Checking Xi Variables

We have determined that the two bottlenecks for the mcg TH are: a major bottleneck at solving the RMP; and a minor bottleneck at clipping coverage windows.

The time burn of the RMP is determined by two factors: how often we call it and the speed at which the RMP is solved, which depends on how many routes are already in the system. When doing the jobs-only problem, I proposed "*n*-column generation", i.e. throwing the best *n* routes into the set $\mathcal{Q}_0$ instead of just the single best one on every SP iteration. This used the assumption that the next routes to be chosen were likely to be the 2nd-best (or something similarly good) route from this iteration. It didn't work: *n*-column generation, which could in theory do as well as decreasing *k* iterations on the RMP to *k/n*, didn't reduce the number of RMP iterations by much. Even worse, we are adding *n* routes on every turn, so whereas in the normal case we would do RMP with *q*, *q*+1, *q*+2... routes, now we do it with *q*, *q*+*n*, *q*+2*n*, ... routes!

There is not much we can do to optimize the RMP beyond its natural best, because this is a Gurobi black box. (Maybe we try HiGHS? But it's unlikely to change anything, just a n_jobs = 37 answer.) But the purpose of this document is to see that we *are* actually hitting its natural best. Specifically, I will check that we don't have ξ_{it} problems that require intervention like stabilization.

What do we check? There are three main ways to do this, which I perform:

1. Reduced cost analysis
2. Frequency analysis of new routes generated:
    1. by nodes only (e.g. [0, 5, 26]);
    2. by nodes and times.
3. Analysis of xi-values / treasure locations:
    1. by (i, t) nodes only;
    2. by (i, t, rc).

We want reduced cost to be generally decreasing.

In frequency analysis, we check that neither of these two phenomena occur: insistence on pushing the same exact route AAAAA...; and cycles like ABCABC...

In analysis of xi-values, we want a small proportion of nuggets to have the same identity over time with the same xi-values. For instance, if *i*=1 *t*=5 xi=102 is present at one iteration, then here are our cases, from best to worst:

- (1, 5) doesn't appear at all afterwards

- (1, 5, not-102) appears later
- (1, 5, 102) appears a few more times

Methodology
We run all algorithms on mcg TH from 10 to 50.

Reduced cost: Every time we get a route with a certain best reduced cost, push that reduced cost into a waiting array.

Frequency analysis is conducted by first extracting all the routes, then performing analysis on them. We wait until the algorithm finishes, then extract the m_cov+1 to end indices of cov_routes. Then these routes are passed into functions to obtain their details.

Xi-values are pushed into the array every time. Analysis functions will be run later.

Helper algorithms are as follows:

Reduced cost:
None. Just plot.

Frequency analysis:
1. A function dict_nodes that obtains the frequency of routes based on nodes;
2. A function dict_nodes_times that gets frequency of routes based on nodes and times.
We plot a cdf on 1 to see what proportion of routes are repeated. Same for 1. Then we do manual inspection of the routes to see how different times are.

Xi-values:
Do the same cumulative-plot analysis as before: Frequencies of (i, t) only, and then frequencies of (i, t, rc). This will tell us how much "sag" there is in the cumulative distribution, i.e. how much of the effort is burned by a few annoying xis. The more the sag from y = x, the worse it is.

There is another parameter which helps us assess whether our routes are being added well: what proportion and number of the previous xi's were changed in the current iteration. We can distinguish between whether the previous xi's were merely rc-changed, or removed altogether. We plot them separately because we are going to have lots of different n-values. Also, proportion and number form separate graphs as well.

Example: Suppose we have 8 xi's at one iteration. 3 of them are not there in the next iteration, and 1 has its rc changed. Then 0.375 is in one plot, and 0.125 in the other.

Note that this method does not really consider whether a nugget disappears, only to reappear later, but we do not care. Our goal is to test the efficiency of our routes.

Results

We do the following analysis for 10 to 50 jobs inclusive.

Reduced Costs

I tested this parameter to see what the curves for jobs and covs looks like. It isn't actually a graceful curve like the objective value versus iterations; there are lots of spikes. Since the plot is too cluttered with all 41 cases, here's an example with 41/4 of them. Notice the spikes. This means we have a healthy dual-primal relationship. A steady curve would also be healthy; the only unhealthy thing is if we have repeats or a horizontal line for quite some time before finally breaking towards zero from the negative direction. *Figure 1* displays results for job and coverage reduced costs. Notice the differences: Job RCs tend much closer to 0, because heuristic optimization was called upon as the first stage of mcg!

