Finding the Column Generation Error

Something is wrong with the algorithm. In principle, column generation is supposed to proceed by generating the single best route from the subproblem and inserting it into the list of potential routes to be processed by the RMP.

Yet when I run the algorithm, prematurely, the z-value of the last-added route is 0, and the entire program stops. (The program stops because colgen means the next generated column must be selected for the algorithm to proceed.) Why do I say prematurely? Because **my colgen stops not because the reduced cost of the newest route is nonnegative, but because the reduced cost IS deeply negative, yet the RMP gives it a z-value of zero!** In other words, **my colgen stops because, despite the last route having a negative reduced cost, it is not accepted by the RMP**.

For example, with n_jobs = 25, and setting the colgen loop to only accept the single (1) best route from the subproblem, we terminate after 6 new routes were generated (and the first 5 accepted), because the z-value of the last route is 0.0 (Fig. 2). Infuriatingly, this leaves us stranded above the true optimum of 2'845 at 4'022 (Fig. 1).

```
Solved in 23 iterations and 0.01 seconds (0.00 work units)
Optimal objective  4.022684267e+03

User-callback calls 40, time in user-callback 0.00 sec
We don't use any of our new routes. Column generation is ended.
```

Figure 1: The final Gurobi output before the algorithm is stopped because the z-value of the newest route was zero.

```
[0.0, 0.0, 0.5, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.5, 0.0, 0.5, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 1.0, 1.0, 0.5, 0.5, 1.0, 1.0, 1.0, 1.0, 0.5, 0.5, 0.5]

[0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0]

[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0]

[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0]
```

Figure 2: z-values as the column generation algorithm proceeds with 1 new route added each time. Notice the zero at the last element of the last list of z-values, causing the colgen to stop. Also notice the 0.5-values three routes before termination.

Yet increase the number of routes accepted per colgen loop subproblem to 2, and the objective decreases. The reason I added in the *n*-route colgen was because I hypothesized that the machine had reached some roadblock, but we didn't deserve to

fully stop because I don't believe the machine actually found all the routes that would reduce the cost. Therefore, to circumvent this issue, I allow the machine to take on the best 2 routes instead of the single best. In fact, if we apply this argument forward we can make use of "*n*-route column generation", in which the best *n* routes are ingested rather than the best 1. As can be seen in Fig. 3, this turned out to be true. The machine indeed gave a better objective of 3'900. Better, but far from acceptable.

```
Solved in 62 iterations and 0.01 seconds (0.00 work units)
Optimal objective  3.900256501e+03

User-callback calls 79, time in user-callback 0.00 sec
We don't use any of our new routes. Column generation is ended.
```

Figure 3: With 2-route colgen, the final objective is 3'900.

Examining the z-output shows the merit of this algorithm. In particular, there exist a few routes for which the last z-value (corresponding to the second new route generated) is nonzero (in this case, by inspection, either 0.5 or 1.0), but the penultimate z-value (corresponding to the first new route generated) is zero. This is chilling, because it means *n*-route colgen works, so there's got to be a mistake somewhere in the colgen. Theoretically, we shouldn't have to resort to *n*-route colgen as a "patch" for colgen.

Unbelievably, it is not until *n* equals *n_jobs* itself (25) that we get the exact objective minimum of 2'845 with the exact correct routes, though approximate routes do exist once *n* hist 15.

So our goal now is to find the mistake buried somewhere in the colgen.

Ahh, after 10 minutes I finally found what I shouldn't have done.

```
if sum(last(z_values)[end - num_routes_at_a_time + 1 : end]) < 1e-2
    # we don't use any of the routes we just generated
    println("We don't use any of our new routes. Column generation is ended.")
    keep_going = false
    break
end
```

Figure 4: The offending piece of code: If the last *n* z-values are all zero (*n* is the number of routes we've chosen to generate), stop the algorithm immediately.

I wrote a piece of code which would check if column generation used any of the last *n* routes, and if not, we would stop—because nothing better was to be found.

This is not true. It is not the case that, if the last *n* z-values are 0, we should stop the whole thing. The only reason colgen ends is because the new route has nonnegative

reduced cost! And it is true that for many iterations, there are strings of 0's for the z-values of the newest generated route. But that doesn't mean nothing is happening under the surface. In fact, even when a string of 0's happens, and even when the objective appears to stand still, the pi-values are still silently churning, with a few modifications each time! See Fig. 5 below for an example.

```
[114.5, 132.4, 0.0, 512.5, 15.3, 14.0, 45.7, 180.0, 115.7, 31.4, 398.4, 38.5, 18.7, 19.1, 33.7, 16.4, 404.1, 18.8,
34.1, 145.7, 27.5, 11.7, 46.6, 276.1, 57.7]

[114.5, 76.4, 0.0, 531.2, 15.3, 14.0, 27.0, 180.0, 115.7, 31.4, 438.4, 17.1, 0.0, 0.0, 15.0, 16.4, 460.1, 0.2, 15.
4, 145.7, 27.5, 11.7, 46.6, 276.1, 57.7]

[114.5, 89.6, 0.0, 531.2, 15.3, 14.0, 27.0, 180.0, 115.7, 31.4, 438.4, 17.1, 0.0, 0.0, 15.0, 16.4, 446.9, 13.4, 15.
4, 145.7, 27.5, 11.7, 46.6, 276.1, 57.7]


[114.5, 205.1, 0.0, 456.6, 0.0, 29.3, 101.6, 180.0, 115.7, 31.4, 342.5, 53.8, 0.0, 0.0, 74.3, 16.4, 331.3, 86.3, 2
0.7, 145.7, 27.5, 11.7, 46.6, 276.1, 57.7]

[114.5, 171.4, 0.0, 492.2, 17.0, 15.7, 62.6, 181.7, 113.9, 31.4, 378.1, 40.2, 1.8, 0.0, 52.3, 16.4, 365.1, 57.8, 1
5.4, 145.7, 27.5, 11.7, 46.6, 276.1, 57.7]

[114.5, 170.1, 0.0, 492.2, 17.0, 15.7, 62.6, 180.5, 115.1, 31.4, 378.1, 40.2, 0.5, 0.0, 52.3, 16.4, 366.3, 56.6, 1
5.4, 145.7, 27.5, 11.7, 46.6, 276.1, 57.7]
```

Figure 5: Pi-values at the end of column generation. Observe how no two lists are exactly the same: there are always modifications taking place, even if new routes are not accepted!

The reason my colgen stopped, as I bolded on the first page, is because **I made it stop when there was actually no conceptual reason to stop it**.