

Advanced Models and Methods in Operations
Research
Column Generation Heuristics

Florian Fontan

November 23, 2021

Table of contents

Exponential MILP formulations

Solving the relaxation by Column Generation

Finding solutions after the Column Generation

`columngenerationsolver.py`

Conclusion

Cutting Stock Problem, Description

Input:

- ▶ a capacity C
- ▶ n item types; for each item type $j = 1, \dots, n$, a weight w_j and a demand q_j

Problem:

- ▶ Pack all items such that the total weight of the items in a bin does not exceed the capacity.

Objective:

- ▶ Minimize the number of bin used.

Cutting Stock Problem, Formulation

Let us define the K possible patterns such that $x_j^k = q$ iff pattern k , $k = 1 \dots K$ contains q copies of item type j

► Variables:

► $y^k \in \mathbb{N}$, $\forall k = 1 \dots K$.

$y^k = q$ iff q copies of pattern k are used

► Objective:

$$\min \sum_{k=1}^K y^k$$

► Constraints:

$$\sum_{k=1}^K x_j^k y^k = q_j \quad \forall j = 1 \dots n$$

Cutting Stock Problem, Formulation

Let us define the K possible patterns such that $x_j^k = q$ iff pattern k , $k = 1 \dots K$ contains q copies of item type j

► Variables:

► $y^k \in \mathbb{N}, \forall k = 1 \dots K.$

$y^k = q$ iff q copies of pattern k are used

► Objective:

$$\min \sum_{k=1}^K y^k$$

► Constraints:

$$\sum_{k=1}^K x_j^k y^k = q_j \quad \forall j = 1 \dots n$$

Why is this formulation good compared to the classical one?

Cutting Stock Problem, Formulation

Let us define the K possible patterns such that $x_j^k = q$ iff pattern k , $k = 1 \dots K$ contains q copies of item type j

- ▶ Variables:

- ▶ $y^k \in \mathbb{N}$, $\forall k = 1 \dots K$.

- $y^k = q$ iff q copies of pattern k are used

- ▶ Objective:

$$\min \sum_{k=1}^K y^k$$

- ▶ Constraints:

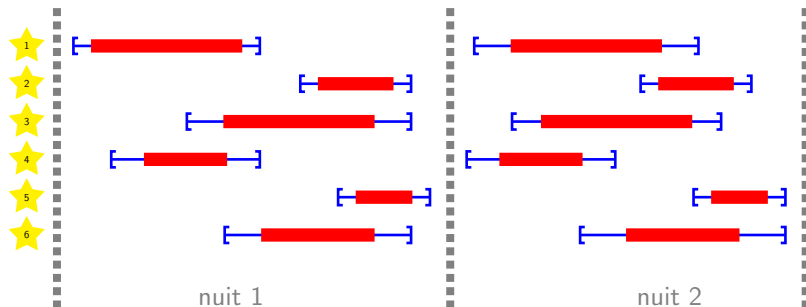
$$\sum_{k=1}^K x_j^k y^k = q_j \quad \forall j = 1 \dots n$$

Why is this formulation good compared to the classical one?

- ▶ No big-M constraint
- ▶ Better relaxation
- ▶ Easier to write

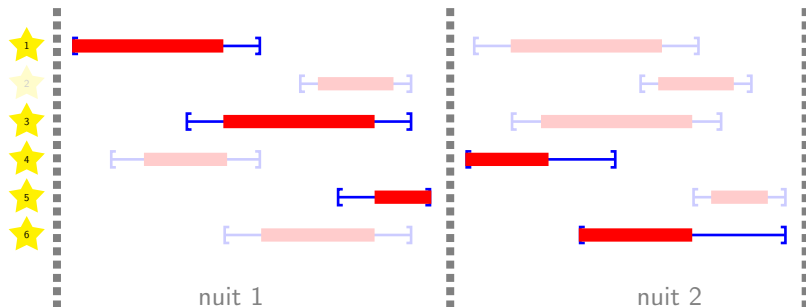
Star Observation Scheduling Problem, Description

Input: a set \mathcal{M} of nights and a set \mathcal{N} of stars; for each star $j \in \mathcal{N}$, a scientific interest w_j , an observation duration p_j^i and a visibility window $[r_j^i, d_j^i]$, depending on the night i of the observation.



