How to Implement Column Generation for Vehicle Routing

Sean Kelley

26 January 2023

Overview

Why Implement Column Generation for Vehicle Routing

Context

- Vehicle Routing Problems (VRPs) are hard to solve directly (e.g. as a mixed-integer program (MIP)).
- There exist plenty of heuristics that can quickly come up with solutions to VRPs.
- Many heuristics lack a MIP solver's ability to leverage one solution in finding a better one or declaring it's the best.
- We would like a middle ground between these two approaches:
  - Finding solutions quickly
  - Iteratively improving upon previously found solutions

Solution methods for VRPs often apply column generation for its ability to do exactly this.

Emperical Motivation

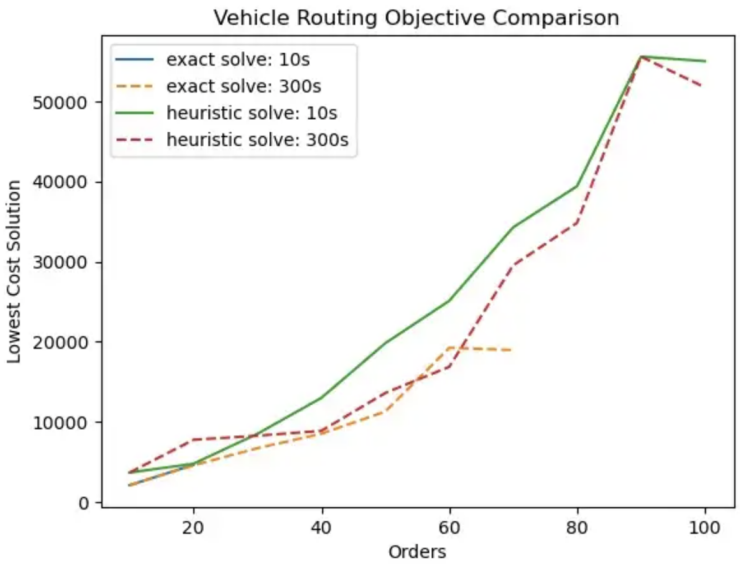

Figure 1: Comparing the best found solutions at 10 seconds and 5 minutes for VRPs solved directly (exact) or with column generation (heuristic)

- Column generation heuristics can uncover solutions more quickly than direct solves.
- For example, consider the following:
  - **a.** 10 seconds of column generation heuristic (green)
  - **b.** Five minutes of directly solving (orange)
- Notice **a.** regularly finds a solution that is 50% as good as **b.** but in 3% time.
- **a.** becomes advantageous past 70 orders when **b.** can no longer find a solution.

Column generation heuristics can help us continue to find "good" solutions even problems become intractible to solve directly.

Vehicle Routing Problem Preliminaries

VRPTW Parameters, Variables, and Objective

- For simplicity, we will work with the VRP with Time Windows (VRPTW).
- The VRPTW has the following parameters:
  - c_{ij} covers cost to travel from node i to node j.
  - There are $n+1$ nodes (n customers and 1 depot (indexed 0)).
  - There are p vehicles.
  - $[a_i, b_i]$ is the time window of customer i.
  - t_{ij} denotes the time to get from customer i to customer j (with service at customer i included).
- The VRPTW has the following variables:
  - s_i denotes the time that a vehicle starts serving customer i.
  - x_{ijk} has a value of 1 if the arc from node i to node j is in the optimal route driven by vehicle k and 0 otherwise.
- The VRPTW has the following objective:

$$\text{Min} \sum_{k=1}^{p} \sum_{i=0}^{n} \sum_{j=0}^{n} c_{ij} x_{ijk}$$

VRPTW Constraints

The VRPTW has the following constraints:

- Each vehicle leaves each node that it enters:

$$\sum_{i=0}^{n} x_{ijk} = \sum_{i=0}^{n} x_{jik} \qquad \forall j \in \{0, ..., n\}, \quad k \in \{1, ..., p\} \tag{1}$$

- Every node is entered once:

$$\sum_{k=1}^{p} \sum_{i=0}^{n} x_{ijk} = 1 \qquad \forall j \in \{1, ..., n\} \tag{2}$$

- Every vehicle leaves the depot at most once:

$$\sum_{j=1}^{n} x_{0jk} \leq 1 \qquad \forall k \in \{1, ..., p\} \tag{3}$$

VRPTW Constraints

- Vehicle capacities are respected:

$$\sum_{i=0}^{n}\sum_{j=1}^{n} q_j x_{ijk} \leq Q \qquad \forall k \in \{1, ..., p\} \tag{4}$$

