be placed only at points p such that no item of I has some part right of p, or above p (here, $I \subseteq \bar{J}$). In other words, the items of I define an "envelope" separating the two regions where the items of $\bar{J} \setminus I$ may (resp. may not) be placed. More formally, a new item may be placed only in the set:

$$\hat{S}(I) = \{(x, y): \forall i \in I, x \geq x_i + w_i \text{ or } y \geq y_i + h_i\}.$$

The Figure 2 shows a set of already placed items and the corresponding envelope:

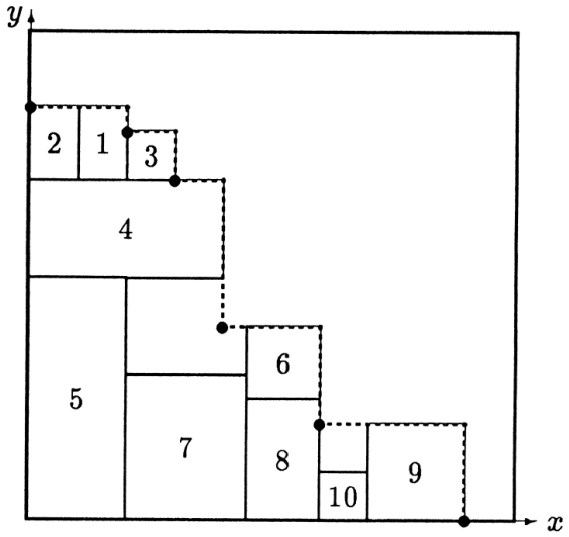

Figure 2.

Observe that, as a result of Property 1, the next item j may be placed only at points where the slope of the envelope changes from vertical to horizontal; such points are called in the following *corner points* of the envelope.

Given an item set I, it is quite easy to find the set $\hat{C}(I)$ of corner points of the envelope associated with the feasible region $\hat{S}(I)$. Following Property 2, let us order the items according to their end points $(x_j + w_j, y_j + h_j)$, so that the values of $y_j + h_j$ are nonincreasing, breaking ties by the largest value of $x_j + w_j$. The following algorithm for determining the corner points set consists of two phases. First, extreme items are found, i.e., items whose end point coincides with a point where the slope of the envelope changes from horizontal to vertical. In the second phase, corner points are defined at intersections between the lines leading out from end points of extreme items. Thus, we get the following algorithm:

Algorithm 2D-CORNERS:
begin
if $I = \emptyset$ **then** $\hat{C}(I) := \{(0, 0)\}$ and **return**;
comment: Phase 1 (identify the extreme items $e_1, \ldots, e_m$);
$\bar{x} := m := 0$;
for $j := 1$ **to** $|I|$ **do**
if $x_j + w_j > \bar{x}$
then $m := m + 1$; $e_m := j$; $\bar{x} := x_j + w_j$;
comment: Phase 2 (determine the corner points);
$\hat{C}(I) := \{(0, y_{e_1} + h_{e_1})\}$;
for $j := 2$ **to** m **do**
$\hat{C}(I) := \hat{C}(I) \cup \{(x_{e_{j-1}} + w_{e_{j-1}}, y_{e_j} + h_{e_j})\}$;
$\hat{C}(I) := \hat{C}(I) \cup \{(x_{e_m} + w_{e_m}, 0)\}$;
end.

Consider the example in Figure 2. The resulting corner points determined by the 2D-CORNERS algorithm are indicated by black dots.

The time complexity of 2D-CORNERS algorithm is $O(|I|)$, plus $O(|I|log|I|)$ for the initial item sorting, i.e., $O(|I|) + O(|I|log|I|)$.

We can now easily derive, in a recursive way, a branch and bound algorithm BBA4SP for finding the best filling of a strip using items from a given set $\bar{J}$. Initially, no item is placed, so $\hat{C}(\emptyset) = \{0,0\}$. At each iteration, given the set $I \subseteq \bar{J}$ of currently packed items, set $\hat{C}(I)$ is determined through 2D-CORNERS together with the corresponding height $H(I)$. If H_b is the height achieved by the current best packing, we may backtrack whenever $H(I) > H_b$ because we would not improve on H_b. If all items of $\bar{J}$ are already packed, we possibly update H_b and backtrack. Otherwise, for each position $(x, y) \in \hat{C}(I)$ and for each item $j \in \bar{J} \setminus I$, we assign the item to this position (provided that its end points do not exceed the strip limits) and call the procedure recursively.

Implementation

The program is implemented in Python 3.9. It is created a class *StripPacking* containing the method *generatesolutions* that recursively generates feasible solutions of 2SP in accordance to the BBA4SP algorithm. At each iteration of this method, it is used the *getcorners* method (it implements the 2D-CORNERS algorithm) for getting corner points of the set of currently packed items. Also, the *StripPacking* class contains the method *visualize* for printing the optimal packing for the listed rectangles and shows a diagram of the solution. The *StripPacking* class is stored in "sp.py" file. The program is launched through "main.py" file.

Results

Since 2SP is NP-hard, the exact algorithms perform a search over a tree of possible solutions, which does not exceed $n!^2$ nodes in the worst-case time complexity. The value $n!^2$ results from the fact that there are $n!$ insertion order permutations of items for packing and each ith item of the corresponding order can be placed at most in i corner points.

Now we establish the practical computing time for solving 2SP. For this purpose, we use instances with 7, 8 and 9 rectangles and 4 widths. Computing experiments are implemented on AMD PC with Ryzen 5600u 2.3 GHz (up to 4.2 GHz in Turbo mode) CPU.

Table 1.

Nr. of rectangles	Average computing time, seconds			
	Width=10	Width=15	Width=20	Width=25
7	0.21	0.56	0.33	0.24
8	1.96	9.17	4.23	3.82
9	39.18	129.23	30.74	26.48

As we see in the Table 1, the average computing time per instance with at most 7 rectangles does not exceed 1 second.

The worst average computing time per instance with 8 rectangles is obtained for width 15 and it is equal to 9.17 seconds. The hardest instance (9 rectangles, width 15) requires almost average 130 seconds of computing time.

Note that too wide or too narrow width essentially reduces the number of potential corner points and, as a result, the execution time becomes smaller.

Conclusion

The developed program is very time consuming and can be practically used only for solving relatively small instances of 2SP (with at most 10-11 rectangles when there is no tight constraint on time). The reasons for such unsatisfied results are: NP-hardness of 2SP, exact algorithm based on enumerative approach, Python interpreted language, and unperfect implementation of the above-described algorithm. The first reason is intractable. The second can be overcome by using different heuristics but in such circumstances obtaining the optimal solution is not guaranteed. The third can be beaten by using compiled programming language like C++, and the last requires more time allocated for this purpose.

References

1. Martello, S., Monaci, M., & Vigo, D. (2003). *An Exact Approach to the Strip-Packing Problem*. INFORMS Journal on Computing, 15(3), 310–319. doi:10.1287/ijoc.15.3.310.16082
2. Martello, S., Pisinger, D., & Vigo, D. (2000). *The Three-Dimensional Bin Packing Problem. Operations Research, 48(2), 256–267.* doi:10.1287/opre.48.2.256.12386