Two-dimensional cutting-stock problem

R. Bocquillon, N. Monmarché, R. Raveaux, V. T'kindt

Polytech Tours, University of Tours

December 17, 2024

The project has to be made in autonomy: the only allowed help are the documents issued from the Teaching Unit "New Solution Approaches for AI". You can get some additional help from the Internet as long as you are only looking for new knowledge: any code taken from the Internet whatever it comes from (git repositories, generative AI like ChatGPT, *etc.*) is prohibited. This project evaluates your capacity to produce models, algorithms and programs.

1 Forewords

In this tutored project, you work on a strongly NP-hard cutting-stock problem for which you are asked to propose solution algorithms. You will have to dig into the set of techniques and algorithms that have been presented to you in this Teaching Unit (TU), in order to conceive an algorithm as efficient as possible.

At the end of the project, several outputs have to be produced, including a report. This one must account for the work you have done: it is part of the evaluation. To achieve this project you have 24h of TPs which are, indicatively, split as follows:

1. problem analysis, choice of models and algorithms: 4h,
2. algorithm development: 10h,
3. experimental evaluation and algorithm improvement: 10h.

In this tentative planning, each step includes time for writing your report. You work in groups of three.

2 Problem presentation

Suppose you are given a set of big boards in which you have to cut some items. These boards can be of wood, glass, *etc.*. For sake of generality, we will call them *jumbos*. Figure 1 presents jumbos while Figure 2 pictures a jumbo cutting plan. The blue rectangles correspond to items to cut and the gray parts are waste.

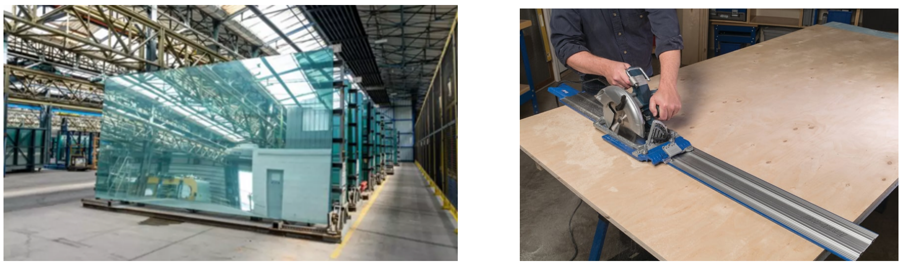
Figure 1: Jumbos: Glass or wood boards

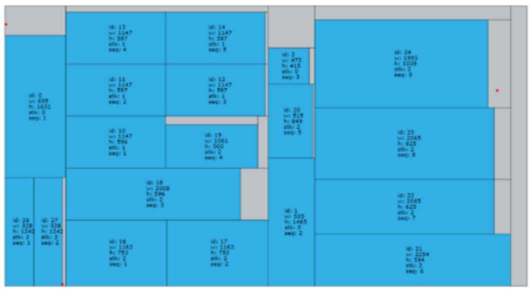
Figure 2: A cutting plan

More formally, the problem is defined by a set of input data and constraints. Let's start with the input data:

- a set $\mathcal{B}$ of m jumbos; each jumbo $b \in \mathcal{B}$ is defined by a width W_b and a height H_b;
- a set $\mathcal{I}$ of n items; each item i is defined by a width w_i and a height h_i; it can be cut on any jumbo in any 90-degree rotation.

The main constraint of this problem relies on the way the jumbo can be cut. Only guillotine cuts can be done: it is either an horizontal or a vertical cut done from one end to the other one. Consequently, we cannot make angles while cutting a jumbo. Figure 3 shows, on the left, a cutting plan that does not answer the guillotine cut constraint and, on the right, a cutting plane compatible with this constraint.

