Sean Kelley Follow
Sep 7 · 8 min read · Listen

How to Implement Column Generation for Vehicle Routing

If you're reading this post, chances are, you're like me prior to the week before I wrote this.

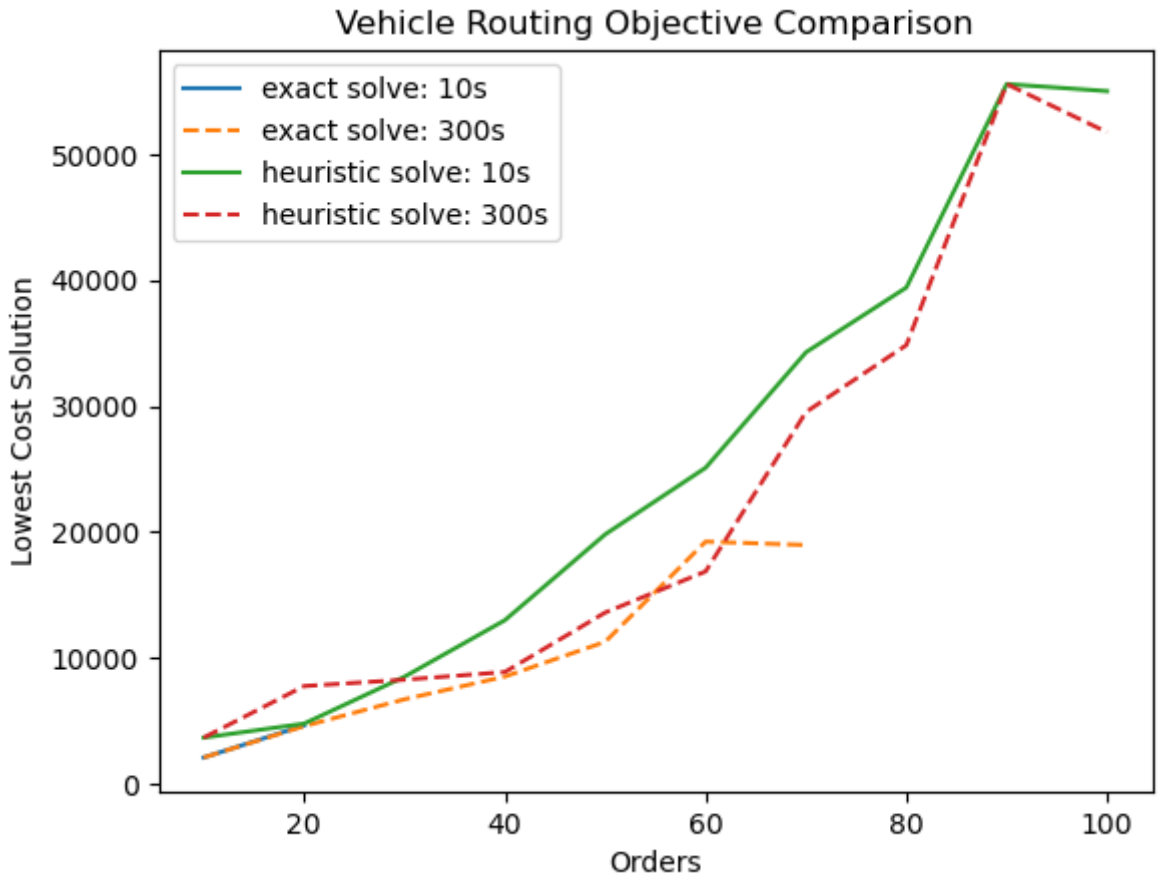
You've modeled a variant of the Vehicle Routing Problem (VRP), but, when you solve it directly with a mixed-integer programming (MIP) solver, it runs indefinitely without generating a single solution. Faced with a deadline to find one, you start to sweat as you search for possible workarounds. As you dig, you come across suggestions to use column generation to speed up the solve, but you're still stuck: the explanations are dense, and an implementation you can reference is nowhere to be found.

If this is you, read on. We'll address both below and get you a working implementation in no time. Along the way, we'll look at a couple other simpler performance improving techniques that may just do the trick for you, too.

Why Implement Column Generation for Vehicle Routing

As mentioned above, VRPs are hard to solve directly, and, in particular, it can be a challenge to find even a single solution in this manner. On the other hand, there exist plenty of heuristics that can quickly come up with solutions to VRPs, but many lack a MIP solver's ability to leverage one solution in finding a better one or declaring it's the best. What we would like then is to strike a middle ground between these two approaches: finding solutions quickly but also iteratively improving them. Solution methods for VRPs often apply column generation for its ability to do exactly this.

directly. Note that the line for a solution method vanishes when it no longer finds a solution within its time limit for a given number of orders.



