DM872 – Mathematical Optimization at Work

Answers to Obligatory Assignment 1, Spring 2022

Emil Ovcina

emovc18@student.sdu.dk

1 Task 1

This is the VRPTW problem formulated as a set partitioning problem.

`Notation:`
N is the set of customers.
K is the set of vehicles.
Ω is the set of feasible VRPTW paths.
c_p is the cost of path $p \in \Omega$
$a_{ip} \in \{1, 0\}$ constant 1 if $i \in N$ is visited on path $p \in \Omega$

`Variables:`
$\theta_p \in \{1, 0\}$ 1 if path $p \in \Omega$ is chosen by a vehicle.

`Model:`

$$\begin{aligned}
\min \quad & \sum_{p \in \Omega} c_p \theta_p && (1)\\
s.t. \quad & \sum_{p \in \Omega} a_{ip} \theta_p = 1 \quad \forall i \in N && (2)\\
& \sum_{p \in \Omega} \theta_p \leq |K| && (3)\\
& \theta_p \in \mathbb{B} \quad \forall p \in \Omega && (4)
\end{aligned}$$

The goal is to minimize the total cost of all chosen paths (1). (2) ensures that each customer is only visited once and (3) ensures that the number of paths chosen is at most the number of vehicles available. (4) is the binary constraint of the decision variables.

Computatonionally the set partitioning problem is a well-known NP-hard problem, meaning that it is not solveable in polynomial time, and that is clear to see here as well. The number of paths in Ω grows exponentially with the number of customers, so finding the optimal solution can only be at least as hard as the set partitioning problem. The integrality constraint also suggest that a branching algorithm might be necessary, which only adds to the computational complexity.

2 Task 2

A simple heuristic to use is a variation of 'nearest neighbour heuristic' [1], which basically calculates the closest customer of a partial path and appends that to the path. To calculate the closest customer, one needs to look at the travel time, distance, time window and capacity.

So at each iteration of the algorithm finds the closest neighbour in the sense of traveling time, urgency and least waiting time, and appends that to the end of the current partial path. If no customers are left, simply stop, but if there still are customers left but no feasible way to append them into the path, a new path is created. When looking at feasibility the algorithm also has to check if it is possible to go back to the depot in time, if the customer is picked. Doing this allows us to specify that a maximum of 3 paths can be created, because of the number of vehicles in our problem instance.

So first priority is to find the closest in the sense of traveling time, for tie breaker the urgency is looked at, meaning that the node with the closest l_i has priority. For a third tie-breaker the algorithm will prioritize the node with the least waiting time. If ties still exists, pick a random.

Initial feasible solution:

$$\begin{tikzcd} \boxed{0} \arrow[r] & \textcircled{8} \arrow[r] & \textcircled{5} \arrow[r] & \textcircled{6} \arrow[r] & \boxed{0} \end{tikzcd}$$

$c_1 = 12$

$$\begin{tikzcd} \boxed{0} \arrow[r] & \textcircled{4} \arrow[r] & \textcircled{3} \arrow[r] & \textcircled{2} \arrow[r] & \boxed{0} \end{tikzcd}$$

$c_2 = 13$

$$\begin{tikzcd} \boxed{0} \arrow[r] & \textcircled{1} \arrow[r] & \textcircled{7} \arrow[r] & \boxed{0} \end{tikzcd}$$

$c_3 = 21$

[1] Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints - Marius M. Solomon

3 Task 3

The Master problem (MP) is the linear relaxation of the SP problem described in task 1. I will relax the integrality contraints (4) so the model will look like this:

$$\min \quad \sum_{p\in\Omega} c_p\theta_p \tag{5}$$
$$s.t. \quad \sum_{p\in\Omega} a_{ip}\theta_p = 1 \quad \forall i \in N \tag{6}$$
$$\sum_{p\in\Omega} \theta_p \leq |K| \tag{7}$$
$$\theta_p \geq 0 \quad \forall p \in \Omega \tag{8}$$

For the restricted master problem (RMP) I restrict the number of paths, since the problem with Ω described in task 1 is that there is too many paths to enumerate in one problem, so a new set of paths $\hat{\Omega} \subset \Omega$ is used.
RMP:

$$\min \quad \sum_{p\in\hat{\Omega}} c_p\theta_p \tag{9}$$
$$s.t. \quad \sum_{p\in\hat{\Omega}} a_{ip}\theta_p = 1 \quad \forall i \in N \tag{10}$$
$$\sum_{p\in\hat{\Omega}} \theta_p \leq |K| \tag{11}$$
$$\theta_p \geq 0 \quad \forall p \in \hat{\Omega} \tag{12}$$

The RMP allows the column generation procedure to take place since restricting the number of varaibles corresponds to restricting the number of columns in the tableau.

