1 User Manual

To solve a VRP problem with a column generation method, we will decompose our problem into a master problem, which is a set-covering formulation of a vehicle routing problem, and a sub-problem, which is a resource-constrained shortest path problem. Below I will provide their exact formulations and the algorithms I used to solve these problems.

There is usually only one depot in VRP problems where routes start and finish, but in our case, we duplicate the depot to a "starting depot" and an "ending depot" just for convenience of modeling, we are not actually changing the setup.

1.1 Master problem: set-covering formulation of a vehicle routing problem

Suppose there are n customers with indexes $\{1, .., n\}$ in a directed graph G we want to serve, and graph G also contains a starting depot with index 0 and ending depot with index $n + 1$.

The problem requires us to assign a certain number of vehicles to serve all customers, each vehicle needs to depart from the starting depot and arrive at the ending depot, however, there is no constraint on the number of vehicles you can use (although the "soft upper-bound" should be n, which means that you are serving each customer with an unique vehicle.). Each customer has a strictly positive service demand, and each vehicle has a capacity constraint on this demand. All vehicles have the same service capability, and in our case, by default, we set it to:

$$\textbf{max}\{\text{max(All customers' demand)}, 6 \times n/4\}$$

You can also input a number between 0 and 1 (boundaries not included)to set the capacity to that proportion of the total capacity demand of all customers or input a number more than 1 to set the capacity limit to that number. In all cases, if the number you inputted ends up being smaller than max(All customers' demand), it will automatically be scaled up to that number to ensure that the problem is feasible.

Each edge in the graph has a time cost that is computed from the Euclidean distance from the graph we generated, and the objective is to find a set of vehicles such that the time cost is minimized and the service capacity is not violated on each vehicle.

In summary, we can express the problem in the following way:

Notations

r: Index for routes. A feasible route must satisfy three conditions:

1. Start from the starting depot and end at the ending depot;
2. Its cost must not exceed the capacity constraint;
3. It is elementary, which means that no vertex is visited twice.

c_r: cost of route r

x_r: binary variable indicating if we are using route r

R: The set of routes r's r

V: The set of customer vertices.$|V| = n$.

$R_i : \{r \in R : \text{r visits vertex } i\}$

$R'_i : \{\text{subsets of } R_i\}$

Formulation

$$\begin{aligned} \min \quad & \sum_{r \in R'} c_r x_r \\ \text{s.t.} \quad & \sum_{r \in R'_i} x_r \geq 1 \qquad \forall i \in V \quad (1) \\ & x_r \geq 0 \qquad\quad \forall r \in R \quad (2) \end{aligned}$$

Variable Definition

$$x_r = \begin{cases} 1 & \text{if we use route } r \\ 0 & \text{otherwise} \end{cases}$$

The formulation can be interpreted as follows: The objective function minimizes the sum of costs associated with the active routes. There are two sets of constraints. The first set, known as the "covering constraints," ensures that all customers are visited at least once. The number of these constraints is equal to the number of customers, and there is a clear one-to-one correspondence between each customer and its associated covering constraint. The second set consists of the standard non-negativity constraints for the variables x_r. It is not necessary to include the constraint $\sum_{r \in R'_i} x_r = 1$, as visiting a customer more than once would never be optimal.

In my implementation, I will use Gurobi to solve a continuous relaxed version of this master problem, which simply expands the domain of x_r to the non-negative real-line. However, the number of routes(which is the number of variables) grows factorially with the number of customers, so we will use a column generation algorithm to include only the relevant routes.

The algorithm, however, needs to be initialized in a feasible way to start working. Denote the initial set of routes as R_{start}, we are using the following algorithm to initialize our

column pool:

Algorithm 1 Initializing routes

Input: Number of customers $n > 0$
Output: A set of routes R_{start}
$R_{\text{start}} \leftarrow \emptyset$
for $i \leftarrow 1$ **to** n **do**
  Create new route r_i by linking starting depot $\rightarrow$ customer $i \rightarrow$ ending depot
  $\text{R}_{\text{start}} \leftarrow R_{\text{start}} \cup r_i$
end
return R_{start}

There are at least three compelling reasons to use this algorithm: first, the number of routes in the initial column pool is equal to the number of customers, which ensures a manageable number of routes. Second, the routes are guaranteed to be feasible; if route r_i for customer i is not feasible, it indicates that there is no possible way to serve this customer without exceeding the capacity constraint, making the problem itself infeasible. Third, all customers are guaranteed to be served, ensuring a complete and viable solution.

Replace $R^{'}$ with R_{start} we can start the first iteration of the algorithm, then, to generate more routes(columns) that will improve the solution, we go to the subproblem, which we will model as a resource-constrained shortest path problem.

1.2 Subproblem: resource-constrained shortest path problem

We now have our master problem initialized and solved for the first time (which solution should be $x_r = 1 \forall r \in \{1, .., n\}$ with an "optimal" objective value of $\sum_{r \in \{1,..,n\}} c_r$), we need to find new variables(columns) to further improve our solution. To do this, we will need to start from analysiing the dual problem of the master problem.

