A MILP for the arrangement of rectangular elements in a rectangular space

Pierluigi Mansueto
E-mail address
pierluigi.mansueto@stud.unifi.it

1. Introduction

In this paper, an overview of a MILP (Mixed Integer Linear Problem) for the arrangement of rectangular elements in a rectangular space is given, exposing firstly the definition of variables, constraints and objective function and lastly the description of other undertaken roads for the problem definition. The main objective of this MILP is to maximize the scaling factors (one scaling factor for each element), in order to fill the area of a given rectangular space as much as possible. The aspect ratio of each element must be unchanged after the execution of the algorithm that solves the MILP.

2. Data and variables

The data are all the needed information to execute the optimization algorithm.

They are the following:

- N: the number of elements;
- W, H: the dimensions of the rectangular space where the elements will be collocated;
- $\forall i \in [1, N] \quad w_{0i}, h_{0i}$: the initial dimensions of the elements, the aspect ratio is defined by these ones ($a_i = w_{0i}/h_{0i}$);
- $\forall i \in [1, N] \quad q_i$: the declared weigth of the elements, the greater the weight the greater the element importance and scaling factor will be.

The MILP variables related to an element are:

$$\forall i \in [1, N] \quad ElementVariables = [x_i, y_i, s_i] \tag{1}$$

The coordinates x_i and y_i describe the position of the bottom-left corner of the element i (the Cartesian diagram origin is placed on the bottom-left corner of the rectangular space). The value s_i represents the scaling factor to apply to the initial dimensions of the element i. All these variables are continuous.

In addition, there are also other variables to describe and they are related to the No Overlap constraints. To avoid confusion, they will be declared later on this paper.

3. Constraints

There are four categories of constraints:

- Containment constraints;
- Fill constraints;
- No Overlap constraints;
- Trivial constraints.

3.1. Containment constraints

These constraints are defined to make sure that each element is located in its entirety inside the established rectangular space. These constraints must be defined both for the width and for the height of the element. Below the constraints are declared as mathematical inequalities.

$$x_i + w_{0i} * s_i \leq W \qquad \forall i \in [1, N] \tag{2}$$

$$y_i + h_{0i} * s_i \leq H \qquad \forall i \in [1, N] \tag{3}$$

3.2. Fill constraints

The declaration of the element importance introduces some constraints about how the elements should fill the rectangular space. Indeed, if an element A has a bigger importance value than an other one B, then a bigger scaling factor should be assigned to the element A. How much bigger should this scaling factor be? This question depends from the assigned importance values of the two considered elements.

Below there are the mathematical equalities.

$$s_i = (q_i/q_j) * s_j \qquad \forall i, j \in [1, N] : q_i \geq q_j \tag{4}$$

It is noteworthy that two elements with the same weigth value should have the same scaling factor after the optimization algorithm execution.

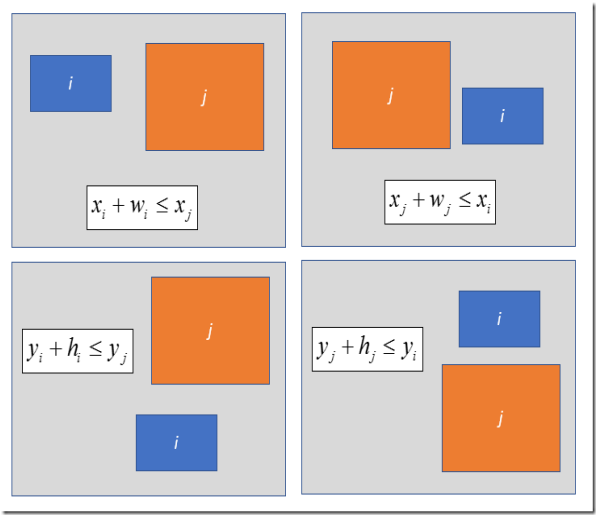

Figure 1. No Overlap Image [1]

3.3. No Overlap constraints

Each element must not overlap any other one in the space. Consequently a definition of this constraint type is needed.