Given an instance of the problem, you must propose an algorithm that computes a cutting plan of the jumbos so as to minimize the total waste. Notice that the total waste is defined as the lost parts of all but the last jumbo: for this one, we consider that the *first residual* is not lost and so does not count in the waste. For a cutting plane s, let us denoted by $\mathcal{B}_s \subseteq \mathcal{B}$ the set of jumbos that are cut. Let $p_b(s)$ be the waste on jumbo $b \in \mathcal{B}_s$:

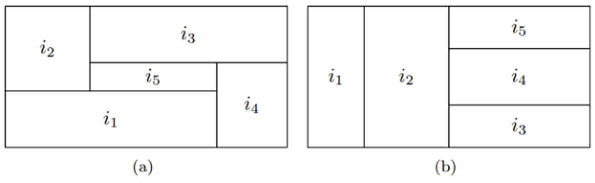

Figure 3: Cutting plans: the guillotine cut

$$p_b(s) = H_b W_b - \textstyle\sum_{j \in s_b} w_j h_j,$$

with $s_b \in s$ the set of item cuts done on b. Thus, the objective function to minimize is:

$$f(s) = \sum_{b \in \mathcal{B}} p_b(s) - r_{last},$$

with r_{last} the surface of the residual on the last cut jumbo (Figure 4). The surface r_{last} is obtained by multiplying the widght and the height of the unused part after the *first* cut on the jumbo.

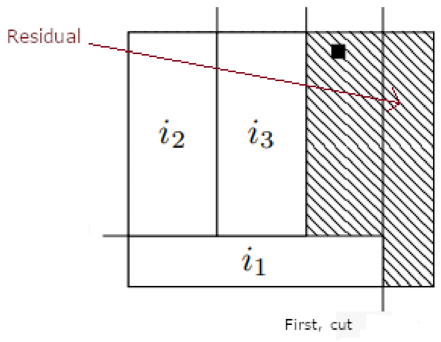

Figure 4: Residual on the last jumbo

The last important point to mention relates to the way cuts are described. We make the assumption that, for any piece of jumbo, the left-down corner is the origin of offset 0. A cut is defined by a cutting direction and an offset from the origin: V indicates a vertical cut and H an horizontal one. Consider Figure 4 with a jumbo b of size $W_b = 400$ and $H_b = 400$ (all distances are given in mm), then the first cut is, e.g., $V320$ (assuming that the first cut is at a distance of 320, starting from the left-down corner).

Also notice that no jumbo rotation is allowed. This is useless to describe the cutting of a series of items on a jumbo.

It follows that, for a jumbo, a series of cuts can be represented by a binary tree which leave nodes are the items: this is discussed in section 3.4.

3 Technical constraints

3.1 Creation of your algorithm

To achieve this project, you have to answer a set of technical/pedagogical constraints. The teachings in this TU are split into three major parts: (1) Search tree based methods and CP, (2) Operations Research and Machine Learning, (3) Bio-inspired methods.**Your solution algorithm have to make use of, at least, two of these three parts**. For example, you can hybridize techniques/elements/algorithms from part (1) and part (3) or, use techniques seen part (2) to improve algorithms from part (1) or (3)... Feel free to create your own mixture! Notice, that you will have to clearly and fully explain, in your report, how you proceed.

3.2 Evaluation of your project

With respect to the evaluation of your project, part of your mark will result from the evaluation of your algorithm on a database of unknown instances. A first test set will be given to you so that you can tune and evaluate your algorithm. **The final evaluation will be done by on instances you do not know.** How will be evaluated your algorithm on that set of unknown instances? Two measures will be taken into account:

(C1) Quality of the computed solutions: on each instance, lower is be the value of f for your solution and the better is your algorithm,

(C2) Computational time: on each instance, lower is the CPU time required to compute your solution and the better is your algorithm.

Obviously, criterion (C1) is the most important one, even if (C2) must not be neglected.

You will be given two set of instances: the set of **easy** instances and the set of **hard** instances. The former contains small size instances with few items while the latter contains much bigger instances. Notice, that the set of unknown instances used for evaluation are as hard as the ones in the set **hard**.

3.3 Software Engineering

Your codes must be compliant with the good practices of software engineering. You are free to decide about the languages you want to use along the project to develop or tune your algorithm. However, in order for us to evaluate your algorithm we impose the following constraints:

- Operating system: you will work on a Virtual Machine that will be provided to you and running under a Linux OS. All the basic tools and software that you need will be available, even though you can add some if necessary.
- Test automation: with your program you must provide a script, **named run_all.sh**, that takes two parameters. The first one is a repository that contains a set of instances to solve (one instance per file). So, your script runs your algorithm on each of the instances in that first repository and writes the produced output files

in the repository indicated as the second parameter of the script. One output file describes a single solution for a single instance file.