Star Observation Scheduling Problem, Description

Input: a set \mathcal{M} of nights and a set \mathcal{N} of stars; for each star $j \in \mathcal{N}$, a scientific interest w_j , an observation duration p_j^i and a visibility window $[r_j^i, d_j^i]$, depending on the night i of the observation.



Star Observation Scheduling Problem, Formulation

For each night i , $i = 1 \dots m$, let us define the K_i possible schedules such that $x_{i,j}^k = 1$ iff schedule k , $k = 1 \dots K_i$ of night i contains star j

► Variables:

- $y_i^k \in \{0, 1\}$, $\forall i = 1 \dots m$, $\forall k = 1 \dots K_i$.
 $y_i^k = 1$ iff scheduled k of night i is selected

► Objective:

$$\max \sum_{i=1}^m \sum_{k=1}^{K_i} \sum_{j=1}^n w_j x_{i,j}^k y_i^k$$

► Constraints:

$$\sum_{k=1}^{K_i} y_i^k = 1 \quad \forall i = 1 \dots m$$

$$\sum_{i=1}^n \sum_{k=1}^{K_i} x_{i,j}^k y_i^k \leq 1 \quad \forall j = 1 \dots n$$

2D Guillotine Variable-sized Bin Packing, Description

Input:

- ▶ n item types; for each item type $j = 1, \dots, n$, a width w_j , a height h_j and a demand q_j
- ▶ m bin types; for each bin type $i = 1, \dots, m$, a width W_i , a height H_i , a lower bound l_i , an upper bound u_i and a cost c_i

Problem:

- ▶ Find a subset of guillotine patterns such that all item type demands and bin type use bounds are satisfied

Objective:

- ▶ Minimize the cost of the selected bins.

2D Guillotine Variable-sized Bin Packing, Formulation

For each bin type i , $i = 1 \dots m$, let us define the K_i possible patterns such that $x_{i,j}^k = q$ iff pattern k , $k = 1 \dots K_i$ of bin type i contains q copies of item type j

► Variables:

► $y_i^k \in \mathbb{N}$, $\forall i = 1 \dots m$, $\forall k = 1 \dots K_i$.

$y_i^k = q$ iff q copies of pattern k of bin type i are used

► Objective:

$$\min \sum_{i=1}^m \sum_{k=1}^{K_i} c_i y_i^k$$

► Constraints:

$$l_i \leq \sum_{k=1}^{K_i} y_i^k \leq u_i \quad \forall i = 1 \dots m$$

$$\sum_{i=1}^n \sum_{k=1}^{K_i} x_{i,j}^k y_i^k = q_j \quad \forall j = 1 \dots n$$

Other examples

Usually, variables represent:

- ▶ A bin/knapsack (for packing problems)
- ▶ The schedule of a machine (for parallel scheduling problems)
- ▶ The route of a vehicle (for vehicle routing problems)
- ▶ ...

Table of contents

Exponential MILP formulations

Solving the relaxation by Column Generation

Finding solutions after the Column Generation

`columngenerationsolver.py`

Conclusion

Introduction

- ▶ With these formulations, generating all the variables is generally not possible since their number grows exponentially with the size of the problem.
- ▶ First we focus on solving the **linear relaxation**

The Column Generation procedure

- ▶ We use the **simplex algorithm**.
 - ▶ At each iteration, it adds a variable of negative reduced cost to the current basis

- ▶ Objective:

$$\min \sum_{j=1}^n c_j x_j$$

- ▶ Constraints:

$$\sum_{j=1}^n a_{i,j} x_j \leq b_i \quad \forall i = 1 \dots m$$

- ▶ Reduced cost of variable x_j :

$$c_j - \sum_{i=1}^m a_{i,j} v_i$$

with v_i the dual value of constraint i .

