Vehicle Number Testing

Now that our LSA is up-to-speed, we will run some tests on different sizes of job numbers. This will be conducted through a large loop which will iterate through different values of n_jobs. Then the time values will be collected in a matrix.

Recall that the function for extracting time in seconds is datetime2unix(now()).

Take all our contents from the entire without-ndc python notebook and shove it into one for loop. Then perform the following. For each iteration:

1. **Start Overall Timer (time_overall_start)**
2. **Start time for parameter setup (time_parameter_start)**
    1. [1.1] Parameters, [1.2] Locations, Windows, Loads, [2] Helper Methods
        1. None of this should take too much time because it is just initialization...any computation comes from setting up the random locations
3. **End time for parameter setup (time_parameter_end)**
4. **Start time for column generation loop (time_loop_start)**
    1. Go through the modelcg = Model... and the definitions that lie outside the loop
        1. I'll consider this part of the loop despite not being in the while loop, because it's practically a loop thing and happens once and the ratio of time used is consistent across n_jobs numbers.
    2. Go through the full loop
5. **End time for column generation loop (time_loop_end)**
6. **Start time for post-loop processing (time_post_start)**
    1. Do all the procedures to extract routes etc., whatever is there
7. **End time for post-loop processing (time_post_end)**
8. **End Overall Timer (time_overall_end)**

There are 4 different sets of times to test: Overall, parameter setup, colgen loop, post-loop processing. These should all be stored into an array.

Put these into a graph and we know how well the algorithm performs with different numbers of jobs.

Note: To accommodate super-small groups, I've tweaked the create_cluster_sizes function in *[1.2] Location, Windows, Loads* at the num_to_add variable, so that the minimum supported cluster drops below size 3. Specifically, when n_jobs is 13 or less, there will throw an error complaining that we have an empty random range. When n_jobs is 14 or higher, this is avoided because n_jobs is large enough.

The code I modified is (this below is the initial code):

```
num_to_add =
rand(
min(3, jobs_total - jobs_created) :
min(Int((n_jobs/5)÷1), jobs_total - jobs_created))
```

I also made an adjustable parameter for the number of target clusters.

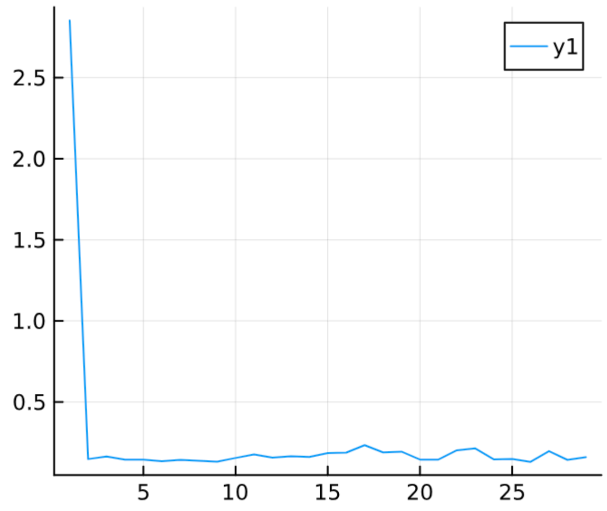
Now we can go down to as low as 10 jobs.

Results

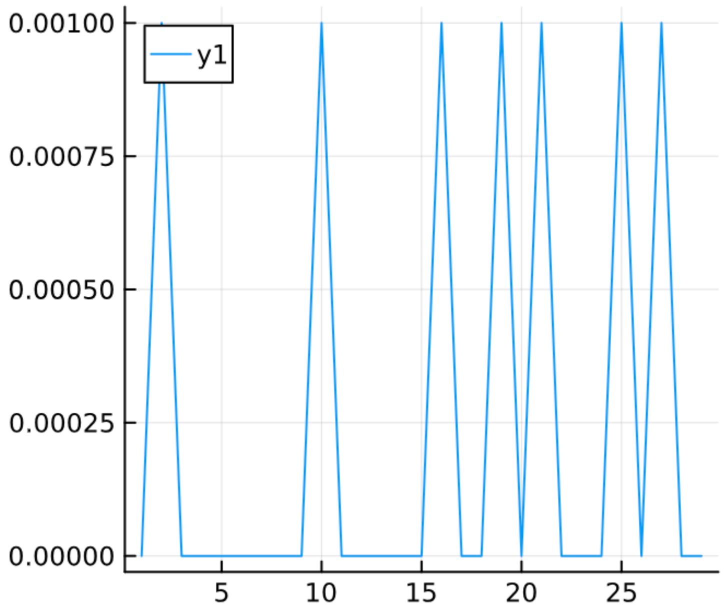
Random seed 1234: We are unable to break 38, after which the time goes too long to be measured. Before 39, the last time recorded for the time loop was 36.449 seconds.