- Arrival happens after travel time is added to previous departure:

$$s_i + t_{ij} - M * (1 - x_{ijk}) \leq s_j \qquad \forall i \in \{0, ..., n\},\ \ j \in \{1, ..., n\},\ \ k \in \{1, ..., p\}$$
$$M = max\{b_i + t_{ij} - a_i\} \qquad \forall i, j \in \{0, ..., n\} \tag{5}$$

- The time window of each customer is respected:

$$a_i \leq s_i \leq b_i \qquad \forall i \in \{0, ..., n\} \tag{6}$$

Since the VRPTW is intractible to solve directly for many production instances, reformulating can help us find something solvable.

Example Solution

From this formulation, if we were to plot the routes formed by the solution, we might see something like the following:

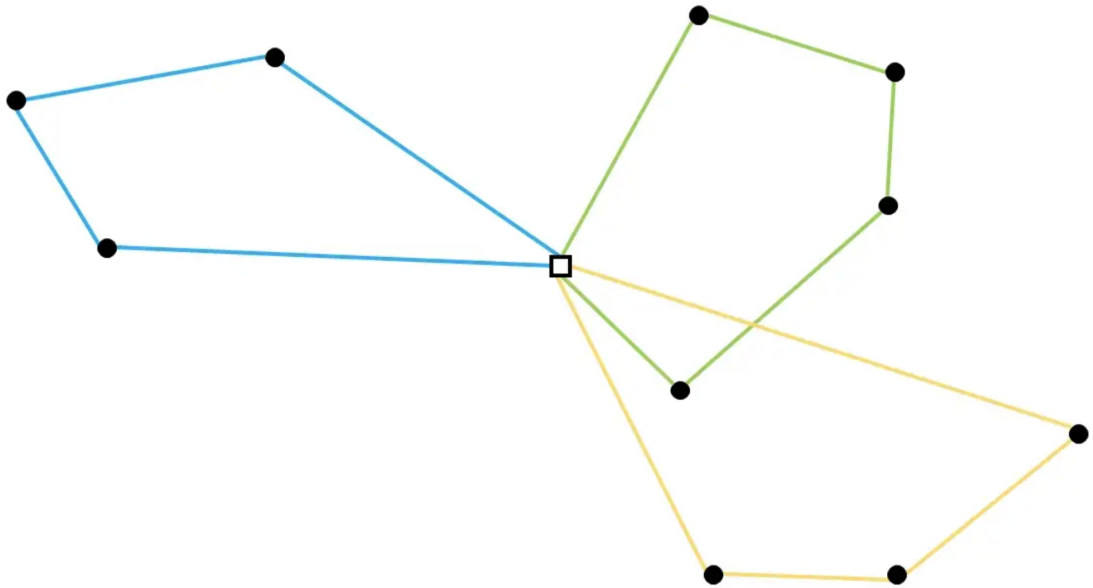

Figure 2: A solution to a vehicle routing problem encodes each route with a distinct color and stop order by the edges between customers.

Set Covering Problem (SCP) Reformulation

The SCP has the following parameters:

- Ω is the set of feasible vehicle routes.
- c_k is the cost of route $r_k \in \Omega$.
- $a_{jk} = 1$ if route r_k visits customer j and 0 otherwise.
- $b_{ijk} = 1$ if route r_k travels directly from customer i to customer j and 0 otherwise.
- Note the following relationships:

$$c_k = \sum_{i=0}^{n} \sum_{j=0}^{n} c_{ij} b_{ijk}$$

$$a_{jk} = \sum_{i=0}^{n} b_{ijk}$$

The SCP has the following variable:

- $z_k = 1$ if route r_k is selected and 0 otherwise.

The SCP has the following objective:

$$\text{Min} \sum_{k \in \Omega} c_k z_k$$

Subject to:

$$\sum_{r_k \in \Omega} a_{jk} z_k \geq 1 \qquad \forall j \in \{1, ..., n\} \quad (7)$$

The SCP is equivalent to the VRPTW, albeit often much larger. However, the SCP can be restricted to a much smaller problem while maintaining solution quality.