- ▶ It stops when there are no variable of negative reduced cost
- ▶ The difference with the traditional simplex algorithm, is that here, it is not possible to loop through all the variables to find a variable of negative reduced cost, since they have not been all generated.

The Column Generation procedure

- ▶ Instead, finding a variable of negative reduced cost becomes an optimization problem
- ▶ Example with the Cutting Stock Problem
 - ▶ Objective:

$$\min \sum_{k=1}^K y^k$$

- ▶ Constraints:

$$\sum_{k=1}^K x_j^k y^k = q_j \quad \forall j = 1 \dots n$$

- ▶ Reduced cost of y^k :

$$1 - \sum_{j=1}^n x_j^k v_j$$

with v_j the dual value of constraint j .

The Column Generation procedure

- ▶ We look for a variable y^k such that

$$1 - \sum_{j=1}^n x_j^k v_j < 0$$

- ▶ Finding a variable of negative reduced cost is equivalent to finding a pattern with total profit ≥ 1 with the profit of item type j being equal to v_j .
- ▶ In practice, we solve the problem as an optimization problem: we find the best solution of the Knapsack Problem and check if the reduced cost of the corresponding variable is negative.

The Column Generation procedure

Summary:

function ColumnGeneration(P)

$Y \leftarrow$ initial set of columns

while True **do**

 Solve the Linear Program P' with variables from Y

 Look for a variable of negative reduced cost (**Pricing Problem**)

if there is one **then**

 Add it to Y

else

return the solution of P'

Initial set of columns

- ▶ To get dual values, the LP needs to be feasible
- ▶ With 0 variable, the LP might be infeasible
 - ▶ Example: Cutting Stock Problem, demand constraints are not satisfied
- ▶ Therefore, we need to find a way to get an initial set of columns such that the LP is feasible
 - ▶ Find a feasible solution and add the corresponding columns
 - ▶ Example: Cutting Stock, Best Fit algorithm
 - ▶ Drawback: Problem specific, additional work for the implementation of the heuristic
 - ▶ Advantage: if the solution is good, it might speed up the column generation procedure
 - ▶ Find manually a set of columns that ensures the LP to be feasible
 - ▶ Example: create n columns with only one item
 - ▶ Generate a dummy column with very high cost for each problematic constraint
 - ▶ Advantage: not problem specific
 - ▶ Drawback: numerical issue is the cost of the dummy columns is not well calibrated

Star Observation Scheduling Problem, Pricing

- Objective:

$$\max \sum_{i=1}^m \sum_{k=1}^{K_i} \sum_{j=1}^n w_j x_{i,j}^k y_i^k$$

- Constraints:

$$\sum_{k=1}^{K_i} y_i^k = 1 \quad \forall i = 1 \dots m \quad \text{dual } u_i$$

$$\sum_{i=1}^n \sum_{k=1}^{K_i} x_{i,j}^k y_i^k \leq 1 \quad \forall j = 1 \dots n \quad \text{dual } v_j$$

Star Observation Scheduling Problem, Pricing

- Objective:

$$\max \sum_{i=1}^m \sum_{k=1}^{K_i} \sum_{j=1}^n w_j x_{i,j}^k y_i^k$$

- Constraints:

$$\sum_{k=1}^{K_i} y_i^k = 1 \quad \forall i = 1 \dots m \quad \text{dual } u_i$$

$$\sum_{i=1}^m \sum_{k=1}^{K_i} x_{i,j}^k y_i^k \leq 1 \quad \forall j = 1 \dots n \quad \text{dual } v_j$$

- Reduced cost of y_i^k :

$$\sum_{j=1}^n w_j x_{i,j}^k - u_i - \sum_{j=1}^n x_{i,j}^k v_j = \sum_{j=1}^n (w_j - v_j) x_{i,j}^k - u_i$$