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

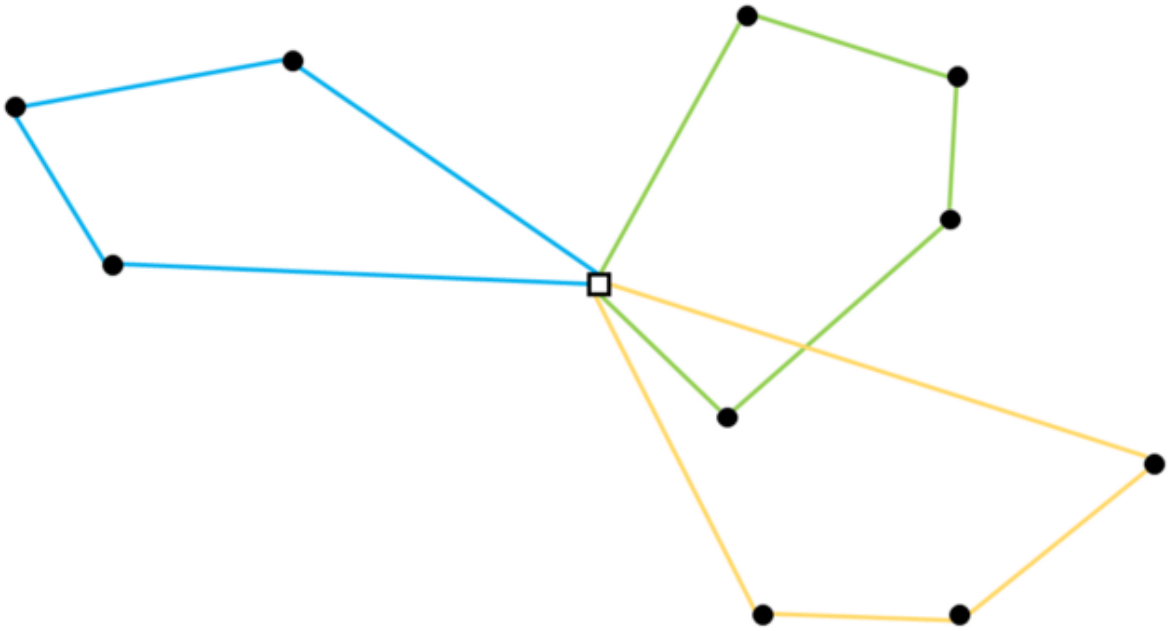
For problems with 70 orders or fewer, we can see that 10 seconds of the column generation approach (labeled “heuristic solve: 10s”) regularly finds a solution that is 50% as good as the best one found from five minutes of solving the VRP directly (labeled “exact solve: 300s”). Although a solution of 50% quality may not be very compelling, the fact it was found in 3% of the time might be when considering that 10 seconds of directly solving (labeled “exact solve: 10s”) regularly fails to find a solution. Also, bear in mind this column generation implementation is for illustration and has much room to be improved.

Further, notice the lines for direct VRP solves vanish beyond 70 orders whereas the lines for the below column generation approach do not. This illustrates another aforementioned advantage column generation approaches have over direct solves

With both of these points in mind, we now have an understanding of why column generation is often applied to VRP solution methods. Let's next dive into the specifics of how this happens, beginning with some background for both the VRP and column generation to make sure we're all on the same page.

VRP Preliminaries

For this article, I'm going to focus on the VRP with Time Windows (VRPTW), which can be formulated as follows.



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

At this point, there are two major performance issues we can take into account when solving the VRP.

First, if your variation does not have time windows, then it is possible that it has subtour elimination constraints. Adding all subtour elimination constraints to your model explicitly (read: all at once) can bottleneck a solve as they are exponential in size. To fix this, you can add subtour elimination constraints implicitly (read: as needed), and Gurobi has a nice tutorial on how to do so with their solver.

> *As a side note, I chose to work with the VRPTW for this tutorial as the implementation is a little neater than other VRP formulations owing to the fact that you don't need to implicitly add any constraints. If your formulation is not the VRPTW, the following approaches to improve solve performance will likely work for you, too, but you'll want to account for the differences when generating routes. If you're not sure what I mean by "generating routes", don't worry. Read on!*

Second, even with a polynomial number of variables and constraints, the VRPTW

we'll refer to going forward as the Set Covering Problem (SCP).