- Inputs and Outputs: File formats are imposed both in the instance file and the solution file. This is the matter of section 3.4. However, for any instance file named XXXX, the corresponding output file must be named XXXX.out.

To develop your algorithms you are allowed to use any of the libraries or code used during the courses of the TU. If you want to use any other library you must get before the agreement of one of the supervisors of the project (Ronan, Nicolas or Vincent).

Keep in mind that your evaluation will be based on the originality and quality of the algorithm you create. We evaluate your production not your capability of copy/paste-ing codes from the Internet.

3.4 Inputs and outputs

In this section we describe the json file formats are imposed for the input and output files.

Input file. We provide below a snippet of an instance file that belongs to the set **Easy**.

```
{
   "cut-width": 0,
   "items": [
      {
         "id": 0,
         "name": "name_0",
         "size": {
            "height": 1578,
            "width": 758
         },
         "nb": 1
      },
      {
         "id": 1,
         "name": "name_1",
         "size": {
            "height": 738,
            "width": 1550
         },
         "nb": 1
      },
      ...
   ],
   "jumbos": [
      {
         "id": 0,
         "name": "jumbo_0",
         "size": {
            "height": 5379,
            "width": 3703
         },
         "nb": 15
      },
      {
         "id": 1,
         "name": "jumbo_1",
         "size": {
            "height": 7667,
            "width": 2884
         },
         "nb": 15
      },
      ...
   ]
}
```

Each item is described by the following features:

- `id`: an unique identification number,
- `name`: a string containing the name of the item,
- `size`: this field contains the `height` and `width` of the item (in millimeters),
- `nb`: the number of such items in the instance.

Each jumbo is described by:

- `id`: an unique identification number,
- `name`: a string containing the name of the jumbo,
- `size`: this field contains the `height` and `width` of the jumbo (in millimeters),
- `nb`: the number of such jumbos in the instance.

Output file. We now provide the file format of an output file. The described solution applies to the instance previously introduced.

```
{
  "op_list":[
    {
      "jumbo_id":0,
      "cut-tree":
      {
        "dir-cut":"vertical",
        "offset":758,
        "left":
        {
          "dir-cut":"horizontal",
          "offset":1578,
          "left":
```

```
    {
        "item_id":0
    },
    "right":
    {
      "dir-cut":"vertical",
      "offset":738,
      "left":
      {
        "dir-cut":"horizontal",
        "offset":1550,
        "left":
        {
          "item_id":1
        },
        "right":
        {
          "dir-cut":"horizontal",
          "offset":581,
          "left":
          {
            "dir-cut":"vertical",
            "offset":1550,
            "left":
            {
              "item_id":2
            }
          }
        }
      }
    }
  },
  "right":
  {
    "dir-cut":"vertical",
    "offset":1396,
    "left":
    {
      "dir-cut":"horizontal",
      "offset":781,
      "left":
      {
        "item_id":3
      }
    },
    "rigth":{
      "item_id":-1
    }
  }
}
```

```
    },
    {
      "jumbo_id":1,
      "cut-tree":
      {
        "dir-cut":"vertical",
        "offset":320,
        "left":
        {
          "item_id":4,
          "dir-cut":"horizontal",
          "offset":-1
        }
      }
    }
 ]
}
```

The format of the json file follows a hierarchical framework: the cutting plan consists in a list of operations (`op_list`) on a given list of jumbos. For each given jumbo a list of cuts is provided as a binary tree (`cut_tree`): a cut is done at an `offset` distance from the left side (vertical) or the bottom side (horizontal). Each cut produces two pieces (`left` and `right`) which can be at their turn, cut into pieces. If one side (left/right) is not used, it is not mandatory to let it in the list. if a piece corresponds to an item then `item_id` must be provided provided.

If `offset` is -1 or not given then the piece is not cut anymore.

In the example file, the last jumbo (`jumbo_id=1`) is cut only once vertically and the left part obtained is linked to item id 4 (dimensions of item 4 are given in data input file). The right part is not used and described in the file (but still exist in real life !). In item 4 description, a `dir_cut` is given but not used since no `left` or `right` are given. Also `offset` at -1 means "no cut" (not mandatory).