Restricted Set Covering Problem (RSCP) Reformulation

The RSCP, w.r.t $\Omega' \subset \Omega$, (RSCP-Ω') is formulated as follows:

$$\text{Min} \sum_{k \in \Omega'} c_k z_k$$

$$\text{s.t.} \sum_{r_k \in \Omega'} a_{jk} z_k \geq 1 \qquad \forall j \in \{1, ..., n\} \quad (8)$$

$$z_k \in \{0, 1\} \qquad \forall k \in \Omega'$$

The RSCP-Ω' can efficiently approximate the VRPTW when:

- $|\Omega'|$ is small.
- Ω' covers all customers at low total cost.

We need to find a reasonable Ω'. For that, we will leverage column generation.

Column Generation Preliminaries

A Column Generation Heuristic for the VRPTW

Our column generation heuristic can be broken down as follows:

- Initialize
  - Find an initial set of feasible covering routes, Ω'_0, which could be:
    - The set of singleton routes.
    - The K nearest neighbors for each customer.
- Iterate (let i be the iteration index)
  - Solve the LP relaxation of RSCP-Ω'_i.
  - Find $\Gamma_i \subset \Omega \setminus \Omega'_i$, a set of improving routes for the RSCP-Ω'_i.
  - Let $\Omega'_{i+1} = \Omega'_i \cup \Gamma_i$.
  - Increment i and repeat the above until $\Gamma_i = \emptyset$ or we are satisfied.
- Cover
  - Solve the final RSCP-Ω_i.
  - Return the chosen routes.

We can determine if a route improves the solution for the RSCP-Ω_i by calculating its reduced cost.

Dual LP Relaxation for the SCP

Parameters:

- Let A represent the matrix of a_{jk}.
- $A_k = [a_{1k}, ..., a_{nk}]^T$, i.e. stops of route r_k.
- c is vector of route costs $[c_1, ..., c_{|\Omega|}]^T$.

Sets:

- B is the indices of basis variables for the solution to the RSCP-Ω_i LP relaxation.
- N is remaining variable indices in the SCP.

Variables:

- y is row reduced costs.
- s is column reduced costs.

Formulation:

$$\begin{aligned} &\text{Max } 1^T y \\ &\text{s.t. } A^T y + s = c \\ &\qquad y, s \geq 0 \end{aligned}$$

From Linear Programming (LP) theory, we can conclude the following about the SCP solution with basis B:

- The reduced cost of z_k (a.k.a. s_k) is $c_k - A_k^T y$ for all $k \in N$.
- If $c_k - A_k^T y < 0$, then adding r_k to RSCP-Ω_i will improve its objective.

A Critical Observation

- We can evaluate the reduced costs of routes not in the RSCP with the dual solution to the SCP.
- Done explicitly, this creates the size of problem we're trying to avoid.
- However, we can do so implicity by solving the Pricing Problem, which includes the following:
  - Minimize the reduced costs of a route.
  - Subject routes to feasibility defined in the VRPTW.
  - Return routes found during the optimization problem with negative costs.

Pricing Feasible Routes for the VRPTW

Connecting Previous Formulations to the Pricing Problem

By the definition of b_{ijk}

$$b_{ijk} = x_{ijk} \qquad \forall i, j \in \{0, ..., n\},\ k \in \Omega$$

By the definition of a_{jk}

$$a_{jk} = \sum_{i=0}^{n} x_{ijk} \qquad \forall j \in \{0, ..., n\},\ k \in \Omega$$

Then, reduced costs can be rewitten as

$$c_k - A_k^T y = \sum_{i=0}^{n}\sum_{j=0}^{n} c_{ij} x_{ijk} - \sum_{j=1}^{n}\left(y_j \sum_{i=0}^{n} x_{ijk}\right)$$

The Pricing Problem differs from the VRPTW in the following ways:

- Index k is removed from arc variables, x, since we solve for a single route.
- Constraint 2 is removed since it is accounted for in the (R)SCP.
- The objective is to minimize the **reduced** cost of the route.

Consequently, the Pricing Problem is a factor of $\frac{1}{k}$ as large as the original VRPTW, which is why our column generation heuristic is tractible.

