>= vs ==: What's Going On?

One of our constraints in the jobs and coverage formulation is that there is exactly one vehicle in a route which visits each job. Technically this can also be greater than or equal to, because for our optimizer, >= is better practice.

$$\sum_q u_i^q z^q \ ? \ 1 \ \forall i$$

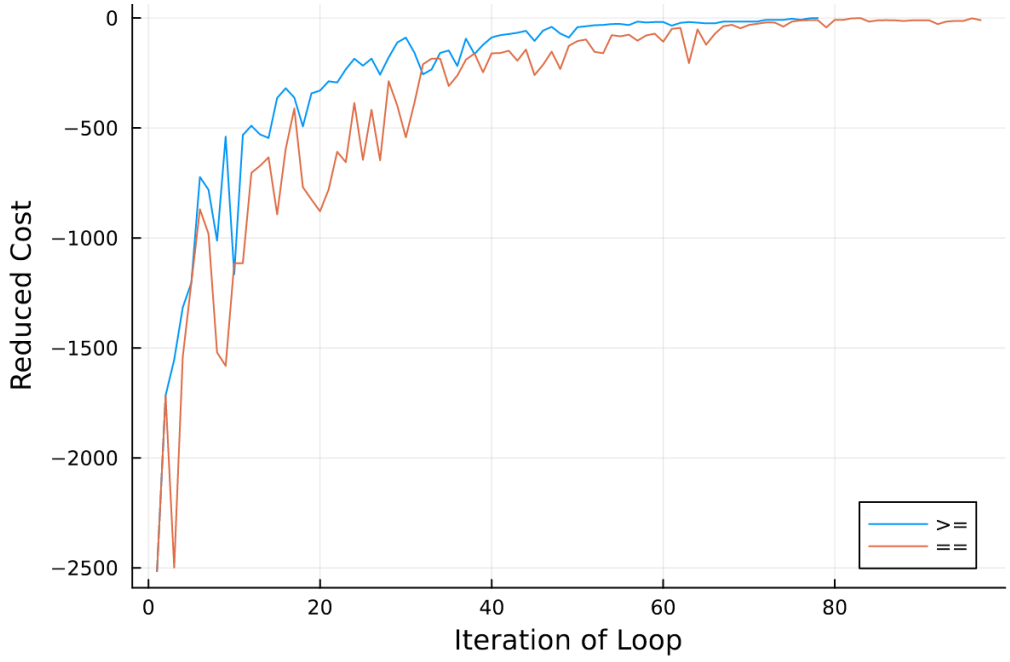
Interestingly, neither >= nor == has proven superior! When using PT NyVA, on n=25, >= gives a split-route objective of 2'853 while == gives a non-split wrong answer of 2'856. On n=28, >= is split with 2'942, while == is correct with 2'933. Mysteriously, with dcg on n=25, >= yields the correct answer of 2'845 + 1'931, while == gives a split answer of 2'853 + 1'931.

Here we inspect >= vs == on jobs-only colgen first, and figure out what's going on.

n = 28
This is the only case in jobs-only for which one of >= and == is correct, and one is wrong. >= splits the route; == doesn't.

Reduced costs of best route:
>=: -2'515, -1'716, -1'553, -1'315, -1'204...
==: -2'515, -1'716, -2'498, -1'540, -1'204...

If we graph these, we get the following plot.



As can be seen, the methods diverge at iteration 3 and never come together again (though coincidentally their 5th reduced cost is -1'204 both). The reduced cost for the == method is almost always more negative, suggesting that better routes are being found and that the loop will take longer to come to the top.

If we look at the dual variables, rho is always 0 (because the constraint isn't saturated) while pi is the same for the first two iterations. But then pi turns different as well!

This can be explained by the different choices of routes. After adding the second new route (this is the 3rd z-value), the differential is: For >= we accept the 3rd route while == rejects, so >= doesn't use dummy routes 1, 5, 6, 11, while == does all 4.

Meanwhile, the 30th route (2nd added one) reads:
0 → 1 → 11 → 5 → 6 → 7 → 29.

Both >= and == accepted the first new route, which reads:
16 → 17 → 18 → 19 → 13 → 14 → 7

Therefore, >= accepts the second new route by jettisoning 1, 11, 5, and 6 (7 was already kicked out at the first new route). This is allowed because 7, which is now covered fully by 2 routes, is protected under the >=.