The Subproblem (pricing problem) is used in the column generation process to find new routes that gives the RMP i better objective function value:

`Notation:`
V is the set of vertices
$N = V \setminus \{0\}$ set of customers
A is the set of arcs between vertices.
K is the set of vehicles.
Q is the maximum capacity of a vehicle.
q_i demand of customer i.
e_i earliest service time for customer i.
l_i latest service time for customer i.
t_{ij} is the travel time of arc $(i, j) \in A$
c_{ij} is the cost of using arc $(i, j) \in A$
λ_i is the dual variables from RMP constraint (10).

`Variables:`
$x_{ij} \in \{1, 0\}$ 1 if arc $(i, j) \in A$ is used, 0 otherwise.
$y_i \in \mathbb{R}_+$ load of vehicle after serving i.
$z_i \in \mathbb{R}_+$ time to serve customer i.

`Model:`

$$\min \quad \sum_{(i,j)\in A} (c_{ij} - \lambda_i)x_{ij} \tag{13}$$

$$s.t. \quad \sum_{(i,k)\in A} x_{ik} = \sum_{(k,j)\in A} x_{kj} \quad \forall k \in N \tag{14}$$

$$\sum_{(0,j)\in A} x_{0,j} = 1 \tag{15}$$

$$\sum_{(i,0)\in A} x_{i,0} = 1 \tag{16}$$

$$y_i + q_j + M'x_{ij} \leq y_j + M' \quad \forall (i,j) \in A, j \neq 0 \tag{17}$$

$$0 \leq y_i \leq Q \quad \forall i \in V \tag{18}$$

$$z_i + t_{ij} - M(1 - x_{ij}) \leq z_j \quad \forall (i,j) \in A \tag{19}$$

$$e_i \leq z_i \leq l_i \quad \forall i \in V \tag{20}$$

The objective is to minimize the reduced cost of the RMP. Constraints (14), (15) and (16) are the flow constraints, while (17) and (18) are the capacity constraints, (17) being big M' constraints which should be as tight as possible. Setting the M' to be Q will give a tighter constraint while they still serves their purpose of Big-M constrains. Constraints (19) and (20) are the time window constraints and (19) are also big-M constraints. To find the M one can do that easily by looking at the time windows. The latest time window can simply be used here, which is the l_0, meaning the latest time of the depot.

4 Task 4

First the restricted master problem was solved to find the dual variables used to calculate the reduced cost: $\hat{c}_r = c_r \left(\sum_{i \in N} a_{ir} \lambda_i \right) - \lambda_0$

Objective: 41.6666667
The first set of lambdas:
$\lambda_0 = 0.0$
$\lambda_i = [12.0, 5.3334, 3.0, 4.6666, 4.6666, 5.6667, 4.6666, 1.6665]$

The first found column with negative cost:
$[0, 0, 0, 0, 1, 1, 1, 0]$ which has a reduced cost of -3.992

The restricted master problem is solved again to find the new dual variables:
$\lambda_0 = -3.996$
$\lambda_i = [16.0, 5.334, 3.0, 8.666, 4.666, 5.667, 4.666, 5.666]$
Objective function value: 40.5

The second column found with a negative reduced cost:
$[1, 1, 1, 0, 0, 0, 0, 0]$ which has a reduced cost of -2.333

The third column:
$[1, 1, 0, 0, 0, 0, 0, 0]$ with reduced cost: -3.0, objective value = 40.5

The fourth column:
$[0, 0, 0, 1, 0, 0, 0, 0]$ with: -1, objective value = 40.0

The variables:
$\theta = [0.5, 0.5, 0, 0, 0, 0, 0, 0, 0.5, 0, 0, 1, 0, 0, 0.5]$

All source code used to solve the restricted master problem and find the dual variables can be found in `/src/task4.py`

5 Task 5

All code used for this task can be found in `/src/task5.py`

When running the code, the first function that is called is `solve_master_problem_with_cg`, which will take the initial routes and the corresponding costs from /src/data.py and create the restricted master problem using *gurobi*

When the model has been set up, a while loop will run and perform the column generation. The loop will first find the optimal solution to the RMP and then call the `solve_pricing_problem` function using the dual variables from the RMP.

