Stemming Techniques for Generating Hybrid Routes

The Stem-and-Blender Method

At the conclusion of the PT NyVA algorithm, we are not guaranteed to have a perfect answer. Route-splitting is rare, but it does happen. The objective is slightly higher than it could've been without route-splitting.

This document creates techniques for removing route-splitting.

Case Study 1: n = 25, 26, 28 on PT NyVA >=
In all three of these cases, there is a branch caused by:

11 12 15
11 12 13 14

16 17 18 19 13 15
16 17 18 19 14

Two of the routes share a common stem of 11 12, while their possible branches are 15 and 13 14. The other two share 16 17 18 19, with branches 13 15 and 14.

Each of these routes have $z = 0.5$; the others have $z = 1$.

Claim 1: We only need to process $z < 1$ routes.
Proof 1: If a route has $z = 1$, all of its nodes are never going to appear in split routes, otherwise the constraint of "one" node total visiting a job is violated.

It is also worth noting that even in the relaxed constraint "at least one node visits a job", in practice, from n=10 to n=74, I never had a node visited by more than "one" vehicle. One is in parentheses because it can technically be 0.5 of a route + 0.5 of a route. So despite both the relaxed and tight constraints producing routes that satisfy the tight constraint, the >= or == mode of the constraint affects how colgen is processed!

Observation 1: The partial routes put into SP are insufficient.
As is observed in Case Study 1, the only nodes without $z = 1$ routes are 11 through 19 inclusive. But there is no way to combine these routes such that all nodes are visited exactly once by exactly one route. You need to have one of two each in the 11 12 ... and 16 17 18 19 ... stems, yet their branches never perfectly create 13, 14, 15 in some way without overlap or gap. Indeed, when these four routes (and the trivial stay-at-depot route and the $z = 1$ routes) are thrown into SP, it reports no solution.

Claim 2: The "blender technique", or throwing all non-z=1-route-nodes (nzrn) into one big pot and generating all their routes for the SP, guarantees the optimal answer.
Proof 2: CG guarantees that routes with z = 1 are exactly as they stand and cannot be modified. Therefore, the remaining nzrn must be categorized into routes in some way. The blender technique enumerates all possible valid routes, so the optimal solution, which involves the z=1 routes as they are, plus some hybrid combination of the nzrn into routes, is contained somewhere in the blender.

The blender technique is guaranteed to work. However, it is time-inefficient. In the majority of split-route cases, like n = 25, 26, 28, nzrn is few in number (9 out of 25, 26, 28) and the number of routes is rather small. But if we have extensive route-splitting, disaster ensues because we are essentially solving SP (which is slow and times out at n=35) with how many nzrn are there.

Case Study 2: n = 73 on PT NyVA ==
Paradoxically, the PT NyVA >= on 73 gives the closer (not guaranteed correct; job 33's not being included in a route is suspicious) answer with only integer routes, while the == on 73 yields many route splits, with not just z = 0.5, but also 0.25!

Here are the routes for 73 on ==, marked with z-values. (33 is not in here.)
1: 72 17 18 19 20
1: 62 22 23 24 25 26 27 28 29 30 31 32
1: 34 35 36 37 38 39 40 58 60
1: 41 42 43 44 45 46 47
1: 48 49 50 51 52 53 54 55 56 57 59 61

Nodes 1-16, 21, 63-71, 73 are all torn between different routes:
0.25: 01 63 64 65 04 05 06 15
0.25: 01 63 64 65 10 11 12 13 14 15
0.5:   01 63 03 66 67 68 69 70 71

0.25: 16 73 02 03 66 67 68
0.25: 16 73 02 03 66 67 68 69 70 71
0.5:   16 73 02 04 05 06

0.25: 21 07 08 09 65 04 05 06 69
0.25: 21 07 08 09 65 10 11 12 13 14 70 71
0.5:   21 07 08 64 09 10 11 12 13 14 15

The true routes among these, as computed by >= on 73, are:
1: 01 63 64 66 67 68 69 70 71
1: 16 73 02 03 04 05 06
1: 21 07 08 09 65 10 11 12 13 14 15

Observation 2: The true routes have the same stem as the shared routes.
By stem I mean: I hypothesize we can always group routes based on 'shared patterns', like what we see above. Their z-values together will sum to 1, and they have the same start patterns. Specifically, the first three share 01 63; the second three share 16 73; and the final three share 21 07 08.