Meanwhile, == cannot accept the second new route because it would have to deal with the too-high 7 value. And if it tried any funny business, like doing 0.5 on the two new generated routes, the cost would be higher than that of not accepting the second new route (because you'd have to accept back 0.5 on 16, 17, 18, 19, 13, 14). As a result, == passes on the second route.

This pattern continues and because the routes explored here diverge, the two methods never produce the same reduced cost again. Even if the route is the same, the reduced cost will be different.

But why does >= settle at 2'942? Shouldn't >= be lax enough to avoid the issues with ==, and do at least as good as 2'933?

If we look at the true answers, one of the routes that has z-value 1 in == but 0.5 in >= is 11, 12, 15. Another route with z = 1 in == is 16, 17, 18, 19, 13, 14. With >= there is a four-way split of 0.5 between 11, 12, 15; 16, 17, 18, 19, 13, 15; 11, 12, 13, 14; and 16, 17, 18, 19, 14.

Inspecting all the routes that were generated through >=, we came painfully close to the true real route of 16, 17, 18, 19, 13, 14. Had we done this exact route, the algorithm would've been over. Such routes include:

16, 4, 18, 19, 13, 14
16, 17, 18, 13, 14
16, 17, 18, 19, 13, 15
16, 17, 18, 19, 13, 14, 15
16, 17, 18, 19, 14

This is evidence that maybe once the best route was taken and chosen, certain other good routes were missed. This still doesn't answer the question of WHY in theory this would actually happen. So if we actually remember the best two routes (instead of 1), but proceed with the algorithm as usual, let's see what we end up with.

Heartbreakingly, the exact route we were looking for—16, 17, 18, 19, 13, 14—showed up as the second-best route on two occasions.

Three weeks ago I coded an implementation that allowed column generation on the *n* best routes instead of just adding the single best route every time. If we do this with 2 and test >= again, let's see what we get.

It doesn't work. We hover around this route with routes like
16, 17, 18, 19, 13, 15
16, 17, 18, 19, 15
16, 17, 18, 13, 14

The reason just adding a new route doesn't work is because our column generation scenarios are now different because our z-values with adding 2 routes every time are not guaranteed to be in line with those of adding 1 route every time. There is no guarantee that the second-best routes found through normal column generation will all, or at all, show up with 2-column generation.

At this point, there has to be a way to force this route to show up. Conceptually, we need to force close but good routes to show up in the dual.

This is the same issue that happens that explains why dcg n=25 >= yields the correct answer, while dcg n=25 == goes wrong with partitioning.

Eventually, if we get to our split-route formula, adding in 16, 17, 18, 19, 13, 14 should be a favorable reduced cost! So let's see, at each iteration, where this route is.

```
Path [1, 17, 18, 19, 20, 14, 15, 30] has reduced cost -2130.7335549431496
Path [1, 17, 18, 19, 20, 14, 15, 30] has reduced cost -86.5857849517138
```

The route only shows up as valid twice. What if we removed the pruning?

Well, we do get the right answer of 2'933, but now we have a lot more options for the path's incorporation. In particular, look at the bottom.

```
Path [1, 17, 18, 19, 20, 14, 15, 30] has reduced cost -5.9649343509788935
Path [1, 17, 18, 19, 20, 14, 15, 30] has reduced cost -21.942063365771475
Path [1, 17, 18, 19, 20, 14, 15, 30] has reduced cost -8.050424734014797
Path [1, 17, 18, 19, 20, 14, 15, 30] has reduced cost -8.050424734014797
Path [1, 17, 18, 19, 20, 14, 15, 30] has reduced cost -8.050424734014797
Path [1, 17, 18, 19, 20, 14, 15, 30] has reduced cost -2.842170943040401e-14
Path [1, 17, 18, 19, 20, 14, 15, 30] has reduced cost -2.842170943040401e-14
Path [1, 17, 18, 19, 20, 14, 15, 30] has reduced cost -2.842170943040401e-14
```

Is it possible that we need a tolerance factor for the pruning, and everything will be peachy? It looks like we might have one of those "1 is not >= 1.00000...02" situations.

So let's try: proposed_reduced_cost < NRC[i] + 1e-5.

