Speeding Up Each Loop

Our jobs-only column generation is rather fast, taking less than 10 minutes to solve 74 jobs and 1 minute to solve 50. But when we perform double column generation, the speed is dramatically higher. In this report I analyze each individual loop's time, and propose ways to reduce the time taken.

Loop Contents

Within our loop are the following components:

1. **Build a model's constraints, variables, objectives**
2. **Optimize the model**
3. **Extract variables for further processing**
4. Obtain clipped windows
    1. **For jobs**
    2. **For coverage**
5. Run the subproblem algorithm - Jobs / Coverage
    1. Enter while loop
        1. Check if we go to the next state (we know previously this is not the bottleneck)
        2. Enter for loop
            1. **Check time feasibility**
            2. **Check pruning**
            3. **Push new route**
        3. Exit for loop
    2. Exit while loop
6. Extract best route
    1. **For jobs**
    2. **For coverage**
7. **Update variables**

I will time-check all the stages in bold to see what happens.

n = 18
The model runs 29 times, with model-building taking on average 26 ms to build for a total time of 0.77 seconds. The pattern is steady.

The model's time to optimize steadily increases, from 2ms to 8ms and 9ms as the iterations stack.

Extracting dual variables takes 1ms.

Getting clipped windows for jobs (i.e. function clip_time_windows_by_cov, checking coverage is enough for work being done) takes 0.5ms, but its counterpart, getting clipped windows for coverage (function work_windows) is considerably more expensive, burning 6 or 7 milliseconds each time.

We'll discuss subproblem in a bit.

Extracting the best route is cheap, usually 0.5ms.

Updating variables takes an average of 4.5ms, not the bottleneck.

Therefore, the bottleneck has to be the subproblem.

SP Jobs: Time feasibility check 7ms. Check pruning 14ms. Push new route 2ms. These are totals! (There might be rounding due to milliseconds, we will account for that when we track overall time for running the SP.)

The SP Coverage is where everything starts to unravel. The sheer number of coverage routes available is what causes the problem to go long.

A total of 1'024'518 time feasibility checks were performed, burning a humiliating 21.106 seconds. Each check lasted on average 0.02ms.

442'057 times we checked pruning, taking a total of 9.714 seconds for an average check of 0.02ms.

44'543 routes were added, burning 33ms.

The real problem is the sheer number of routes that could be added in coverage. About 95% of the time is burned here.

Now what's fascinating is: Earlier, the algorithm ran much faster, and ever since I added in that little tolerance for the coverage, suddenly the time exploded. In fact, when I remove the 1e-8 tolerance in the branch pruning, the time taken plummets to 7.8 seconds total.

Now, only 96'744 time feasibility checks are performed in SP, averaging 2.118 seconds. Pruning is checked 83'174 times, totaling 2.296 seconds. And 4'205 routes were added, totaling 6ms.

Maybe the tolerance of 1e-8 is still too high? Rounding errors should occur later, perhaps I will use 1e-12 instead. After testing, we get the same issue.

This is because a lot of coverage reduced costs are zero, so it won't do us any good to allow that tolerance that puts so many routes back in.

As a result, it is worth sacrificing insignificant coverage routes. I declare we are not going to use any tolerance for the subproblem.

n = 22
As expected, the coverage SP burned most of the time, 12.045 seconds out of 18.758. The model runs 59 times, and each model build took an average of 42ms.

Optimization time has a steady, upwards linear trend, totaling 0.876 seconds.

Extracting dual variables takes 61ms.

Obtaining clipped windows for jobs, coverage takes 132ms, 967 ms, respectively.

The first check for coverage SP takes 4 seconds, for pruning, 8 seconds. (This is without the tolerance for error.)

Extraction of best route is longer for jobs than coverage.

n = 25
It took a minute. Most of the blame lies on the SP coverage for having so many routes, 400'874, to process.

At this point, it should be clear that what we need is a new method for solving the UROP question. If 25 jobs on dcg takes the same time as 50 jobs on jobs only, we should change how we approach the question. Indeed I have several ways. I'll close this document right now.