Removing the NDom Check

From the algorithm time analysis I just conducted (code here, writeup here), it is evident that the n-domination check burns 99.233% of the time and is the bottleneck of the shortest paths LSA, which in turn is the bottleneck of the whole colgen loop.

I propose to remove the n-domination check (ndc) for the following reasons.

#1 Without the ndc, our results are still mathematically correct. What does ndc do? It tells us that if two routes have the same start and end points, and go through the same exact nodes, possibly in a different order, and one is costlier than the other, then the costlier one deserves to be overridden.

Specifically, if a route we just generated is costlier, then we write over it in our next iteration of the loop that yields a route. (We don't remove this new route that we just checked because that is time-inefficient.) And if the route we just generated dominates over some older route, then we set the applicability of that old route to false, but keep going as if nothing else happened.

So what happens if we don't have the ndc? Not much. Wrt the previous paragraph, in the first case, we would leave costlier routes in our full route system without overwriting them. In the second case, we would not turn off the applicability of the old route, so it would still be viable. In both cases, once we look for the single best route to add post-subproblem LSA, the more costly route will be dropped from consideration or 'overruled' by a better route. So if we don't have the ndc, it won't impact whether our search for the single best route yields the correct answer or not.

#2 The probability of getting a dominated/costly route is sufficiently small. Note that we've generated 16576 routes. Not a single one of them triggered the second condition (some old route got dominated, and its applicability was set to false). Furthermore, not a single one of them turned out to be dominated by some older route (because otherwise, we wouldn't have gone through the 4th time check—modifying an existing place—exactly ZERO times). This is the empirical evidence.

Now let's look at the theory. In order for the ndc to have any effect on our output, there needs to be a new route which has both the same nodes in the route as an old route, as well as ending at the same node. In practice, this is quite rare. The most common case is likely to be when two routes start and end at the depot (as opposed to starting at the depot and both ending at the same job location), and happen to go through the same nodes, just one route is more efficient than the other. To share the same 'in-between' nodes is quite rare (which can be empirically tested).

One can imagine that, without the ndc, there is a buildup of such routes that would be flagged by the ndc. For instance, route A has the same configuration (same nodes and end node) as route B, then neither one gets its applicability turned to 'false', and then route C has the same config... But the ndc does not prevent this. If the applicability of B is turned to 'false', route C will fail the check against route A. There are two scenarios. First - an old route is falsified. Second - a new route is overwritten. Although the second

case will somewhat stop 'same configuration chains' from getting too long, it does not reduce time complexity because that costly route had to be created in the first place in order to be tested, and the time complexity of ndc is worse than the time complexity of pushing a variable. Each pushing takes about 0.001 second, while each check takes about 0.007 seconds. Furthermore, the former value stays constant as seen in the graph on time check #3 (pushing nodes), while it increases as seen in the graph on time check #5 (ndc).

What would happen if the probability of getting dominated was high? With or without the ndc, we would not be able to stop the creation/pushing of new inefficient routes. I suppose this is fine, the complexity of pushing isn't too bad. But checking them as the list grows longer accumulates apparently quadratically, which is bad.

#3 The ndc is slow. Yeah this is the main reason! We'd see our column generation loop speed up dramatically if we remove it.

What Happens Now?
We would:
1. delete the ndc entirely.
2. Add a statement "total_state += 1" within our loop (the "if ~(i in N[current_state])" one).
3. delete the "else" part of the statement if length(N) < total_state + 1. It will never happen because a route will never be flagged as costly; therefore, we will always increment total_state by 1.
4. Render the use of the A array for applicability gone. No longer will we check for this; it will be naturally filtered out with finding the best route.