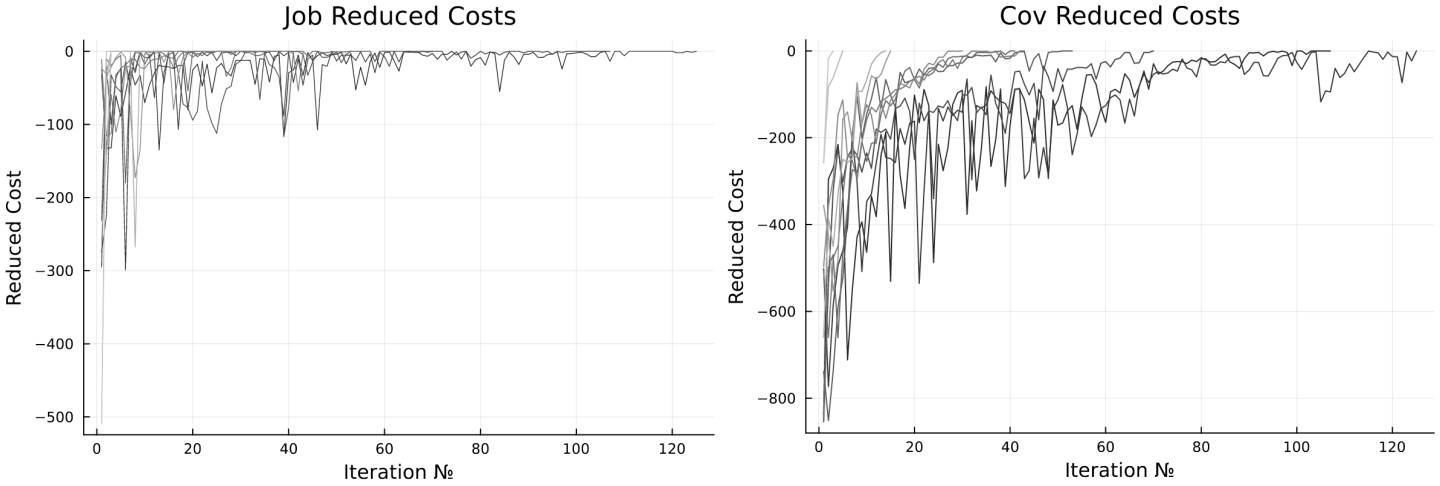


Figure 1: Job and coverage reduced costs, mapped over iterations. Different line segments mean different number of jobs.

Coverage Routes

As the number of jobs increases, the tendency for certain routes to be generated several times increases as well. This can be seen in the slight but noticeable "hammock" shape in the cumulative graph at *Figure 2*. Here, each color (a shade of gray; darker = higher number of jobs) represents a cumulative curve; the perfect curve in which all routes are generated exactly once is the blue dash. As the number of jobs increases, the hammock sags a bit more. You will also notice that for almost all cases, until we hit ~0.7, we have just straight line segments; this indicates about 70% of all our distinct routes are only used once.

For small cases, there is no noticeable Pareto-like 80-20 distribution; near 50 jobs, 75% of distinct routes are responsible for 50% of the total routes generated.

Also, observe that the slope of ascent towards (1, 1) is not particularly steep. This means the route most often repeated isn't so intensely popular.

