COLUMN GENERATION IMPLEMENTATION FOR PICKUP AND DELIVERY PROBLEM WITH TIME AND CAPACITY CONSTRAINTS
TEAM COATI

Internship Report

Athanasia Farmaki

Supervisor: Dr. David Coudert
Co-supervisor: Dr Nicolah Nice

Team: COATI

Sophia-Antipolis, 2019

Table of Contents

1 Introduction

In this report we analyze the implementation of "Branch and Cut and Price for the Pickup and Delivery Problem with Time Windows" by Ropke and Cordeau(2009) [5]. The paper refers to the pickup and delivery problem using capacity and time constraints. The goal is to find the best set of paths to serve all the requests in the optimal way given that the vehicles used are identical and have a certain capacity and that the requests must be served within a time window. The column generation algorithm used finds the optimal solution by solving a relaxed version of the master problem using simplex while generating paths found by a constrained shortest path pricing problem. In the next sections of this report we will briefly present column generation, we will introduce the data used to test the algorithm along with the preprocessing steps and the decisions taken on creating the time windows and capacity constraints. Then we will highlight some points in the implementation of the master and the pricing problem and last but not least we will refer the branching strategy used in order to retrieve an integer solution. I would like to highlight that the purpose of this report is not to analyze the theoretical aspects of the algorithm as this can be found in the corresponding paper and its references. The main purpose is to highlight parts of the implementation in order to help the reader to effectively study the code.

2 Theoretical Background

2.1 Column Generation

In many problems the space of the possible solutions is too great to consider. For example in Pickup and Delivery problem, in which we need to minimize the cost of the paths traversed by multiple vehicles in order to pickup and deliver all the clients, the number of paths is huge even with constraints limiting the space of our possible solutions. These are the type of problems that the column generation technique aims to solve. The main idea is to work only with a sufficiently meaningful subset of variables (paths), forming the so-called restricted master problem (RMP). Then more variables are added by solving a subproblem, provided that they can improve the current solution. In more details, in column generation an iteration consists of:
(a) optimizing the restricted master problem in order to determine the current optimal objective function value.
(b) finding, if there is, a new variable (path) p that could improve the current optimal objective function value in order to introduce it to the master problem. The search for a better path corresponds to solving a sub-problem, in our case a constrained shortest path.

2.2 Restricted Master Problem (RMP):

The master problem corresponds to the actual problem we are trying to solve. In our case it is a vehicle routing problem which can be formed as a set partitioning problem with side constraints where the variables are associated with vehicle routes. The mathematical formulation of the problem is:

$$Min \sum_{r \in \omega} c_r y_r$$

subject to

$$\sum_{r \in \Omega} a_{ir} y_r = 1 \qquad \forall i \in P$$

$$y_r \in 0, 1 \qquad \forall r \in \Omega$$

An analytical explanation of this can be found in section 2 of the paper.

2.3 Pricing Subproblem:

A mentioned, in order to retrieve improving variables a constrained shortest path sub-problem must be solved. There are many approaches to solve it analyzed in "Shortest Path Problems with Resource Constraints" by Irnich and Desaulniers [4]. The approach used in the paper we implement is dynamic programming and more specifically labeling algorithms. The idea of these algorithms is to extend the path into all possible nodes and to discard non-useful paths by applying dominance rules which strongly depend on the path's structural constraints. The name "labeling" stems from the fact that the paths are encoded by labels. In this paper two possible sub-problems are proposed. The Elementary Shortest Path Problem with Time Windows, Capacity and Pickup and Delivery(ESPPTWCPD) or SP1 and the Shortest Path Problem with Time Windows, Capacity and and Pickup and Delivery(SPPTWCPD) or SP2. In this implementation we have used the SP1 as a sub-problem because, unlike SP2, it satisfies outfork and rechability inequalities with almost the same performance as SP2. A detailed explanation of these algorithms can be found in Branch and Cut and Price for the Pickup and Delivery Problem With Time Windows by Ropke and Cordeau(2008). Apart from SP1 which searches all the possible paths we can also use heuristics based on SP1 and local search algorithms.