Pricing Problem Formulation

We seek to minimize the reduced cost of a given route:

$$\text{Min} \sum_{i=0}^{n}\sum_{j=0}^{n} c_{ij}x_{ij} - \sum_{j=1}^{n}\left(y_j \sum_{i=0}^{n} x_{ij}\right)$$

Subject to:

- The route leaves each node that it enters:

$$\sum_{i=0}^{n} x_{ij} = \sum_{i=0}^{n} x_{ji} \qquad \forall j \in \{0, ..., n\} \tag{9}$$

- The route leaves the depot at most once:

$$\sum_{j=1}^{n} x_{0j} \leq 1 \tag{10}$$

Pricing Problem Formulation

- The route respects vehicle capacity:

$$\sum_{i=0}^{n}\sum_{j=1}^{n} q_j x_{ij} \leq Q \tag{11}$$

- Arrival happens after travel time is added to previous departure:

$$s_i + t_{ij} - M * (1 - x_{ij}) \leq s_j \qquad \forall i \in \{0, ..., n\},\ \ j \in \{1, ..., n\} \tag{12}$$

- The route respects the time window of each customer:

$$a_i \leq s_i \leq b_i \qquad \forall i \in \{0, ..., n\} \tag{13}$$

By collecting all negative solutions found while solving the Pricing Problem, we generate a set of routes improving the RSCP-Ω_i.

Why Formulate the Pricing Problem as a MIP?

Pricing problems are often formulated as Dynamic Progams (DPs). In practice, I've seen MIP formulations implemented instead for the following reasons:

- Clients have diverse constraints in their pricing problems, necessitating rich modeling capabilities.
- MIPs often find many solutions (i.e. routes) in a single solve.
- MIPs can be resolved with slight adjustments to the constraints, generating additional routes.
- Each MIP solve can possibly be warm started.

Column Generation Closing Thoughts

Recap: A Column Generation Heuristic for the VRPTW

Our column generation heuristic can be broken down as follows:

- Initialize.
  - Find an initial set of feasible covering routes, Ω'_0, which could be:
    - The set of singleton routes.
    - The K nearest neighbors for each customer.
- Iterate (let i be the iteration index).
  - Solve the LP relaxation of RSCP-Ω'_i.
  - Find $\Gamma_i \subset \Omega \setminus \Omega'_i$, a set of improving routes for the RSCP-Ω'_i **by solving the Pricing Problem**.
  - Let $\Omega'_{i+1} = \Omega'_i \cup \Gamma_i$.
  - Increment i and repeat the above until $\Gamma_i = \emptyset$ or we are satisfied.
- Cover.
  - Solve the final RSCP-Ω_i.
  - Return the chosen routes.

A Note on Branch-and-Price

Our column generation heuristic is unlikely to yield an optimal solution to the VRPTW. Branch-and-Price is often used if one is desired.

- Branch-and-Price retains tractibility by solving RSCP LP relaxations and Pricing Problems.
- It differs from our column generation heuristic as follows:
  - If the LP relaxation solution to the final RSCP-Ω_i is not integer feasible, branching occurs, and our algorithm repeats on each branch.
  - One branch requires all new routes to visit a location j immediately after location i (i.e. $x_{ijk} = 1$).
  - The other branch prohibits all new routes from visiting a location j immediately after location i (i.e. $x_{ijk} = 0$).
- Consequently, Branch-and-Price takes longer to run and implement.
- Dominique Feillet's "A Tutorial on Column Generation and Branch-and-Price for Vehicle Routing Problems" provides further details.

Implementation Details

Important Parameters

The following are paramaters input to our column generation algorithm:

Parameter	**Trade-Off**
Solutions per Pricing Problem solve	RSCP solve time vs. Pricing Problem solves
Pricing Problem MIP gap	quantity vs. quality of columns generated
Pricing Problem time limit	quantity vs. quality of columns generated
Relative RSCP LP resolve improvement	solution quality vs. solve time
Ratio of time iterating vs. covering	number of routes found vs. quality of routes chosen

The values of the following parameters can vastly change solve time and solution quality. Practitioners should tune each to ensure desired results.

Important GurobiPy API Endpoints

The following "out of the ordinary" GurobiPy API Endpoints were used in the implementation:

Endpoint	**Meaning**
Constraint.pi	Row reduced costs
PoolSolutions	Max number of solutions to cache
Model.SolCount	Number of solutions cached
Model.SolutionNumber	Index of the current solution
Model.PoolObjVal	Objective value of the current solution
Column()	Object that sets nonzero constraint coefficients for new variables

For practitioners seeking to use a different MIP solver, one will want its API to include attributes similar to the above.

GitHub Repo

Source code for the implementation of our column generation heuristic can be found on GitHub (spkelle2/vehicle_routing_column_generation).

Questions