Let $\pi_i \forall i \in V$ denote the dual variables associated with the covering constraints. Following the intuition of Lahaie (2008), we derive the dual problem as the following:

Variable definition

$$\delta_{ir} = \begin{cases} 1 & \text{if route } r \text{ passes customer i} \\ 0 & \text{otherwise} \end{cases}$$

Formulation

$$\begin{aligned} \max & \sum_{i \in V} \pi_i \\ \text{s.t.} \quad & c_r - \sum_{i \in V} \delta_{ir} \pi_i \geq 0 \quad \forall r \in R \\ & \pi_i \geq 0 \quad \forall i \in V \end{aligned}$$

Now we are looking for routes r such that $c_r - \sum_{i \in V} \delta_{ir} \pi_i < 0$ because we know that following the intuition of the Dantzig-Wolfe decomposition these routes can improve the solution.

To find such route, we remodel the problem as the following:

Formulation

$$\begin{aligned} \min_{r \in R} \quad & c_r - \sum_{i \in V} \delta_{ir} \pi_i \\ \text{s.t.} \quad & \delta_{ir} \in \{1, 0\} \end{aligned}$$

And to solve this problem, we consider this as a resource-constrained shortest path problem(RCSPP) which goal is to find the shortest path from the starting depot to the ending depot after modifying the original graph with the following algorithm:

Algorithm 2 Modify graph

Input: A directed graph G with edge set E & A set of dual multipliers $\{\pi_i : i \in \{1, ..., n\}\}$
Output: An updated graph $G^{'}$
$G^{'} \leftarrow G$
for $i \leftarrow 1$ **to** n **do**
  $E_i^{'} \leftarrow$ set of arcs in $E^{'}$ that ends at vertex(customer) i
  for $e_i^{'} \in E_i^{'}$ **do**
    $c_i^{'} \leftarrow$ cost of edge $e_i^{'}$
    $c_i^{'} \leftarrow c_i^{'} - \pi_i$
  end
end
return $G^{'}$

After this modification, we can solve the RCSPP and check if the path find has a negative cost. If yes, then we have found a new route that can reduce the total cost, hence we add this route back to the master problem and resolve it.

We will repeat this master problem-subproblem loop until no columns with negative cost can be found in the subproblem, which indicates that we have solved the continuous-relaxed VRP to optimal, then, we can use the columns generated to solve a binary integer programming problem to get a primal bound.

1.3 Algorithms to solve the subproblem

However, this is just the starting point of my analysis. Solving the master problem is not particularly difficult, as it is simply a continuous linear programming problem. Therefore, to improve the efficiency of solving this problem, it is crucial to find an effective method for solving the subproblem and generating new columns.

To solve the RCSPP problem, we use a label-setting algorithm based on dynamic programming, where paths are represented by labels that contain sets of resources. Irnich and Desaulniers (2005) provides an excellent survey on this type of algorithm. For a detailed explanation, refer to their work; here, we will provide a brief summary tailored to our specific case.

In the document of the boost graph library in C++, Drexl (2006) described the algorithm as the following:

> "The (RCSPP)functions are an implementation of a priority-queue-based label-setting algorithm. At each iteration, the algorithm picks a label l from a priority queue (the set of unprocessed labels) and checks it for dominance against the current set of labels at the vertex i where l resides. If l is dominated by some other label residing at i, l is deleted from the set of labels residing at i. If l is undominated, it is extended along all out-edges of i. Each resulting new label is checked for feasibility, and if it is feasible, it is added to the set of unprocessed labels and to the set of labels residing at the respective successor vertex of i. If a new label is not feasible, it is discarded. The algorithm stops when there are no more unprocessed labels. It then checks whether the destination vertex could be reached (which may not be the case even for a strongly connected graph because of the resource constraints), constructs one or all Pareto-optimal (i.e., undominated) s-t-path(s) and returns. "

The concept of dominance mentioned above is a widely used term when describing labeling algorithm Boland et al. (2006), so instead of a rigorous mathematic definition I would like to provide a short but easy-to-understand interpretation: label l_a dominates label l_b if and only if l_a is as good as l_b in all aspects and at least strictly better than l_b in one aspect.

When extending a label from one vertex to another, a cost associated with the arc must be paid. Following the intuition of **Beasley1989<empty citation>**, where they added an extra resource for each customer, in our version of the label-setting algorithm, we define a vector of resource costs for each arc e_{ij} in the graph G'. This vector has a length of $n + 2$ and is structured as follows:

$$\{d_j, i, 0, 0, 0, ..., 1, ...0, 0\}$$

where:

d_j : service demand of customer$j$

i : this resource is recording the index of the last vertex we visited. Once we extend the label l from vertex i to j, this index will replace the original one in label l This way, we can prevent loops with only two vertices

$\{0, 0, 0, .., 1, ..0, 0\}$: This is a binary vector in which only the jth component is 1, other components are all zeros. This is like a indicator of "if customer j has already been visited". Consequently, we can only visit j if the vertex j has never been visited before.