In the image 1 there are the four conditions to have no overlap between two elements[1]. Below the four conditions are defined basing on the paper context.

$$\begin{aligned} x_i + w_{0i} * s_i - x_j \le 0 \quad or \\ -x_i + x_j + w_{0j} * s_j \le 0 \quad or \\ y_i + h_{0i} * s_i - y_j \le 0 \quad or \\ -y_i + y_j + h_{0j} * s_j \le 0 \\ \forall i \in [1, N] \quad \forall j > i \in [1, N] \end{aligned} \tag{5}$$

At least one of these conditions must be satisfied: consequently the logical operator between these conditions is the logical disjunction. This latter one is one of the most difficult logical operators to use on an optimization algorithm: it is not difficult to define it as a constraint but this operator leads to a modification of the optimization problem type. Indeed some binary variables are needed to define the four conditions as constraints.

$$\begin{aligned} x_i + w_{0i} * s_i - x_j - W * \delta_{ij1} \le 0 \\ -x_i + x_j + w_{0j} * s_j - W * \delta_{ij2} \le 0 \\ y_i + h_{0i} * s_i - y_j - H * \delta_{ij3} \le 0 \\ -y_i + y_j + h_{0j} * s_j - H * \delta_{ij4} \le 0 \\ \sum_{k=1}^{4} \delta_{ijk} \le 3 \\ \forall i \in [1, N] \quad \forall j > i \in [1, N] \end{aligned} \tag{6}$$

It is noteworthy that at least one binary variable must be 0 and, consequently, its related condition must be satisfied. So in this case the binary variables can reproduce perfectly the behavior of a logical disjunction.

These definitions lead to two consequences:

1. Introduction of new binary variables: this fact leads to a significant increment of the variables number;

$$\begin{aligned} \forall i \in [1, N] \quad \forall j > i \in [1, N] \\ BinaryVariables = [\delta_{ij1}, \delta_{ij2}, \delta_{ij3}, \delta_{ij4}] \end{aligned} \tag{7}$$

2. Modification of the optimization problem type: before the binary variables introduction the problem was a Linear Problem (LP), now it is a MILP. Therefore the methods to find a problem solution are very different.

3.4. Trivial constraints

The trivial constraints are the most simple constraints. They are related to the variables of each element.

$$x_i, y_i, s_i \ge 0 \quad \forall i \in [1, N] \tag{8}$$

4. Objective function

The objective function is easily intuible for what is said above. The function is the following:

$$f = \sum_{i=1}^{N} s_i \tag{9}$$

The objective is to maximize it.

5. Solving the problem

Unluckily, the binary variables cannot be relaxed to tranform the problem in a LP. Indeed, using a relaxation like transforming these variables in continuous ones, there is no way to respect the No Overlap constraints during the optimization algorithm execution. In addition, also after the execution, these relaxed variables cannot neither be transformed in binary ones (through methods like thesholding) because this action does not lead to the respect of No Overlap constraints. Therefore suitable methods for MILP problems must be used.

5.1. Branch & Bound algorithm

The most common and used method is Branch & Bound. Branch & Bound is a general technique for solving combinatorial optimization problems (problems with finite solution space) and is based on the decomposition of the original problem into simpler subproblems to be solved. [2]

The generic algorithm is the following (g is a bounding function that computes lower bounds of f on nodes of the search tree)[3]:

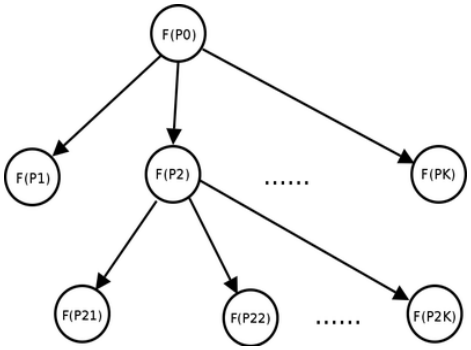

Figure 2. Example of a Branch & Bound tree [2]