Star Observation Scheduling Problem, Pricing

- Objective:

$$\max \sum_{i=1}^m \sum_{k=1}^{K_i} \sum_{j=1}^n w_j x_{i,j}^k y_i^k$$

- Constraints:

$$\sum_{k=1}^{K_i} y_i^k = 1 \quad \forall i = 1 \dots m \quad \text{dual } u_i$$

$$\sum_{i=1}^m \sum_{k=1}^{K_i} x_{i,j}^k y_i^k \leq 1 \quad \forall j = 1 \dots n \quad \text{dual } v_j$$

- Reduced cost of y_i^k :

$$\sum_{j=1}^n w_j x_{i,j}^k - u_i - \sum_{j=1}^n x_{i,j}^k v_j = \sum_{j=1}^n (w_j - v_j) x_{i,j}^k - u_i$$

- Finding a variable of maximum reduced cost reduces to solving m Single Night Star Observation Scheduling Problems with targets with profit $w_j - v_j$.

2D Guillotine Variable-sized Bin Packing, Pricing

- Objective:

$$\min \sum_{i=1}^m \sum_{k=1}^{K_i} c_i y_i^k$$

- Constraints:

$$l_i \leq \sum_{k=1}^{K_i} y_i^k \leq u_i \quad \forall i = 1 \dots m \quad \text{dual } u'_i$$

$$\sum_{i=1}^n \sum_{k=1}^{K_i} x_{i,j}^k y_i^k = q_j \quad \forall j = 1 \dots n \quad \text{dual } v_j$$

2D Guillotine Variable-sized Bin Packing, Pricing

- Objective:

$$\min \sum_{i=1}^m \sum_{k=1}^{K_i} c_i y_i^k$$

- Constraints:

$$l_i \leq \sum_{k=1}^{K_i} y_i^k \leq u_i \quad \forall i = 1 \dots m \quad \text{dual } u'_i$$

$$\sum_{i=1}^n \sum_{k=1}^{K_i} x_{i,j}^k y_i^k = q_j \quad \forall j = 1 \dots n \quad \text{dual } v_j$$

- Reduced cost of y_i^k :

$$c_i - u'_i - \sum_{j=1}^n x_j^k v_j$$

2D Guillotine Variable-sized Bin Packing, Pricing

- Objective:

$$\min \sum_{i=1}^m \sum_{k=1}^{K_i} c_i y_i^k$$

- Constraints:

$$l_i \leq \sum_{k=1}^{K_i} y_i^k \leq u_i \quad \forall i = 1 \dots m \quad \text{dual } u'_i$$

$$\sum_{i=1}^n \sum_{k=1}^{K_i} x_{i,j}^k y_i^k = q_j \quad \forall j = 1 \dots n \quad \text{dual } v_j$$

- Reduced cost of y_i^k :

$$c_i - u'_i - \sum_{j=1}^n x_{i,j}^k v_j$$

- Finding a variable of minimum reduced cost reduces to solving m 2D Guillotine Knapsack Problems with items with profit v_j for each bin type.

Transition

- ▶ The Column Generation procedure solves the relaxation of the exponential formulation

Transition

- ▶ The Column Generation procedure solves the relaxation of the exponential formulation
- ▶ Thus, it provides a valid bound

Transition

- ▶ The Column Generation procedure solves the relaxation of the exponential formulation
- ▶ Thus, it provides a valid bound
- ▶ But it generally does not provide a feasible solution

Transition

- ▶ The Column Generation procedure solves the relaxation of the exponential formulation
- ▶ Thus, it provides a valid bound
- ▶ But it generally does not provide a feasible solution
- ▶ How to exploit the Column Generation to get feasible solutions?

Table of contents

Exponential MILP formulations

Solving the relaxation by Column Generation

Finding solutions after the Column Generation

`columngenerationsolver.py`

Conclusion

The Branch-and-Price algorithm

- ▶ LP-based branch-and-bound, the relaxation is solved by the Column Generation procedure in each node
- ▶ How to branch?
 - ▶ Branching on columns of the exponential formulation? No, the pricing problem becomes too difficult
 - ▶ Branching on the variables of the primal formulation?
 - ▶ Bin Packing: branch on whether item j is packed in bin i or not. If yes, then the available bins have now different capacities and a Knapsack Problem for each capacity needs to be computed
 - ▶ Best solution for the Bin Packing: branch on whether two items are packed in the same bin or not. If yes, then they are merged into a single item. If no, then the subproblem becomes a Knapsack Problem with Conflicts which is strongly NP-hard
- ▶ \implies Complex to implement.
- ▶ It can be combined with cuts (Branch-and-Price-and-Cut). The added cuts might also change the pricing problem \implies even more complex to implement
- ▶ Only exact method based on Column Generation, state-of-the-art exact method for many Vehicle Routing and Parallel Machine Scheduling Problems

