Combinatorial Problem Solving (CPS)

Project: Box Wrapping.

Updated: February 15, 2018

1 Description of the Problem

It is a common practice that banks reward their most loyal clients with presents, which are typically wrapped in costly corporative paper full of logos of the bank.

Imagine you work in one of such banks, which is going through a severe crisis. Since it is detected that the expenses on wrapping paper are excessive and therefore should be reduced, you (as one of the most competent computer engineers in the company) are designated responsible for solving the Box Wrapping Problem (BWP): given a list of boxes to be wrapped, and given a huge roll of paper of a certain width, you have to decide how to cut off the pieces of paper for wrapping the boxes so that the length of roll that is used is minimized.

The following considerations are made:

- Each box is described by the dimensions of the rectangular piece of paper needed to wrap it.
- There may be several identical boxes.
- The rectangles of paper may be rotated when cutting off if needed.
- The roll has infinite length.

2 Format of the Instances

This section describes the format in which instances of BWP are written (Section 2.1), as well as the expected format for the corresponding solutions (Section 2.2).

2.1 Input Format

An instance of BWP is a text file consisting of several lines of integer values. The first line contains W, the width of the roll of paper. Each of the next lines has this format: a number n_i, followed by numbers x_i and y_i, meaning that there are n_i boxes that need a rectangular piece of paper of dimensions $x_i \times y_i$. It is ensured that $1 \leq W \leq 11$, $1 \leq n_i \leq 6$, $\sum n_i \leq 13$, $1 \leq x_i \leq \min(y_i, W)$, $1 \leq y_i \leq 10$. Moreover, if $i \neq j$ then $(x_i, y_i) \neq (x_j, y_j)$.

The file containing such an instance is named `bwp_`W`_`N`_`k`.in`, where W has the above meaning, N is the total number of boxes, i.e., $N = \sum n_i$, and k is a unique integer identifier.

As an example, a file `bwp_4_5_1.in` could contain the following data:

```
4
3   1 3
1   3 3
1   1 1
```

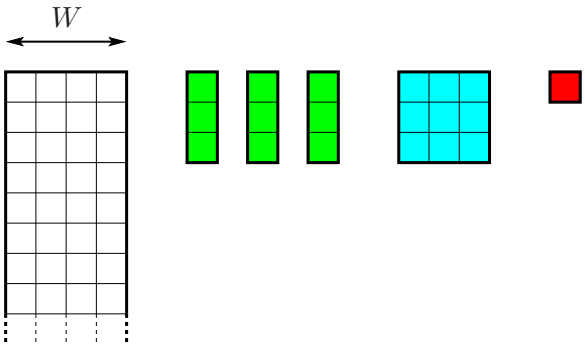

Figure 1: Graphical representation of the instance.

In this instance the roll of paper has width 4, and we have to cut off 5 rectangles of paper in total: 3 of dimensions 1×3, 1 of dimensions 3×3, and finally 1 of dimensions 1×1.

2.2 Output Format

Output files only contain integer values. They should have the following format:

- The first line should be L, the length of roll of paper that needs to be consumed.
- For each of the N boxes, there should be a line, with the following format: first two numbers x_i^{tl} and y_i^{tl}, meaning the coordinates of the top left corner for the corresponding piece of paper; and then two numbers x_i^{br} and y_i^{br}, meaning the coordinates of the bottom right corner.

  The roll of paper, which has width W and length L, has to be understood as a grid of points. These points are described with a Cartesian coordinate system, where the x-axis coordinates take values between 0 and $W - 1$, and the y-axis coordinates take values between 0 and $L - 1$.

The output file corresponding to an instance `bwp_`W`_`N`_`k`.in` should be named `bwp_`W`_`N`_`k`_`t`.out`, where t is an identifier of the problem solving technology that has been used: `CP` for Constraint Programming, `LP` for Linear Programming, and `SAT` for Propositional Satisfiability.

For instance, if the arrangement in Figure 1 (which is clearly optimal) had been obtained with CP, then the output file would be named `bwp_4_5_1.CP.out`, and its contents could be:

```
5
0 0   2 2
3 0   3 2
0 3   2 3
0 4   2 4
3 3   3 3
```

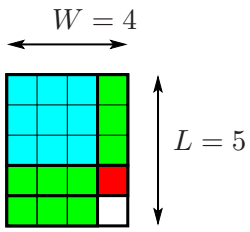

Figure 2: Graphical representation of the solution.

Note that, in this solution, one of the rectangles 1×3 is laid vertically, while the other two are laid horizontally.

3 Project

The purpose of this project is to model and solve BWP with the three problem solving technologies considered in the course: constraint programming (CP), linear programming (LP) and propositional satisfiability (SAT).

For the elaboration of the project, in addition to this document, students will be provided with the following materials:

- a suite of *problem instances* (in the format specified in Section 2.1).
- a *checker* that, given a problem instance and an output file describing a solution (following the format given in Section 2.2), makes a sanity check. If possible, the solution is also plotted on the console.

  Take into account that **the checker does not guarantee that the solution is correct**. It only makes lightweight checks aimed at debugging. In particular, there may be better solutions with shorter paper rolls.

These materials will be available at the webpage of the course:

`http://www.cs.upc.edu/~erodri/cps.html`

Solving processes that exceed a time limit of 120 seconds should be aborted. Bear in mind that some of the instances might be difficult to solve within this time limit on a personal computer. Hence, it is not strictly necessary to succeed in solving all instances with all technologies to pass the project. However, you are encouraged to solve as many instances as possible.

The project has three deadlines, one for each problem solving technology:

- **CP:** 7 May.
- **LP:** 28 May.
- **SAT:** 18 Jun.

For each technology, a `tgz` or `zip` compressed archive should be delivered via Racó (`https://raco.fib.upc.edu`) with the following contents:

- a directory `out` with the output files of those instances that could be solved successfully.

- a directory `src` with all the source code (C++, scripts, `Makefile`, etc.) used to solve the problem, together with a `README` file with basic instructions for compiling and executing (so that results can be reproduced).
- a document in PDF format describing the variables and constraints that were used in your model, as well as any remarks or comments you consider appropriate.