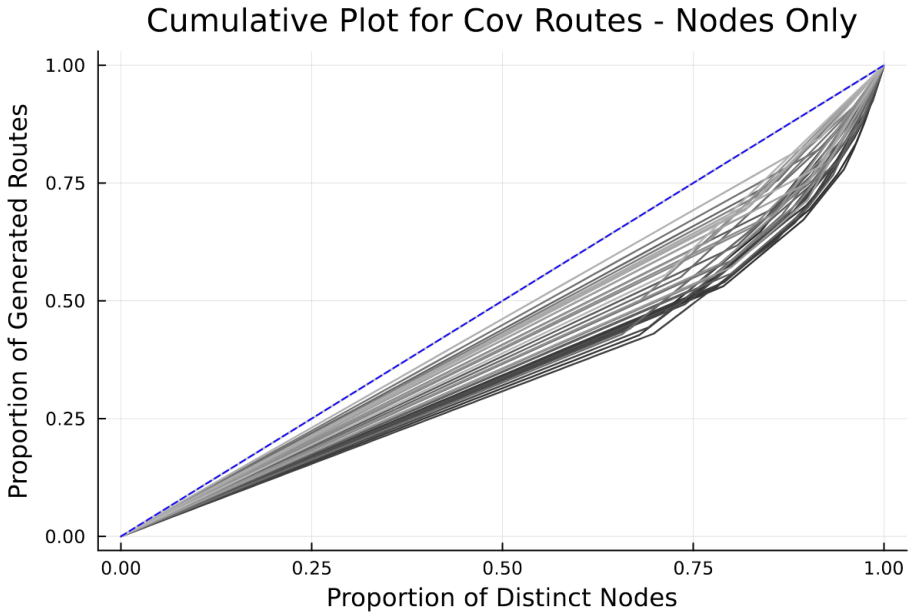


Figure 2: A cumulative plot for coverage routes, based only on their nodes. Each test case gets a line. The closer the hammock is to the dashed blue line (1:1 correspondence: no route by its node is added more than once), the higher the efficiency. Observe that the majority of routes by nodes are used once.

We then examine the next graph, *Figure 3*, a cumulative distribution on nodes and times. In other words, we test how many times the same route node- AND time-wise is added. As can be seen, the blue dash matches with all 41 test cases perfectly. This means there is never a loop or cycle in our generated routes!

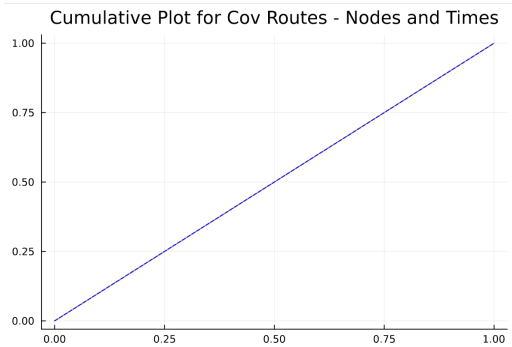


Figure 3: For all test cases, all routes added are unique.

Closer examination, perhaps manually, of routes that *do* have the same nodes shows that their times are all different. There are some tweaks on a few of the spots. This is

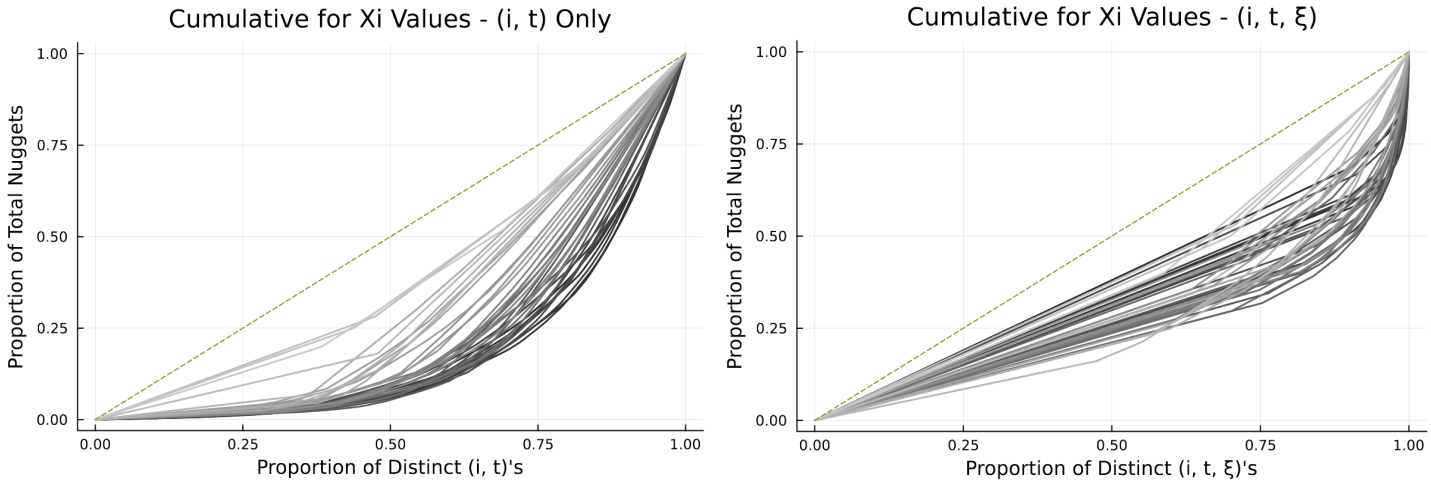
exactly the behavior we want, and the intent of coverage. Either we need to change the time to accommodate better job work windows, or we found a better coverage path with a better coverage time.

