Time Testing CG and PT without y

In "Removing y Variables.pdf" I prove we can remove our y_s/y_e series and their references in the RMP and SP without issue, because our routes, through their nodes, automatically indicate which jobs are performed. In other words, if you go to a location, you will stay long enough to finish the job, and you are marked as having completed it.

I also lamented that CG and PT repeatedly throw up errors, despite my improvements. Specifically, see the document "Timing the Big 4.pdf" for details.

In this document we examine how CG and PT perform without y, and also whether using >= versus == for $\sum_q u_i^q z^q \; ? \; 1$ makes any difference. We will do a time analysis as well.

Wrong answers in the corresponding case are marked with red background and bold white text. Maroon means solution had no route-sharing and was wrong; lighter red means solution had route-sharing and was wrong ("at least it has an excuse").

n	SP (s)	With y (time, s)				Without y (time, s)				SP Obj
		CG >	CG =	PT >	PT =	CG >	CG =	PT >	PT =	
10	0.44	1.53	1.2	1.09	1.18	0.98	0.41	0.58	0.48	2'262
11	0.01	0.06	0.04	0.03	0.05	0.01	0.01	0.01	0.01	2'658
12	0.01	0.14	0.04	0.23	0.04	0.01	0.01	0.01	0.01	2'717
13	0.01	0.12	0.05	0.06	0.04	0.01	0.04	0.01	0.01	2'987
14	0.01	0.14	0.07	0.09	0.06	0.01	0.01	0.01	0.01	3'035
15	0.04	0.24	0.12	0.13	0.34	0.02	0.02	0.02	0.02	2'539
16	0.09	0.35	0.17	0.17	0.2	0.06	0.03	0.03	0.02	2'566
17	0.08	6.31	0.21	0.21	0.2	0.04	0.07	0.03	0.03	3'012
18	0.12	0.25	0.25	0.47	0.21	0.07	0.05	0.03	0.03	3'014
19	0.21	0.49	0.47	0.45	0.62	0.11	0.11	0.08	0.08	3'036
20	0.65	**1.51**	1.24	**1.2**	0.85	0.62	0.4	0.16	0.14	3'028
21	0.83	1.71	1.76	1.26	1.66	0.5	0.59	0.13	0.22	3'257
22	1.03	**2.0**	1.65	**1.58**	1.28	0.56	0.62	0.18	0.17	3'364
23	1.48	4.27	2.83	2.71	1.57	0.72	0.76	0.15	0.17	3'399
24	1.76	**2.82**	3.14	**1.83**	2.63	1.16	1.08	0.27	0.23	3'445
25	4.34	7.14	6.92	4.27	4.32	2.78	3.28	**0.38**	**0.36**	2'845

After the first set of results, we see that without y, CG yields superior performance with no wrong results—and it does so in half the time, if not less. PT also takes on significant improvement, though the route-splitting present with n = 25 (now it's 2'853, 8 higher than without route-splitting) is lamentable. Somehow this occurs both in the = and >= cases! We'll have to test performance later on. One of the important tasks now is to determine a way of 'rounding off' route-sharing.

n	SP (s)	With y (time, s)				Without y (time, s)				SP Obj
		CG >	CG =	PT >	PT =	CG >	CG =	PT >	PT =	
26	5.38	7.91	9.48	3.42	6.79	3.29	3.53	**0.25**	**0.36**	2'856
27	6.46	8.32	8.07	5.18	**4.96**	3.83	3.51	0.34	0.6	2'867
28	9.00	**10.86**	14.56	**4.6**	7.14	6.1	5.18	**0.32**	0.64	2'933
29	12.96	27.12	**17.15**	9.44	13.64	12.38	9.89	0.52	0.76	3'003
30	22.75	38.89	35.91	9.73	**8.08**	21.66	16.37	0.44	0.71	3'282
31	36.11	**38.59**	**37.52**	**8.20**	**11.56**	47.79	31.1	0.58	0.61	3'284
32	91.77	**81.42**	**89.17**	**7.18**	**13.01**	97.54	70.5	0.88	0.9	3'292
33	118.64	**103.20**	**107.68**	**11.87**	**19.05**	92.47	127.68	0.8	0.94	3'406
34	161.89	**176.68**	**188.12**	18.15	**22.88**	168.75	125.73	1.16	1.44	3'649
35	582.22	440.58	469.23	21.07	28.25	319.74	377.21	1.32	1.47	3'314

At this point we see CG has been totally fixed (not a guarantee, but fully patched). Meanwhile, PT still throws errors, but less numerously - and best of all, not only are these all split-route issues, but they are only 8 higher than the objective, if at all! Without manually inspecting each of these route, this is strong evidence that any tinkering algorithm for 'rounding off' route-sharing has a great chance of success. With manual inspection...yes this indeed the case.

