CPS - ASSIGNMENT 3

SAT PROGRAMMING

June 18, 2018

Kymry Burwell
UPC

Contents

1 VARIABLE DEFINITION

For this problem, I envisioned the paper roll as a set of boolean grid points. Each box either occupies or does not occupy a grid point. The following variables hold the input data.

- **int n:** number of boxes
- **int xSize:** width of the paper roll
- **int ySize:** height (length) of the paper roll
- **vector<int> xDim:** x-dimensions of all boxes
- **vector<int> yDim:** y-dimensions of all boxes
- **int MIN_LENGTH:** Theoretical minimum length of paper roll. The paper roll may be longer, but can never be shorter.
- **int MAX_LENGTH:** Maximum length of paper roll. The paper roll will never need to be longer than this.
- **int xStart:** Starting integer for auxiliary variables used to allow box rotation (i.e. to be placed horizontally or vertically).
- **int xtlStart:** Starting integer for top-left variables.

The following three main variables are used to place the boxes on the paper roll.

- **literal x:** This literal is true for each grid position the box occupies.
- **literal xtl:** This literal marks the top-left starting coordinate the box occupies.
- **literal z:** This literal designates whether the box is placed horizontally or vertically on the paper roll.

The **x** literals are integers that begin at **1** and end at **n**×**xSize**×**ySize**, meaning each box has one unique **x** literal for each grid position. The same is true for the top-left literal, **xtl**, however, these literals begin at **n**×**xSize**×**ySize+1** and end at **n**×**xSize**×**ySize**×**2**.

As an example, if there are *2* boxes of size *1x2* and the grid is size *2x4*, box 1 will have **x = {1,2,3,4,5,6,7,8}** and box 2 will have **x = {9,10,11,12,13,14,15,16}**. Furthermore, box 1 will have **xtl = {17,18,19,20,21,22,23,24}** and box 2 will have **xtl = {25,26,27,28,29,30,31,32}**.

These are visualized below.

1	2
3	4
5	6
7	8

(a) box 1: x

9	10
11	12
13	14
15	16

(b) box 2: x

17	18
19	20
21	22
23	24

(c) box 1: xtl

25	26
27	28
29	30
31	32

(d) box 2: xtl

The **z** literal begins at **n**×**xSize**×**ySize**×**2+1** and ends at **n**×**xSize**×**ySize**×**2+n**. Following the example above, box 1 has **z = 33** and box 2 has **z = 34**.

2 CONSTRAINTS

(1a) Each box has at least 1 top-left coordinate (xtl) set to true. This is a basic ALO constraint.

(1b) Each box has at most 1 top-left coordinate (xtl) set to true. This is a basic AMO constraint. These first two constraints simply ensure that the box can be placed in only one position by forcing it to have only one starting, top-left position.

(2) Each grid point is occupied by at most one box. This constraint iterates over each grid point and creates a basic AMO constraint using each box. Continuing with the 2 box example from above, this would create a constraint for the first grid position, (0,0), by using literal 1 from box 1 and literal 9 from box 2.

(3) Each box may be placed vertically. This constraint iterates over all grid points and creates clauses. It first checks if the box is able to be placed vertically in the current grid point (i.e. the dimensions of the paper roll are adhered to). If it can, multiple clauses are created of the form **(¬ xtl ∨ x)**. It creates a clause like this for each grid position the box may occupy. Essentially, it is saying that if the top-left coordinate of the box is set to true, then all grid points that the box will occupy if placed there must also be set to true. In addition, this constraint checks

if the the box can also be placed horizontally. If it can, it adds the auxiliary variable, **z**, to the set of clauses. The new clauses will then look like **(¬ xtl ∨ x ∨ z)**. Continuing the two box example from above, if box 1 is of size 1x2 and is being placed vertically in the (0,0) grid position, two clauses will be created: **(¬ 17 ∨ 1 ∨ 33)** and **(¬ 17 ∨ 3 ∨ 33)**.

(4) Each box may be placed horizontally. This is essentially the same as constraint 3. The only differences are that the box is placed horizontally and instead of using a positive **z** literal in the constraints, a negative (¬**z**) is used instead.

(5) Boxes cannot be placed both horizontally and vertically. This constraint implicitly utilizes the auxiliary **z** variable and creates clauses that ensure each box is placed in only one direction. Each clause has 2 literals, and each literal corresponds to grid points that are diagonally opposite of each other. Using the same example above, if box 1 is placed in the (0,0) coordinate, this constraint will create the following clause: **(¬ 2 ∨ ¬ 3)**. Along with the **z** variable, this ensures the box is only placed either horizontally or vertically, but never both.

(6) Box only occupies the correct grid points. Basically, this constraint ensures that if a box is placed in a certain position, no other grid points can be set to true for that box. For example, if box had **xtl** = 17 set to true, meaning the top-left coordinate was in position (0,0) then 4, 5, 6, 7, and 8 **x** variables are forced to be false. It works by creating clauses of the form **(¬ xtl ∨ ¬x)**. For example, for **xtl** = 17 , the following clauses would be created: **(¬ 17 ∨ ¬ 4)**, **(¬ 17∨ ¬ 5)** ... **(¬ 17 ∨ ¬8)**.

*Note - I debated whether or not I should include this constraint, as it is possible to create a valid CNF without it. However, adding the constraint made the post-processing much easier to code, so I left it in.

(7) Top-left coordinate (xtl) cannot be in unfeasible position. This constraint simply creates single literal clauses that ensure the xtl variable is false for unfeasible positions. For example, box 1 would have the following clause: **(¬ 24)**.

*Note - I had considered simply not creating xtl literals for unfeasible positions during the initial creation, but decided against it due to the way I iteratively decrease the paper roll size as the optimal solution is searched for.

(8) Symmetry breaking - Boxes of same size must be placed in increasing grid positions. This constraint seems to help a bit for smaller instances, but doesn't decrease the running time much for larger instances. It creates 2 literal clauses using the **xtl** variables of boxes of the same size. The clauses are of the form:**(¬ xtl (box 1) ∨ ¬ xtl (box 2))**. It does so by iterating through all grid points and creating multiple clauses where the **xtl** of box 1 remains constant and the **xtl** of box 2 changes. And example helps illustrate. For box 1 with xtl = 19, the following clauses will be created: **(¬ 19∨ ¬ 27)**, **(¬ 19∨ ¬ 26)**, **(¬ 19∨ ¬ 25)**. In doing this for each xtl, it ensures that box 2 will never be placed in a lower position than box 1.

3 OPTIMAL SOLUTION SEARCH

The program begins by calculating the theoretical minimum and maximum paper roll lengths. It determines the maximum length by adding the larger of the two dimensions of every box. Essentially, if every box needed to be placed on top of each other, the initial grid size would be perfectly adequate. The minimum is determined by summing the size of all boxes and diving by the paper roll width.

The program then creates an initial CNF using the maximum grid size. Next, a solution is searched for. If found, the grid length is decreased by 1. This is done by adding single literal clauses for each box that ensures the **x** variable cannot occupy the last row of the grid. Using the same example above, the initial grid length is 4. If the length was decreased to 3, the following clauses would be created for box 1: **(¬ 7)** and **(¬ 8)**. After the new clauses are created for each box, they are added to the existing CNF and a solution is again searched for.

The above process continues, decreasing the grid size by 1 each time, until no solution can be found. The output of the program is the optimal solution consisting of the minimum paper roll length.