Removing y Variables: A Proof

In the CG (colgen) and PT (prune tree) methods I think the y_{it}^S and y_{it}^E variables are superfluous. Here is my proof.

The y variables are necessary only if the route structure, $\mathcal{Q}$, does not encapsulate work. But if we look at the code we have now, we can see that if we enter a node, we must have ample time to do the job at that node.

```
cur_time = last(Times[current_state])
dist_nec = travel_time[L[current_state], i]
old_job_time_nec = L[current_state] > 1 ? load[L[current_state]-1] : 0
new_job_time_nec = load[i-1]
new_window_start, new_window_close = windows[i-1]

if L[current_state] != 1
    if cur_time + dist_nec + old_job_time_nec + new_job_time_nec > new_window_close
        continue
    end
end
```

As can be seen, we are measuring that the summation of:
- Time right now at the START of entering the old node (because T, as interpreted in the old formulation, always refers to when we first enter a new node - in fact, T was only an integer until I decided that it was better to encapsulate the times as well),
- The amount of time necessary to do the old job,
- time required to get from the old node to the new node,
- amount of time to do the new job.

This is now measured against the closing of the new window. If the sum of the above is smaller than the close of the new window, then we go to that node.

Upon optimizing, if we select a route, then based on our uniqueness criterion ($\sum_q u_i^q z^q = 1$), we know that if any route is to have hit the particular job *i*, it must've been the unique route which passed by *i*. And because the optimizer printed out a solution, that unique route did the job at location *i*. Thus, the converse of "If we enter a node, then we have enough time to do the job", which is "If in a route we had enough time to do a job, then we will do the job on that route", is true.

In other words, routes with paths don't just encapsulate nodes—they implicitly tell us that whichever nodes are in that route, jobs are also being done at that route.

By the way the shortest paths algorithm was set up, if we arrive at a node, we are forced to stay at that node for at least as long as the job load due to this piece of code:

```
reach_new_node_time = max(cur_time + dist_nec + old_job_time_nec, new_window_start)
```

More specifically, suppose we were going to a new node. Then we get there at the max of: when we can earliest get there, or when the new window starts. Now, consider when

we get to the node after that. Taking the subtraction of the two yields that we must have taken at least as much time as the load of that new job (the first I mentioned) just sitting at the new node, doing the job.

Therefore, we can delete all constraints related to both sets of *y* variables without punishment. Specifically:

- Remove y_s and y_e from @variable declarations
- All seven constraints on y_s and y_e:
  - That y_s at a time <= y_s at the next time
  - That y_e at a time <= y_e at the next time
  - That before the window opens, y_s = 0
  - That after the window closes, y_e = 1
  - That the difference between y_s and y_e is at least the work load
  - That they are between 0 and 1 (used in place of coverage, which I haven't defined)
  - If a node is not being passed through on any route, y_s = y_e at that i and t

It also means we will no longer have a set of mu dual variables. So we delete the extraction of mu and its calculation in the SP LSA.