2.4 Combining the master and the sub-problem:

The connection between the master problem solution and the sub-problem is done via the dual variables. The dual variables obtained by the RMP problem are used to calculate the reduced cost which is the measure of cost used in the sub-problem. The paths with negative reduced cost found are the ones that can possibly but **not necessary** improve the current solution.

3 Data

In this section we present the data used and the format of the input of the algorithm. The data used to test the algorithm are real and they are stored in the files requests.csv and DistanceMatrix.csv. **In the requests.csv we have the following information:**

- Day of depart,
- month of depart,
- time of depart,
- time zone,
- year,
- departure time in seconds,
- name of the departure location and
- name of the arrival and destination which are used in order to find the time needed to go from pickup point to the destination. This information is stored in DistanceMatrix.csv file.

  The input of the algorithm is:

- A digraph G with the cost as weights. In our test samples the cost is time.
- A dictionary storing:
  - P: set of pickup nodes
  - D: set of delivery nodes
  - Time windows : Dictionary with the node as key and the corresponding time window.
  - Min time window: the time of the earliest request
  - Load: Dictionary storing the load added in each node
  - N: Number of requests
  - vehicle maximum waiting time: set to 45 minutes
  - client maximum waiting time: set to 20 minutes
  - planning horizon: set to 600 minutes following the paper suggestion
  - vehicle capacity: set to 4 for simple vehicles
  - time window interval: set to 30 minutes
  - max ride time per user: this number corresponds to the supplement time allowed in order to deliver a client. We set that to 40 minutes and we use it to eliminate infeasible arcs applying the preprocessing step in section 5.1.2 in "A branch and cut algorithm for the Dial - a Ride problem" by Cordeau.

3.1 Data Preprocessing

All the data preprocessing and the creation of the necessary data structures occurs in read_data_script. Apart from the comments in the code some points to highlight are:

- Every request is formulated with i being the request node and n+i being the delivery node. There are cases that two requests have the same location. Even in these cases we encode them as being different locations putting as weight of the edge between these two nodes the min cost found in the graph – 10.
- During the preprocessing we delete the edges of the type (i,2n+1), (0,n+i) and the edges belonging to infeasible paths as described on 5.1.2 of "A branch and cut algorithm for the Dial-a-Ride problem" by Cordeau [1]. The infeasible paths are identified on the four eliminate_arc functions.
- The weight of the edges starting from the depot node is 10000 as we also try to minimize the number of vehicles used. The same holds for the weights of the edges starting from any delivery node to the target node(2*n +1) which are set to 1000.

4 Column Generation Algorithm Implementation

After the preprocessing step we execute the column generation algorithm. Before we generate columns we have to solve the master problem to obtain the first set of dual variables. For that we have to create an **initial pool of paths** that are valid and will give a feasible solution on the master problem. In this implementation the initial paths chosen are the single request path for each request. Thus, for n requests we generate all the paths of the form (depot, i, n+i, target).

4.1 Master Problem

We used MixedIntegerLinearProgram object of Sagemath to solve the linear programming problem (Simplex) and we use the GLPK backend as solver. We also use the backend of this solver to obtain the dual variables.

4.2 Pricing Problem Solvers

In order to find paths that may improve the current solution we need to identify the paths with negative reduced cost. We use both heuristics and sp1 which find an exact and optimal solution to the problem. A prerequisite in using SP1 and the heuristics is that the triangle inequality holds for both the weights of the graph and the reduced cost. In the beginning of the method we check if the triangle inequality holds for the costs on the weights of the graph. The process to check that is to find the shortest path between each pair of nodes and replace the corresponding edge weights with it. If the resulting weight is the same as before then the triangle inequality was already satisfied. For the reduced cost to satisfy the triangle inequality when we add additional inequality constraints in the master problem, we transform the costs following the instructions on the paper.