It works! Now our objective is down to 2'933 as intended! And indeed, the difference between the optimal solution 2'933 and the split-route solution 2'942 was 8.050..., the reduced cost we saw above and down below here.

```
Path [1, 17, 18, 19, 20, 14, 15, 30] has reduced cost -2130.7335549431496
Path [1, 17, 18, 19, 20, 14, 15, 30] has reduced cost -86.5857849517138
Path [1, 17, 18, 19, 20, 14, 15, 30] has reduced cost -8.050424734014797
Path [1, 17, 18, 19, 20, 14, 15, 30] has reduced cost -8.050424734014797
Path [1, 17, 18, 19, 20, 14, 15, 30] has reduced cost -8.050424734014797
```

If we try this on any case, we should be able to get better results. In particular, now we will test n = 25, 26, 28, and 37, the four "problem cases".

And indeed, n = 25, 26, and 28 resolve themselves satisfactorily. The >= and == give the same exact answers, as they should in all cases.

What about n = 37? Recall that our previous results were:

PT >= y: 3'756, 20.42 seconds, split
PT == y: 3'773, 27.36 seconds, split

PT >= NyVA: 3'765, 1.55 seconds
PT == NyVA: 3'761, 2.01 seconds, split

Today, we have the same with PT >= NyVA: objective 3'765. The routes are integral:

14	15	16	17	18	19	20	
5	6	7	8	9			
25	26	27					
21	22	23	24	28			
36	**10**	**11**	**12**	**13**			
1	**2**	**3**	**4**				
37	**29**	**30**	**31**	**32**	**33**	**34**	**35**

But PT == NyVA improves from 3'761 to 3'757. The routes are very slightly different:

14	15	16	17	18	19	20
5	6	7	8	9		
25	26	27				
21	22	23	24	28		
10	**11**	**12**	**13**			
1	**2**	**3**	**4**	**32**	**33**	**34**
36	**37**	**29**	**30**	**31**	**35**	

Ok, so this is the same pattern of small tweaks that we've seen before. This is also the reason why we only miss the objective by 8 or so. It isn't too big of a deal.

But what if we were also to apply the same logic for the 1 constraint and tone it down to 1 - 1e-5? We improve to 3'758 with the following routes:

14	15	16	17	18	19	20
5	6	7	8	9		
25	26	27				
21	22	23	24	28		
10	**11**	**12**	**13**			
1	**2**	**3**	**4**	**33**	**34**	**35**
36	**37**	**29**	**30**	**31**	**32**	

They're all integral at 1.0. If we drop a little more, to 1 - 5e-5? Now we get to 3'757 with the exact same solution as the == case.

So it appears the difference in hovering around the optimal solution, since the reduced costs are so close to each other, is actually formed by little tweaks in rounding.

Is the optimal solution 3'757? We did get 3'756 with split routes in y-variables, so rounding might account for some of these differences. The only way to truly find out is if we go back to no route pruning, but this will time out.

A final point: Let's try HiGHS Optimizer instead of Gurobi. Same exact code - the only line that changes is "Gurobi.Optimizer" to "HiGHS.Optimizer".

PT >= NyVA HiGHS 1: 3'746.

14	15	16	17	18	19	20		
5	6	7	8	9				
25	26	27						
21	22	23	24	28				
1	**2**	**3**	**4**					
10	**11**	**12**	**13**					
36	**37**	**29**	**30**	**31**	**32**	**33**	**34**	**35**

Yes, the super-long route at the end is actually feasible - I checked.

PT == NyVA HiGHS 1: 3'746. Same exact routes.

Whether we use 1, 1 - 1e-5, or 1 - 5e-5, the routes are exactly the same, and the objective is 3'746. Very interesting.

Conclusion

Once any set of routes gets sufficiently close to the objective, the only difference is rounding. The more leeway you give, the better your result, though in theory you have to be careful not to give so much leeway that you inadvertently let slip a suboptimal result (though in this algorithm, that is practically impossible, as the only thing you're doing is limiting the set of routes you generate in the subproblem, and you can't do better than by allowing all routes in the subproblem to be generated).

In addition to knowing that differences in solutions ultimately came down to rounding errors, there's the fact that HiGHS generates different answers than Gurobi. But that's a matter of the black box called a (commercial) optimization solver, and there's nothing we can do. So we rest satisfied with this answer.