VAUCOURT Tom

TP 3 – Column generation

1)

a) Let's first create a table of size nbPatGen by nbPatterns called PatternGen (instead of Pattern in the wording of the problem).

- nbPatGen = 500 (we'll generate 500 patterns)
- nbPatterns = 21 (the number of different roll sizes)

We then implement the method proposed in the hint. It randomly add roll sizes to a pattern until it can no longer add anymore (the length is limited to 100 cm).

b) This model is a minimization problem on the number of times the patterns were used. The constraints are that each size of rolls are produced in sufficient quantity to satisfy the demand. After implanting it, we find a solution for which the optimal value of the objective function is 1435,6.

2) The way we implement the activation of a certain number of patterns among the 500 we generated is by creating a list of size nbPatGen called PatternAct. Its values are Boolean: 1 if the pattern is activated, 0 else.

We then multiply each term of the sum in the constraints by the corresponding value in this list. If a pattern is "deactivated", it won't contribute to the sum in the constraint and therefore the model won't use it in the solution it proposes.

Next, let's search for interesting patterns that are "deactivated". We'll compute the marginal cost of all the deactivated patterns to see which one is the most interesting. First, we compute the dual variables of the 21 constraints. The marginal cost of a pattern is the opposite of the sum of the product of this pattern values by the dual variables associated with each roll size.

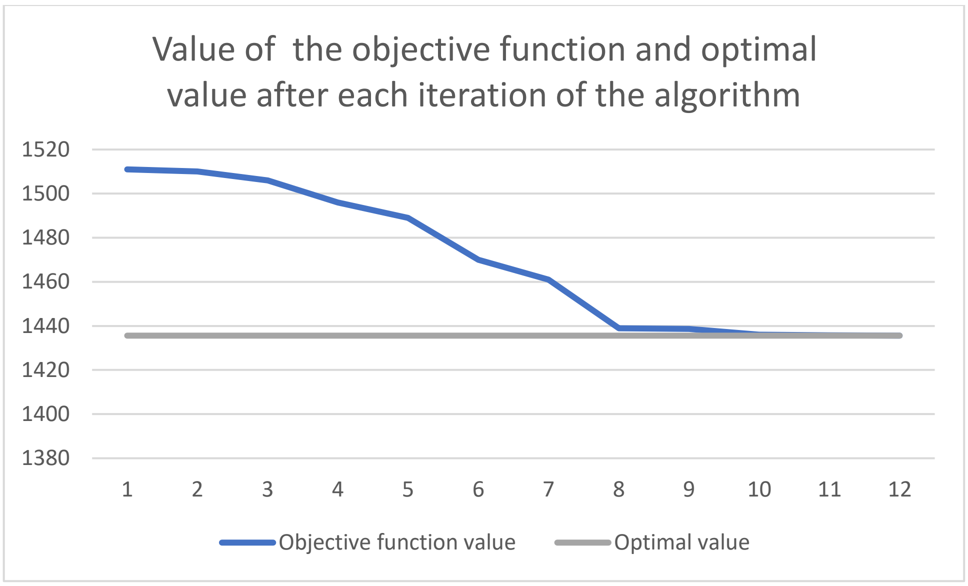
For a certain pattern i:

```
for (var j in patt) {
      ReducedCost[i] -= PatternGen[i][j] * prodDem[j].dual;
```

Where prodDem[j] is the dual variable value of the constraint associated with the j-th roll size.

Once we computed the reduced costs of the deactivated patterns, let's find the minimum value, which is the most interesting pattern since we're working on a minimization problem. We can then add this i-th pattern to the model by setting PatternAct[i] to 1, simulating a column generation algorithm.

By iterating 12 times, we obtain the optimal solution (over the fixed set of generated patterns).



Note that we didn't need to add a list of length 21 composed of 1's only (the identity matrix) to the PatternGen table because the set of generated patterns already contained a solution to the problem.

3) We could obtain an integer solution simply by changing the type of the decision variable to int+ in CPLEX. Implementing this method and using 500 generated patterns, the solver found the optimal solution with a value of 1436. However, when using 200 patterns, the solver only finds a solution with an objective function value of 1514. We could implement a more reliable solution, like the branch and bound algorithm. We could implement it by hand by finding the most fractionable variable, building and solving two models which are the corresponding linear relaxations for every iteration.

4) The total loss can be minimized by adding a penalty term to the objective function that penalizes the number of rolls of fabric that are cut. This penalty term can be set to a large value so that the solver will try to minimize it as much as possible.

In this case, I'll add the number or rolls wasted multiplied by 10. We calculate the amount of fabric wasted for each pattern used and multiply it by the number of times that pattern was used to get this metric. We obtain a solution of 1526 rolls used, and 114 rolls wasted. If we change the weight of the penalty term to 100, the solution is 2194 rolls used with 91 rolls wasted. The integer solution with an objective function value of 1436 we found in question 3 produces 116 rolls of waste by comparison.