4.2.1 SP1

Reduced Cost

The reduced Cost is calculated from the beginning every time the master problem is solved. We use memoization technique by storing the reduced costs of each edge in a dictionary. The calculation method of the reduces cost is referred in Ropke and Cordeau(2009). As mentioned in the paper, when $\mu <>0$ the reduced costs do not satisfy the triangle inequality and we need to transform the reduced costs into costs that satisfy it. The transformation technique can be found in the paper in section 3.3. The functions used for that reason in the implementation are:

- get reduced cost,
- calculate reduced cost,
- calculate transformed reduced cost

Dominance Rule

In order to discard labels on the SP1 algorithm we use the dominance criterion mentioned in "Branch and Cut and Price for the Pickup and Delivery Problem with Time Windows" by Ropke and Cordeau(2008). But in our application we include capacity constraints which are not mentioned in the paper. After some experimentation we concluded that the <= rule, used in the other constraints, cannot be applied as it leads to a sub-optimal solutions and thus we use < when checking the dominance on the "load" constraint.

4.3 Heuristics:

As mentioned above the heuristics are used in an attempt to find negative reduced cost paths without executing the time consuming SP1 algorithm. The heuristics used are the following and they are executed in the following order:
(H3, H3_all, H1_small, H1_lowrank, LNS, H1_large, SP1)

- **H3:** This heuristic is similar to the SP1 algorithm but instead of expanding a path along all its neighbors we expand the cheapest one. So starting from a path containing one request we expand only one neighbor hoping to find a path with negative reduced cost. In this version of H3 we randomly select one starting request to expand.
- **H3_all:** This is the same heuristic as the previous one with the difference that we repeat the aforementioned process using all the requests as starting nodes.
- **H1_small:** H1 is the same as SP1 but we expand a limited number of nodes in each step. In order to select the best nodes in the expansion step we constructed a function called get_random_nodes that assigns probabilities on the first and second half of a sorted (on reduced cost) list of nodes and then selects a specific number of nodes to expand. In this version called "small" we expand only n//4 +2 number of nodes where n in the number of the requests.
- **H1_lowrank:** In this version of H1 we assign higher probability on the second half of the sorted list of nodes in order to also explore another part of the

solutions' space. Again we expand a small number of nodes in order to keep this heuristic efficient.

- **Large Neighborhood Search(LNS)** This heuristic uses the paths that are currently included in the solution and attempts to improve them by alternating between removing requests from the path and inserting requests into the path. The requests to remove are chosen randomly and the inserting requests are chosen using a randomized algorithm that favors low cost nodes and edges. The detailed description of this algorithm can be found in "Branch and Cut and Price for the Pickup and Delivery Problem with Time Windows" by Ropke and Cordeau(2008). The corresponding code is in the module LNS in class LNS that inherits from the PricingProblemHeuristic class.

- **H1_large:** This H1 heuristic is exploring a large area of the solution by expanding n//2 + 3 nodes in each expansion step. It is the most time consuming heuristic but still faster than SP1. A note on the H1 heuristic is that we use an optimization technique in which we use the previously retrieved paths and we insert them from the beginning both in labels ending at and on the exploration queue of the algorithm. In this way we make the execution faster by using the previously found results.

4.4 Optimization Ideas Tested

On this implementation we have tried the following optimization ideas [2] to make the column generation more efficient.

- We created a dictionary of labels ending at to check fast the paths with the same ending node when applying the dominance rules.

- When executing the heuristics and SP1 we store the reduced costs and the transformed reduced cost in a dictionary in order to avoid to recalculate them every time we execute the next heuristic. Thus the reduced cost is calculated once every time we solve the master problem.

- In the current implementation we first create the new Label and then check the dominance conditions to decide whether to introduce the new path to the queue. We tested whether checking the dominance conditions before creating the Label would improve efficiency but it didn't.

- In the paper an optimization called Label Elimination is proposed which is running a TSP algorithm within SP1 to see if there is enough time to traverse the delivery nodes of the open requests and if not to discard the path. After testing it we observed that it didn't succeed into discarding any path.

- A fruitful optimization was to retrieve many negative reduced cost paths per iteration on H3 and H1 heuristic. In H1 we also include a time criterion to stop searching for more negative paths if one is already retrieved and a time threshold is over-exceeded.