1. Using a heuristic, find a solution x_h to the optimization problem. Store its value, $B = f(x_h)$. (If no heuristic is available, as in this case, set B to infinity.) B will denote the best solution found so far, and will be used as an upper bound on candidate solutions.
2. Initialize a queue to hold a partial solution with none of the variables of the problem assigned.
3. Loop until the queue is empty:
   (a) Take a node N off the queue.
   (b) If N represents a single candidate solution x and $f(x) < B$, then x is the best solution so far. Record it and set $B = f(x)$.
   (c) Else, branch on N to produce new nodes N_i. For each of these:
      i. If $g(N_i) > B$, do nothing; since the lower bound on this node is greater than the upper bound of the problem, it will never lead to the optimal solution, and can be discarded.
      ii. Else, store N_i on the queue.

5.2. CPLEX solver

The solver, used in this paper, that implements the Branch & Bound algorithm is CPLEX, powered by IBM ILOG. CPLEX is chosen among many other solvers because it has proved to be well configurable, fast and with a good parameter management. In addition, IBM provides a community edition for this solver while for Gurobi (another good solver) there is not any community edition.

Unluckily, during the execution the algorithm can be blocked on a local optimum: consequently the algorithm may take a long time to finish the execution. The only method to stop the execution is to set the parameter *timelimit* of the CPLEX variable. Therefore, the idea is that the user can decide how much time the algorithm execution must continue.

Lastly, CPLEX is characterized by the possibility to have more than one solution as output. This fact leads the user to have more different solutions (precisely 5 in this implementation, sorted in ascending order, basing on the area covered by the elements in the rectangular space) and to choose that one which is the best for him.

6. Undertaken roads

Below there are some met problems during the MILP definition.

- How should W and H be chosen? This question is not simple as it wants to seem. Surely, they cannot be designed as variables: in this case, the problem would have an infinite solution space because W and H would not be limited above and it would be difficult to choose an upper bound for them. Another idea is to compute the mean, the mode or the median of W and H using the initial dimensions of the elements involved on the problem. But all of these solutions lead to the choice of a scaling factor to apply to the mean, mode or median. In addition, these ideas do not always lead to convincing results. Therefore, these values (W and H) are chosen by the user. If the user does not declare W and H, the mean values are computed.

- Can the Fill constraints be relaxed? They seem to reduce too much the solution space. Unluckily, they are very important. Indeed, relaxing them means that the elements with greater importance value will be characterized by a greater scaling factor than other elements with a smaller importance value. How much greater should this scaling factor be? It does not matter. Now the Fill constraints are represented as mathematical inequalities.

$$s_i \geq (q_i/q_j) * s_j \qquad \forall i, j \in [1, N] : q_i \geq q_j \tag{10}$$

  However this idea leads to inconsistent results: one of the elements characterized by the greatest importance value would have the biggest possible scaling factor while the others would have the smallest possibile one. Therefore, the presence of the Fill constraints is crucial.

- Maximizing w and h of each element, not taking into account the aspect ratio: is it possible? This one is one of the first discussed problem for this paper. This idea leads to a similar problem definition. However the problem is more complex (without considering the aspect ratio the solution space is too much bigger): it happened that after a long time the algorithm did not find any local optima. Obviously this paper does not lead to the conclusion that a better definition of the problem would not produce better results.

7. Conclusion

This problem let a user make an arrangement of some elements in a rectangular space, taking into account the aspect ratio of each element. But there are other benefits: indeed, the user can decide the execution time of the algorithm and the algorithm let the user decide among more than one solution.

References

[1] Yet Another Math Programming Consultant, *Rectangles: no-overlap constraints*, http://yetanothermathprogrammingconsultant.blogspot.com/2017/07/rectangles-no-overlap-constraints.html

[2] Wikipedia, *Branch and Bound*, https://it.wikipedia.org/wiki/Branch_and_bound

[3] Wikipedia, *Branch and Bound*, https://en.wikipedia.org/wiki/Branch_and_bound