Xi Values

We examine nuggets first the same way we did routes. The sag on the hammock is bound to be more severe, and this has more of a "Let's see how good/bad things are" feel than a "Here's formal proof that we don't waste iterations on useless routes".

The sag gets worse as the number of jobs increases, and there are a lot of distinct nuggets. There is a Pareto: at 50 jobs, 75% of the distinct (i, t)'s are responsible for 25% of all nuggets. Based on *Figure 4a*, many (i, t)'s are repeated (as opposed to a few ones).

A neighbor indicator of how bad the repetitions are is to examine not the same (i, t), but the same (i, t, reduced_cost). This forms *Figure 4b*, which looks a bit like the cumulative plot for coverage routes on nodes only.

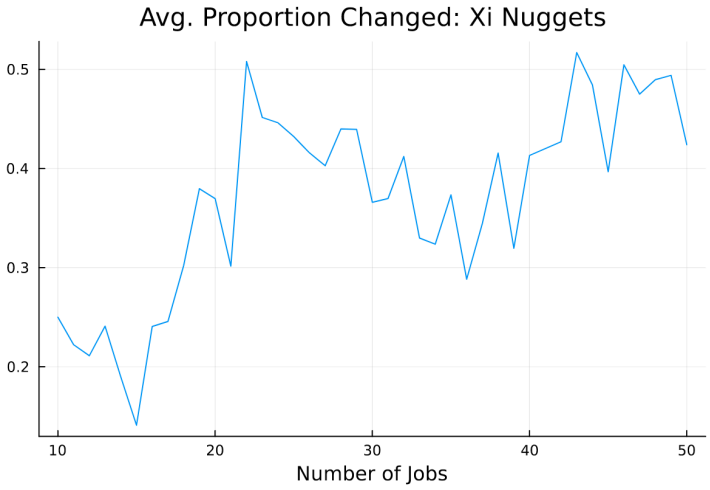


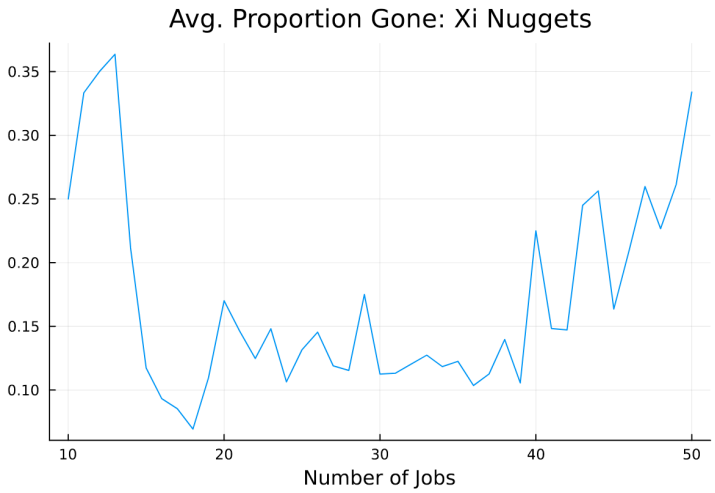
Figures 4a (left) and 4b (right): Cumulative plots for xi nuggets, based on time-space node only and time-space node with reduced cost, respectively.

We cannot expect that all nuggets can be changed or resolved during one iteration, but the final test of dual-primal functionality is to see what proportion of nuggets are changed or gone from one iteration to the next. The average proportion of nuggets changed versus number of jobs is plotted at *Figure 5a*, around 40% for 20-50 jobs. As for nuggets gone in *Figure 5b*, the proportion is between 10% to 35%.

As the number of jobs increases, the proportion of unchanged jobs slowly goes down; for 10-40 jobs, it is about 50%: see *Figure 6*. This is probably expected: we add only one coverage route and one job route, and there's no way all of the nuggets can be

resolved. That roughly half of the nuggets are changed indicates good routes are being added. This is an average, however, so we also plot the proportion on each iteration.





Figures 5a (left) and 5b (right): The proportion of nuggets that are modified (the (i, t) still there, but the RC changed) and disappeared (the (i, t) goes from a positive treasure to zero, indicating good teamwork between dual and coverage route: the cov. route added in response to the nugget's being there from the previous dual alleviated the constraint equality).