4.5 Branch and Bound

The branching rules are chosen to be compatible with the algorithms used to solve the pricing problem as during the branching we will be trying to generate more paths by solving the pricing problem. In this algorithm, we use two branching rules that add a single cut on the xij variables to the master problem. The first branching strategy was proposed by Naddef and Rinaldi (2002) for the CVRP problem and the second strategy was proposed by Desrochers, Desrosiers, and Solomon (1992) [3] and is often called branching on the number of vehicles.

In the second strategy we select a set of nodes S such that $x(\delta^+(S))$ is as far as possible from the nearest integer where x is equal to one when the arc is used in the solution and δ^+ is the number of the outgoing edges from the set. The nodes included in the set do not need to be connected with each other. After we find a good candidate for such a set using a greedy approach we create two branches, $x(\delta^+(S)) <= \lceil x(\delta^+(S)) \rceil$ and $x(\delta^+(S)) >= \lfloor x(\delta^+(S)) \rfloor$. This branching strategy is described in detail in the paper "A new optimization algorithm for the vehicle routing problem with time windows". Due to lack of time we have not implemented this strategy in the code.

The first branching strategy is the one often called branching on the number of vehicles. In this strategy we calculate the number of outgoing edges from node 0 and if the sum of these edges is not integer we create two branches, $x(\delta^+(0)) <= \lceil x(\delta^+(0)) \rceil$ and $x(\delta^+(0)) >= \lfloor x(\delta^+(0)) \rfloor$.

As previously mentioned during the branching we attempt to generate more paths by solving the pricing problem.

5 Important Notes

The cutting planes are not implemented in the code as the algorithm was too slow and we focused on improving its efficiency. Normally the cutting planes are added after the column generation process has finished. So implementing them would not improve the efficiency of finding an initial fractional solution. But it is still useful to mention that the simplex solver when adding constraints has been implemented in the branch and bound and thus it is relatively easy to implement the cutting planes mentioned by replacing the function "solve simplex" with the function "solve_simplex_branch_lev1" of the "branch_and_bound" module.

6 Future Work

Some basic tasks that can be done as future work are the implementation of the first branch and bound strategy and the addition of the cutting planes of branch and cut algorithm. But as the algorithm implemented so far is relatively slow on the real data it would be more significant to find ways to accelerate the existing code. Another idea worth testing would be to perform clustering on the data using time and the coordinates of the requests in order to determine subsets of requests smaller than 10 for which separate optimization would lead to a local optimum close to the global one. It is also useful to mention that currently the algorithm also optimizes the number of vehicles used by choosing the smaller number possible. If we were given a fixed number of vehicles we could possibly add tighter constraints that would lead to a faster execution of the algorithm.

References

[1] Jean-François Cordeau. "A Branch-and-Cut Algorithm for the Dial-a-Ride Problem". In: *Operations Research* 54 (June 2006), pp. 573–586. DOI: 10.1287/opre.1060.0283.

[2] Guy Desaulniers, Jacques Desrosiers, and Marius Solomon. "Accelerating Strategies in Column Generation Methods for Vehicle Routing and Crew Scheduling Problems". In: 15 (Jan. 1999). DOI: 10.1007/978-1-4615-1507-4_14.

[3] Martin Desrochers, Jacques Desrosiers, and M Solomon. "A New Optimization Algorithm for the Vehicle Routing Problem with Time Windows". In: *Operations Research* 40 (Apr. 1992), pp. 342–354. DOI: 10.1287/opre.40.2.342.

[4] Stefan Irnich and Guy Desaulniers. "Shortest Path Problems with Resource Constraints". In: Mar. 2006, pp. 33–65. DOI: 10.1007/0-387-25486-2_2.

[5] Stefan Ropke and Jean-François Cordeau. "Branch and Cut and Price for the Pickup and Delivery Problem with Time Windows". In: *Transportation Science* 43 (Aug. 2009), pp. 267–286. DOI: 10.1287/trsc.1090.0272.