The pricing problem model is slightly different to the formulation in task 3. To more easily model the time window constraints, an "end depot" is created as the 9'th point, which will be the depot the route ends in. Now, the pricing problem is created, it will find the lowest reduced cost by selecting some arcs in the network while still complying the constraints. The reduced cost, the route and the cost of the route will be returned to the column generation procedure.

Back in the column generation procedure: now it should check if the reduced cost from the pricing problem is non negative and if the route already exists in the list of routes. If true, the column generation will stop. If not, it will add the route and the cost of the route to their respective lists and the variable will be added to the gurobi model.

To add a variable in the gurobi model, a column object has to be made, and then for each constraint in the model, the value of the column entry must be set by looping through the route indices. Lastly the last entry of the column is for the "amount of vehicles" constraint in the MP, which always will be set to 1.

When the new route(variable) has been added to the model, the model will optimize again, starting from when the last solution of the model so it does not have to resolve the whole model from the beginning.

The procedure will run until the break conditions are met, and the function will return all the routes and the final objective function value.

Output from running the code using the routes from `Task 2`

```
 Added route:  [1 1 0 0 0 0 0 0]   - with cost:  15.0  - red.cost:  -19.0
 Added route:  [1 0 1 0 0 0 0 0]   - with cost:  15.0  - red.cost:  -25.0
 Added route:  [0 0 1 0 1 0 1 0]   - with cost:  15.0  - red.cost:  -16.0
 Added route:  [0 0 1 0 0 1 1 0]   - with cost:  16.0  - red.cost:  -19.0
 Added route:  [0 0 0 1 0 1 1 0]   - with cost:  15.0  - red.cost:  -8.75
 Added route:  [0 0 0 0 0 0 1 1]   - with cost:  11.0  - red.cost:  -11.33
 Added route:  [0 0 0 1 1 0 1 0]   - with cost:  15.0  - red.cost:  -16.0
 Added route:  [1 1 0 0 0 0 1 0]   - with cost:  22.0  - red.cost:  -14.0
 Added route:  [0 0 0 0 1 0 1 1]   - with cost:  11.0  - red.cost:  -4.0
 Added route:  [0 0 0 0 1 1 1 0]   - with cost:  11.0  - red.cost:  -5.5
 Added route:  [0 0 0 1 0 0 0 0]   - with cost:  6.0   - red.cost:  -2.0
 Added route:  [1 1 1 0 0 0 0 0]   - with cost:  18.0  - red.cost:  -3.5
No new columns could be added to the problem

OBJECTIVE FUNCTION VALUE:  40.0
```

θ assignment:
$\theta_0 = 0.5$
$\theta_7 = 0.5$
$\theta_{11} = 0.5$
$\theta_{13} = 0.5$
$\theta_{14} = 1.0$

Others are 0.0

6 Task 6

All code for solving this task can be found in /src/task6.py

The code from /src/task5.py has been extended to now include a procedure for finding SR-cuts and adding them as a constraint.

The while loop which does the column generation has been wrappen in another while loop which will handle the SR-cuts. So now, when all the column have been generated and a lower bound has been found. The CG loop will break out into the SR loop, where the find_sr_cut function will find the most violating cut, like described in the slides from DTU.

The find_sr_cut function goes through all triplets of customers and looks at all columns to see how many routes cover at least two of the customers. If a column violates $a_{ir} + a_{jr} + a_{kr} \geq 2$. For all columns(routes) that violates this this constraint, the θ value of the RMP solution is looked at and the sum of all θ values for all violating routes are summed up, and if that sum is greater than 1, it is violating, and an SR-cut should be added. However, the procedure finds the "most violating" set of customers, and returns the list of columns(routes) that is violating.

Back in the SR-cut loop: a break condition is checked. If the most violating cut is already found or the solution is integral, the procedure stops, else the constraint is added to the RMP, and the solving continues with column generation. The customer triplet an their corresponding dual variable after resolving the RMP, is added to a list of tuples called mu, which is given to the modified pricing problem.

To keep the amount of constraints and variables as low as possible in the pricing problem (which is displayed on the DTU slides), only the customers participating in an SR-cut is added as variables.