Now our PT looks extremely promising (a few occasional shared routes at n = 25, 26, and for >=, 28 notwithstanding). We will be unable to verify whether the objective provided is the true objective, but in cases in which route-splitting is not present (which can be checked by examining the routes and testing if any job is in two routes), we're pretty confident. Where route-sharing exists, we can just subtract 8 to get the true objective (just kidding).

Let's see how high we can go, with >= and with ==. They don't seem all that different anymore. I do also have an idea to "punish" route-splitting with $\sum_q \lceil z^q \rceil C^q$, will test soon.

n	Objectives				Times				n
	PT > y	PT = y	PT > n	PT = n	PT > y	PT = y	PT > n	PT = n	
36	3'335	3'335	3'335	3'335	26.27	24.50	1.24	2.32	36
37	**3'756**	**3'773**	**3'765**	**3'761**	20.42	27.36	1.55	2.01	37
38	3'967	3'967	3'967	3'967	28.78	29.80	1.47	1.87	38
39	**3'979**	3'971	3'971	3'971	30.73	31.74	1.65	3.45	39
40	3'784	**3'848**	3'784	3'784	85.78	86.60	3.57	5.14	40
41	3'857	3'857	3'857	3'857	132.06	148.88	5.91	7.04	41
42	3'966	3'966	3'966	3'966	100.80	75.56	5.36	9.44	42
43	3'970	**4'034**	3'970	3'970	108.10	98.31	8.09	8.01	43
44	**4'083**	**4'050**	3'986	3'986	93.39	163.44	4.8	7.06	44
45	3'295	**3'399**	3'295	3'295	319.69	214.19	9.64	19.11	45
46	Did not record (DNR)	3'498	3'498	3'498	190.89	215.83	10.3	25.32	46
47		3'570	3'570	3'570	246.88	375.16	17.0	18.59	47
48		3'581	3'581	3'581	406.48	487.11	14.08	23.28	48
49		3'621	3'621	3'621	346.19	510.54	20.91	32.83	49
50		DNR	3'532	3'532	1075.16	1566.34	54.22	124.28	50

"y" means the y-variables were included. "n" means they weren't. DNR indicates I forgot to record the objective, either by running another test in excitement or just forgetting.

Although these numbers cannot be exact-proven with SP, if I see a higher objective without route-splitting, I immediately know it is wrong, so it's maroon. Similarly, route-splitting objectives which are higher than other route-splitting objectives will receive a light red highlight. Route-splitting higher than non-route-splitting also gets maroon.

In a rare case in which route-splitting occurs for all 4 values, or route-splitting is lower than non-route-splitting, it will be distinguished with purple bold. Shockingly this actually happens with a route-splitting 3'756 at n=37 for PT >= with y_s and y_e!

Now we have some great results to compare. The no-y-variable class is superior to the yes-y-variable class. Even more impressive is the total lack of route-splitting after/at 38!

The final table of this document is running up the score and testing our magic number.

n	Objectives		Times		n
	PT > n	PT = n	PT > n	PT = n	
51	3'566	3'566	62.22	114.76	51
52	3'675	3'675	68.58	225.3	52
53	3'739	3'739	144.81	156.7	53
54	3'985	3'985	145.36	79.64	54
55	4'503	4'503	23.58	32.63	55
56	4'514	4'514	25.33	34.44	56
57	4'693	4'693	29.89	36.86	57
58	4'803	4'803	23.44	41.31	58
59	5'037	5'037	34.86	36.07	59
60	4'200	4'200	94.51	132.03	60
61	4'647	4'647	120.11	177.65	61
62	4'538	4'538	75.44	140.41	62
63	4'358	4'358	98.53	135.65	63
64	4'517	4'517	121.21	171.72	64
65	4'574	4'574	200.26	203.96	65
66	4'405	4'405	220.27	324.47	66
67	4'638	4'638	165.81	255.21	67
68	4'748	4'748	184.82	444.37	68
69	4'641	4'641	268.63	212.04	69
70	4'796	4'796	183.67	200.87	70
71	4'987	4'987	194.28	236.25	71
72	4'889	4'889	197.68	228.47	72
73	5'041	**5'047**	189.72	242.39	73
74	4'897	4'897	233.10	489.95	74

The threshold was successfully pushed up from 49 to 74. The results between the >= and == cases are almost entirely identical. Once again we have returned to our desired destination!