As per the above explanation, all non-dominated routes are retained. Therefore, we can consider all non-dominated routes with negative costs at the ending depot, rather than just selecting the single "shortest path".

We will consider the basic elementary labeling algorithm and its four variations, which are described below. These variations are closely related, and the development of each one is often based on the results observed from the previous version.

Algorithm 3 Elementary labeling algorithm(ELA)

Input: A graph G
Output: A set of columns and related solution
Initialize routes by using Algorithm 1.
while *new columns can still be found* **do**
    Solve the master problem with Gurobi.
    Adjust the edge costs with Algorithm 2.
    Solve the subproblem with the basic elementary labeling algorithm.
    Take all non-dominated paths with negative cost as new columns.
end
With the columns generated, solve an integer version of the problem to get a primal bound.

Algorithm 4 State-Space relaxed ELA(SSRELA)

Input: A graph G
Output: A set of columns and related results
Initialize routes by using Algorithm 1.
while *new columns can still be generated* **do**
  Solve the master problem with Gurobi.
  Adjust the edge cost with Algorithm 2.
  Solve the state-space relaxation RCSPP.
  while *there exists non-elementary paths* **do**
    Add resources for duplicated vertices.
    Resolve the RCSPP.
  end
  Take all paths with negative cost.
end
With the columns generated, solve an integer version of the problem to get a primal bound.

The "state-space relaxed ELA" refering to the idea of Kohl (1995), in which we start from adding no resource for vertices and incrementally add node resources until all paths with negative costs are elementary. The node resources added in one iteration will not be carry over to the next iteration.

Algorithm 5 K-best SSRELA

Input: A graph G
Output: A set of columns and related results
Note: This algorithm is the same as Algorithm 4, but when adding new columns, only the k routes with the least cost are taken.
Initialize routes by using Algorithm 1.
while *new columns can still be generated* **do**
  Solve the master problem with Gurobi.
  Adjust the edge cost with Algorithm 2.
  Solve the state-space relaxation RCSPP.
  while *there exists non-elementary paths* **do**
    Add resources for duplicated vertices.
    Resolve the RCSPP.
  end
  Take the k routes with the least cost as new columns.
end
With the columns generated, solve an integer version of the problem to get a primal bound.

Algorithm 6 "Ignoring" algorithm

Input: A graph G

Output: A set of columns and related results

Initialize routes by using Algorithm 1.

```
while true do
    Solve the master problem with Gurobi.
    Adjust the edge cost with Algorithm 2.
    Solve the state-space relaxation RCSPP.
    if we found elementary routes with negative cost then
        Take these routes as new columns.
        continue
    end
    else
        Add resources for duplicated vertices.
        if no duplicated vertices found then
            break
        end
        Resolve the RCSPP.
        Go back to the start of the first if statement.
    end
end
```

With the columns generated, solve an integer version of the problem to get a primal bound.

I call algorithm 6 the "ignoring" algorithm because instead of solving the subproblem to optimal at each iteration, we simply "ignore" any non-elementary paths at the ending depot as long as we can find elementary paths with negative cost. The idea is to generate new columns as fast as possible, instead of exploiting the solution space at each iteration. This modification can drastically improve the efficiency.

Algorithm 7 "Ignoring" algorithm with multiplicity

Input: A graph G

Output: A set of columns and related results

Note: This algorithm has the same setup as Algorithm 6, but when adding resources for duplicated vertices, only add the resource for the vertex that has the highest multiplicity.

In Algorithm 7 the concept of multiplicity is inspired by a similar concept in Boland et al. (2006). Here we define the multiplicity as: "the number of times a vertex i appears as a duplicated vertex in all non-dominated routes with negative cost".

References

Boland, N., Dethridge, J., & Dumitrescu, I. (2006). Accelerated label setting algorithms for the elementary resource constrained shortest path problem. *Operations Research Letters*, *34*(1), 58–68.

Drexl, M. (2006). Boost graph library: R_c_shortest_paths [Accessed: 2024-06-05]. *Boost.org*. https://www.boost.org/doc/libs/1_85_0/libs/graph/doc/r_c_shortest_paths.html

Irnich, S., & Desaulniers, G. (2005). Shortest path problems with resource constraints. In G. Desaulniers, J. Desrosiers, & M. Solomon (Eds.), *Column generation* (pp. 33–65). Springer US.

Kohl, N. (1995). *Exact methods for time constrained routing and related scheduling problems* (Publication No. IMM-PHD-1995-16) [Ph.D. dissertation]. Technical University of Denmark.

Lahaie, S. (2008). How to take the dual of a linear program [Lecture notes].