Figure 5 shows the process of cutting successive pieces, starting from one initial jumbo.

Notice that Figure 5 not only describes the structure of the json output file, but also the way a solution can be stored in memory by means of a binary tree.

4 Outputs of the project

At the end of the project, you have to provide several elements:

- a report explaining the way you encoded a solution, the algorithms and techniques your used (and give justification of them), the results you had on the test set of instances,
- the script and executable files as requested in Section 3.3 to enable testing on the set of unknown instances,
- all the source files of your program.

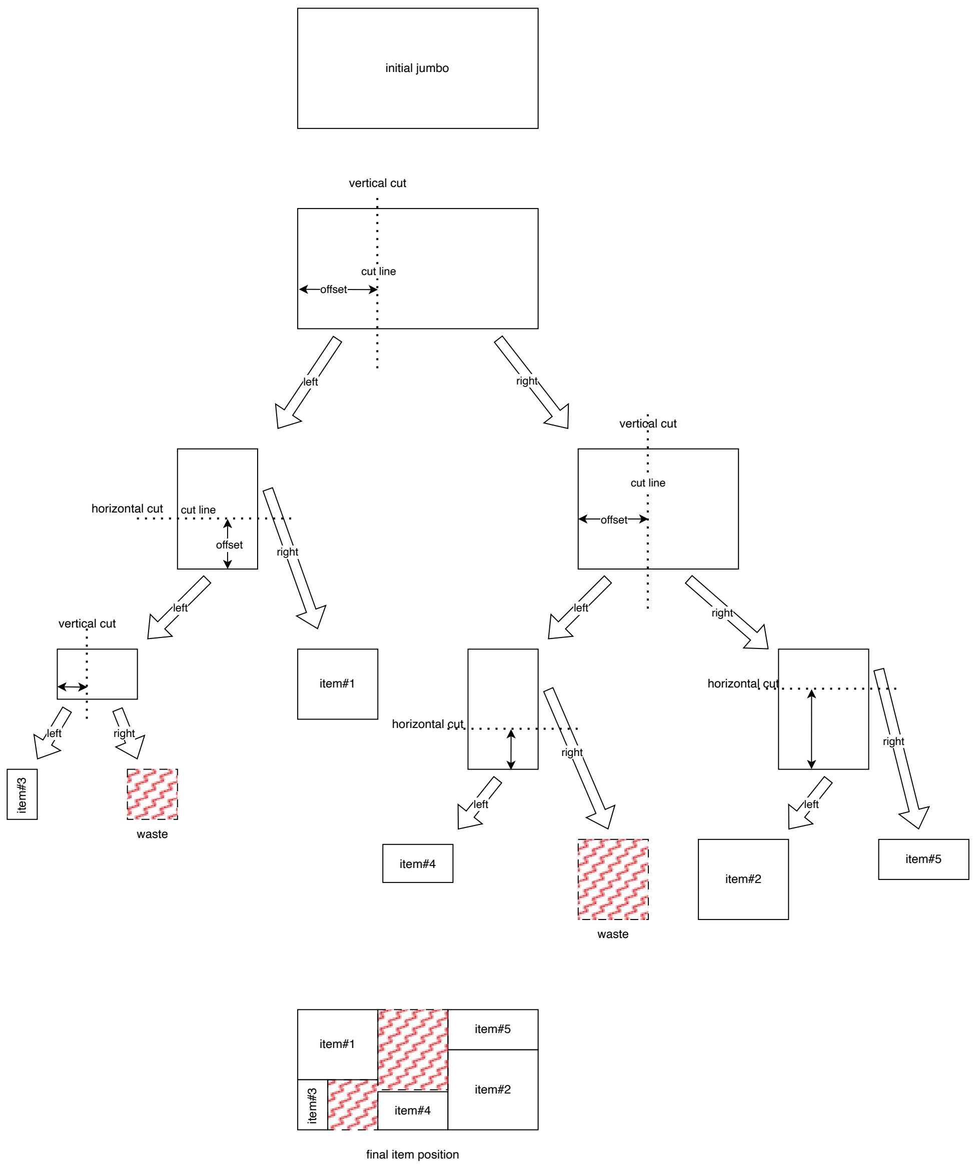

Figure 5: A tree of cuts