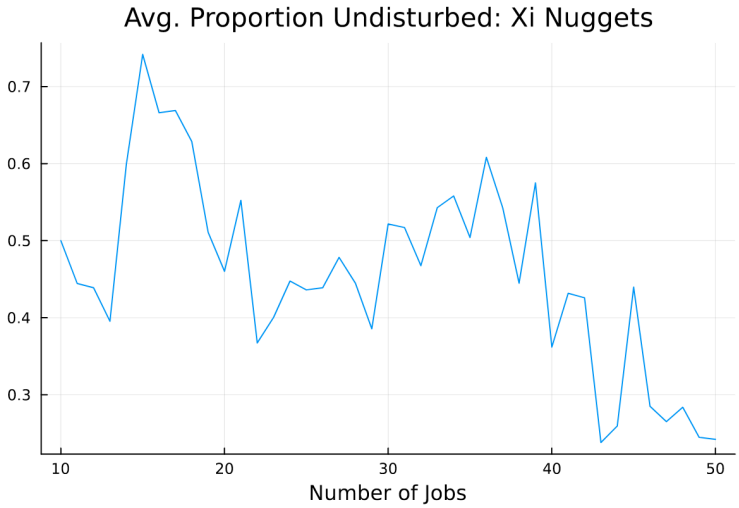
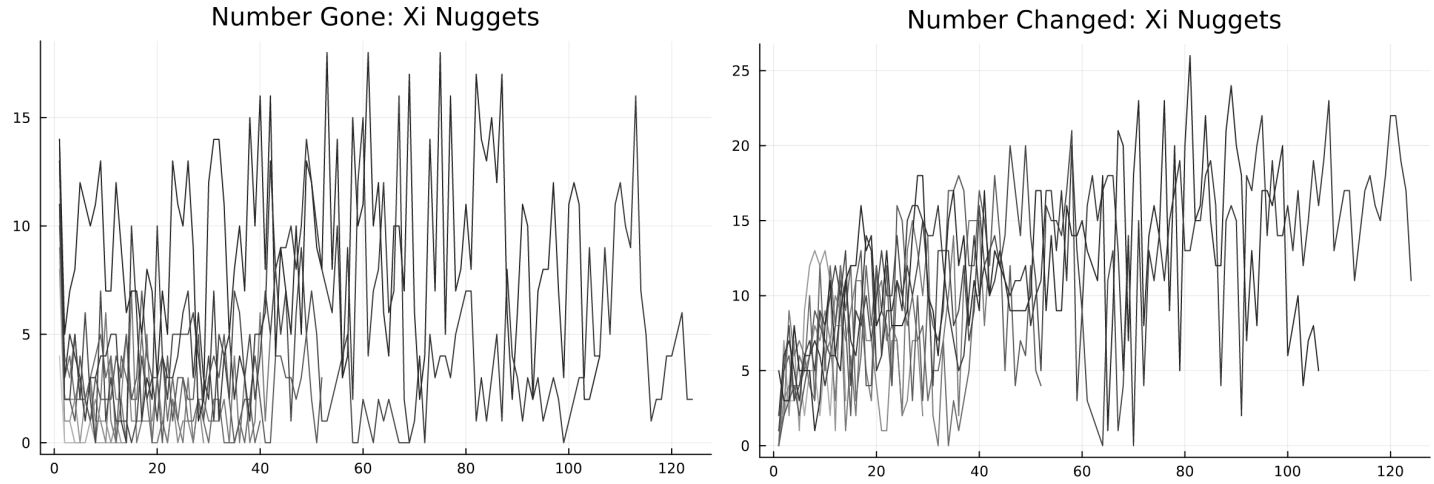


Figure 6: The proportion of nuggets left undisturbed: still there after the previous iteration and no change in the reduced cost. This represents nuggets which the best coverage route did not reach, or had too little value to be explored based on the cost of traveling to that spot.

The plots in *Figure 7* display individual iterations' nuggets gone and changed. Notice that this number is almost always higher than 0 (in fact, always higher if we consider gone+changed), indicating the new coverage path added is responsive to the dual, and there is no "slippage" between the primal and dual (if there were, we might see cycles of routes, or the same route proposed repeatedly).



Figures 7a (left) and 7b (right): On each iteration, how many nuggets are resolved (gone) or have their RC changed. Several test cases are included on the plot.

Conclusion

By all metrics—reduced cost, coverage routes, and xi nuggets—and their means of analysis, we conclude that there is no trouble in coverage route production and that the route generation and dual variables are responsive to each other. There are no cycles or repeats in generating routes, and the process is not inefficient. Therefore, stabilization appears unnecessary.