For all of these experiments, the 10 jobs case had a large spike that resulted in a total time of 4 seconds, well above the next scenarios, which calmed down. In fact, 4 seconds was not hit until we performed 31 jobs, at which time the clock hit 6.023 seconds.

time_parameter: It was stable around 0.15 seconds. No point in measuring this again.



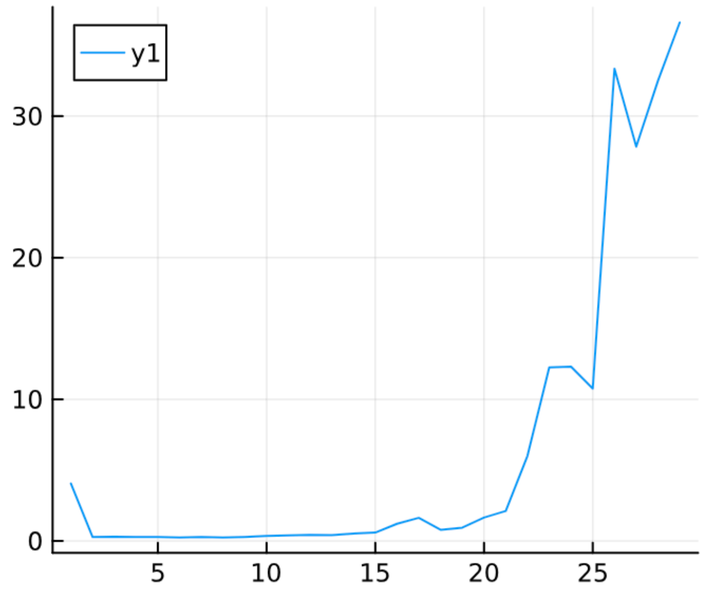
time_post: Sometimes 0, sometimes 0.001 seconds. Also no point in measuring this again. This and time_parameter are O(1).



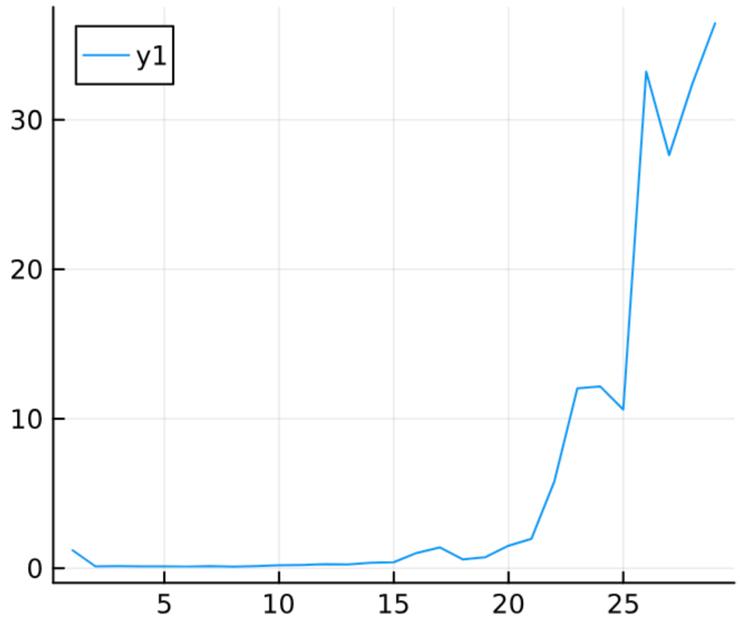
Might this be evidence that Julia's limit in accurately measuring times has a graininess of 1 millisecond? It can't possibly be that these operations take exactly 0 seconds or exactly 0.001 seconds, right?

time_overall: Here's the clearest indication of what's happening. We know the bottleneck of this time trial is obviously the loop, so from now on, we'll only measure the loop.

It is probably exponential, and experiments with more random seeds will clear this up.



The time for the loop (the only thing we'll measure going forward) looks very similar.



Note: Index 1 refers to n_jobs = 10, so index 29 (our highest) refers to n_jobs = 38.