Output from running the code using routes found in Task 4:

```
No more columns to add
Added SR-cut:  [0, 1, 9, 11, 12]  for  [5, 6, 7]
Added column:  [0 0 1 1 1 0 0 0]  - cost:  17.0  - red.cost:  -4.0
Added column:  [0 0 0 0 1 0 0 1]  - cost:  9.0  - red.cost:  -4.0
No more columns to add
Added SR-cut:  [1, 8, 9, 11, 12, 17]  for  [4, 6, 7]
No more columns to add
No more cuts to add!

SELECTED ROUTES:  ['theta[0]', 'theta[13]', 'theta[17]']
OBJECTIVE FUNCTION VALUE:  42.0
```

$\theta_0 = [1, 1, 1, 0, 0, 0, 0, 0]$
$\theta_{13} = [0, 0, 0, 1, 0, 1, 1, 0]$
$\theta_{17} = [0, 0, 0, 0, 1, 0, 0, 1]$

7 Task 7

Code used to help solve this task can be found in `/src/task7.py`

I branch on the routes, since I do this by hand, it seems easier than to branch on the arcs. I follow the procedure described in the DTU slides.

Initial tableau optained from task 4:

15	12	22	18	15	22	18	10	15	11	13	12	11	18	15	6
0	0	1	0	1	1	0	0	0	0	0	0	0	1	1	0
0	0	0	1	0	1	1	1	0	0	1	0	0	1	1	0
0	0	0	0	1	0	1	0	1	0	1	0	0	1	0	0
1	0	0	1	0	0	0	1	0	0	1	0	0	0	0	1
0	1	1	1	0	0	0	0	1	1	0	0	1	0	0	0
1	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0
1	0	0	0	0	1	0	0	1	1	0	1	1	0	0	0
0	1	0	0	0	0	1	0	0	1	0	1	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0.5	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.0	0.0	0.0	1.0	0.0	0.5

First branch: $\theta_0 = 0$
The route θ_0 is removed from the tableau and the pricing problem is modified to not include the route, simply by adding the constraint:
$\sum_{(i,j)\in\theta_0} x_{i,j} \leq T - 1$ where T is the number of arcs traversed in θ_0.

Running the pricing problem generates this column: $[0, 0, 0, 1, 0, 1, 0, 0]$ The optimal solution for the RMP at this node is: 43.0
and the solution is integral:
$\theta_9 = [0, 0, 0, 0, 1, 0, 1, 1]$
$\theta_{13} = [1, 1, 1, 0, 0, 0, 0, 0]$
$\theta_{16} = [0, 0, 0, 1, 0, 1, 0, 0]$

Second branch: $\theta_0 = 1$, which means that route 0 must be chosen,
to continue the customers $4, 6, 7$ are removed and gives this tableau:

22	15	18	18	15	11	22		b
1	1	0	1	1	0	1	=	1
0	0	1	1	1	0	1	=	1
0	1	1	1	0	0	0	=	1
1	0	0	0	0	1	1	=	1
0	0	1	0	0	1	0	=	1
1	1	1	1	1	1	1	$\leq$	2
0.0	0.0	0.0	1.0	0.0	1.0	0.0		

With this matrix where customers 4,6 and 7 are removed, and the CG process is performed again on this smaller problem and generate the two columns: $[0, 0, 0, 0, 1, 0, 0, 1]$ $[0, 0, 0, 0, 1, 0, 0, 0]]$ The solution is also integral and gives an objective function value of 42.0
The chosen routes besides θ_0:
$\theta_{13} = [1, 1, 1, 0, 0, 0, 0, 0]$
$\theta_{16} = [0, 0, 0, 0, 1, 0, 0, 1]$

And because both branches are integral, the procedure stops and the lowest value 42.0 is the optimal solution.
$\theta_0 = [0, 0, 0, 1, 0, 1, 1, 0]$
$\theta_{13} = [1, 1, 1, 0, 0, 0, 0, 0]$
$\theta_{16} = [0, 0, 0, 0, 1, 0, 0, 1]$

In `/src/task7.py` some functions are created to alter the problem to aid in the branching procedure. `remove_customers` is going through all the data the problem needs and removes the list of customers specified as an input parameter. This function is used when a routes is fixed to be equal 1.

The function `remove_route` is used when a route is fixed to be equal 0, and simply removes it from the list of routes and costRoutes.