Again, we have two performance considerations with the above.

First, if your VRPTW formulation is very tightly constrained (i.e. few feasible routes exist), then the above SCP is significantly smaller. You can pass the SCP directly to your MIP solver and may very well find a quality solution in an acceptable amount of time.

original VRPTW formulation. However, the Restricted SCP (RSCP), defined below, can be significantly smaller and easier to solve than the VRPTW while still producing a good solution. The trick is to just make sure the subset of routes chosen is small and includes those that visit all customers at a low cost. We can achieve this through column generation, but before we discuss how, let's take a moment to review what exactly column generation is.

Column Generation Preliminaries

small subset of routes that visit all customers. Ideally, you'd use a heuristic to find a combination of covering routes that would be somewhat low cost in total, but for simplicity, you can start with the set of singleton routes (i.e. those that visit only one customer), which is what I do in the implementation. Next, optimize the LP relaxation of the RSCP and check if the other routes not in the problem can improve the solution. If so, we add some of them to the RSCP and re-optimize its LP relaxation. We continue this process until we can't find any more routes that improve the solution or we're happy with the number of routes we've added. But how can we tell if a route improves the solution for the RSCP?

We calculate its reduced cost! For that, consider the dual of the LP relaxation of the SCP.

Great! But, wait. I'm back where I was before with having a large problem to solve. Don't I have to calculate the reduced cost of possibly exponentially many columns to decide which to add to the RSCP? You could, but you could also solve an optimization problem to *generate* them. For reference this problem is often called the Pricing Problem for how it gets reduced costs for columns.

Everything up to this point in our conversation about column generation is in general relevant to any set covering formulation. However, the Pricing Problem differs based on application. Let's zoom in on the case for the VRPTW.

Column Generation Specific to Vehicle Routing

At this point, we are ready to formulate a MIP model to generate routes to add to the RSCP. Since we desire columns that will reduce the RSCP's objective, we want to minimize the reduced cost of the route we generate. We then need to constrain this objective to that of feasible routes, which we have in constraints 1 and constraints 3–6 in the VRPTW. Below, we put our objective and constraints into the same terms and proceed to define the Pricing Problem.

With this formulation for the Pricing Problem, the set of solutions uncovered by our MIP solver can quickly yield a set of columns improving the RSCP. Iterating between resolving the LP relaxation of the RSCP and adding columns yielded from resolving of the Pricing Problem, we now have a column generation algorithm that will guide us towards a creating a set of routes that includes a relatively low-cost subset visiting all customers. Just make sure that you update the column reduced costs in the Pricing Problem after each resolve of the RSCP LP relaxation.

At this point, you might be asking why I chose a MIP approach when dynamic programming (DP) is usually employed to solve the Pricing Problem. In practice, I've seen clients who need a diverse set of constraints added to the pricing problem, necessitating an algorithm that has richer modeling capabilities. Additionally, solving as a MIP allows for many routes to be generated from a single Pricing Problem instance given that MIP solvers find many solutions en route to terminating. We can further increase the number of routes generated between each optimization of the RSCP LP relaxation by solving the Pricing Problem more than once (as long as we add constraints to not produce previously generated routes).

Seeking Integrality (and a Solution)

Now we have a way to find a "good" subset of routes that visit all our customers. All that remains is to decide which of them to choose. Given the relatively small size of our RSCP, we can do this rather easily by passing it to our MIP solver and solving it directly. We present the chosen routes as our solution!

Extra credit: The above tutorial finds a feasible approximate to the optimal solution(s). In the case that an optimal solution is needed, a method known as Branch-and-Price can be employed to retain some of the performance improvement we unlocked above. It differs from this tutorial in that after column generation, if


the curious reader to check out "A tutorial on column generation and branch-and-price for vehicle routing problems" by Dominique Feillet, which has details on how to do so.

Implementation Details

Alright. You've patiently waited as I've babbled on for several minutes now. Or, maybe you just scrolled right to the code like this is StackOverflow or something. Either way, below is my implementation of the above explanation. The methods HeuristicVRP.solve (line 298) and HeuristicVRP._add_best_routes (line 342) will probably be of most interest to you given they cover the most nontrivial parts of column generation. For details on parameter tuning (lines 86–95) or running on your own machine, check out this article's corresponding GitHub repository here. (If my code is of help to you, then you might benefit from reading into parameter tuning as I put in rather naive values for the sake of time.)

I hope this helps and feel free to leave a comment below!


Acknowledgements

Thank you to AIMMS for the VRP solution image and VRPTW formulation.