Solving the restricted master

- ▶ The Column Generation procedure is executed once
- ▶ Solve the exponential formulation with a MILP solver using only the columns generated during the Column Generation procedure
- ▶ No guarantee to find the optimal solution (or even a feasible solution)
- ▶ Solving the MILP is computationally expensive if many columns have been generated. It can take some time before finding a first solution
- ▶ It requires a good MILP solver

Heuristic Tree Search

Branching scheme:

- ▶ Root node: no column has been fixed
- ▶ Children: solve the relaxation by column generation, select the variable y with the most integral value $v \neq 0$, for each possible value v' of y create one child.
- ▶ The discrepancy of a child is computed as:

$$\text{disc}_{\text{child}} = \text{disc}_{\text{father}} + |v' - v|$$

Algorithms:

- ▶ Greedy
- ▶ Limited Discrepancy Search

Note that the depth of the tree is of the order of the number of columns in a solution.

Additional tricks

- ▶ Using a fast heuristic algorithm to solve the pricing problem. If the heuristic doesn't find a column of negative reduced cost:
 - ▶ Case 1: Try with a more expensive exact algorithm
 - ▶ Case 2: Stop the column generation procedure. The bound is not valid, therefore, it is not possible to use an exact Branch-and-price in this case. But the heuristics still work.
- ▶ Generating columns without the simplex algorithm
 - ▶ It might be faster than the column generation procedure
 - ▶ It might be difficult to generate columns that fit well together
 - ▶ No bound
 - ▶ Then solve the restricted master or use a Heuristic Tree Search algorithm
- ▶ Solving the restricted master with a heuristic algorithm
 - ▶ Often, the master problem is a set covering or set packing problem for which heuristic algorithms have already been developed
 - ▶ It might be faster than a MILP solver

Table of contents

Exponential MILP formulations

Solving the relaxation by Column Generation

Finding solutions after the Column Generation

`columngeneration`
`solver.py`

Conclusion

columngenerationsolverpy

- ▶ A package that simplifies the implementation of Column Generation based algorithms
- ▶ Written in Python3 (original version in C++)
- ▶ <https://github.com/fontanf/columngenerationsolverpy>
- ▶ Install with: `pip3 install columngenerationsolverpy`
- ▶ It includes:
 - ▶ The Column Generation algorithm
 - ▶ The Greedy algorithm
 - ▶ The Limited Discrepancy Search algorithm
- ▶ To solve a problem, one needs to provide the exponential formulation and the solver for the Pricing Problem (able to take as input the currently fixed columns)
- ▶ The implementation of the Greedy algorithm and the Limited Discrepancy Search algorithm relies on the `treesearchsolverpy` package

Table of contents

Exponential MILP formulations

Solving the relaxation by Column Generation

Finding solutions after the Column Generation

`columngeneration solver.py`

Conclusion

Conclusion

- ▶ Column Generation: solving the relaxation of exponential formulations by generating the columns dynamically
- ▶ It can be embedded in a classical Branch-and-bound
 - ▶ State-of-the art exact method for many Vehicle Routing and Parallel Machine Scheduling Problems
 - ▶ Cumbersome to implement
- ▶ It can be embedded in a Heuristic Tree Search framework
 - ▶ Also state-of-the-art heuristics for several problems
 - ▶ Easier to implement
- ▶ Works better when the number of elements in columns is small (≤ 20)

Advanced Models and Methods in Operations Research

Column Generation Heuristics

Florian Fontan

November 23, 2021