Observation 3: The entire route-splitting is comprised of DNA fragments.
It's not just the stems that share this pattern - there are fragments to be found later as well! In fact, every single route is composed of these fragments:

01 63
04 05 06
21 07 08
09
10 11 12 13 14
15
16 73 02
64
65
03 66 67 68
69
70 71
64
65

Observation 4: Actual routes might not conform to route-splitting fragments.
A real route is 16 73 02 03 04 05 06. But "03 66 67 68" was a fragment in nzrn.

Because we cannot do things in hindsight or predict how real routes will run, I don't think splitting by fragments is feasible; it will create wrong answers.

Instead, we adopt a stem approach.

First, we extract all stems. Stems are defined as: First, group the routes by starting node. Then, keep increasing the stem until some of the routes with the same starting node are unequal. Stop there; the previous quantities form the stem.

In other words, stems are defined by their first element, and we keep going until we lose an element.

This is the case because even if there are some common fragments of stems beyond the stem, we can't guarantee that that longer stem will be selected.

After we gather stems, we look within all nzrn that have that stem. All the rest of the nodes will be put into a blender, and all possible configurations generated from there.

For example, suppose we had nzrn

1 4 8 3 5 2
1 4 8 3 9 6
1 4 8 5 2 9 6

Then the stem is 1 4 8 (not 1 4 8 3, as we lose the last route if we do this) and the blender consists of 2, 3, 5, 6, 9.

Using a special stem-gathering LSA, we produce all routes with stem 1 4 8 and blender nodes 2, 3, 5, 6, 9. Then we dump all of those routes into the SP.

Claim 3: Separation by stem is far less time-consuming than the full blender.
You select a tiny subset of the number of total possible routes because you will have filtered by stem.

One way to conceptualize route-splitting is chemistry's hybridization of electrons! This is how I convince myself that the final routes have to use the same stems as were used for here.

Observation 5: A lot of this is empirical and based on limited configurations of errors in CG yielding fully integer z-values.
There were only 3 mistakes committed by each of NyVA PT >= (25, 26, 28) and == (25, 26, 73). As a result, the necessity of this document is thankfully small. So I don't have too much data to go off, and I am inferring possible nzrn structure based off these.

We will test them with different seeds to see if the same holds there.

Algorithm for Extracting All Testable Routes

1. Obtain all potential stems. As described above, stems are extracted by their 1st node, and then we go on. We do this with: unique(1st element of each route).

2. Determine which routes belong with that stem. We can use a dictionary. Go through all the routes once and match up the first element.
    1. Yes, there is some clever way to do steps 1 and 2 together, but not right now.

3. Elongate the stem as much as possible. For 2 until the index of the minimum length among all the lists, check that length of unique of ith element of each route is equal to 1 (all of the routes share the same ith element). If not, break out of the loop. First, have a counter. Then, increment if unique is 1...then that will tell us what index up to inclusive we can go.

4. Determine the blender. For each route, extract counter+1 to end, and push those all to a blender. Then get unique of that full blender to know what nodes go there.

5. Return a LIST of [[stem_elements], [stem_times], [blender_elements]] (so it's a 3d array: array of list of lists). This goes to the LSA.

Parameters: We need to have the stem nodes and stem times. It makes most sense to pass in both the nodes_only and times_only arrays which were created in [4.2] and drawn in [4.3].

We assume that, at least up to and including the stem, the times will be the same. So once we determine what the stem is, fetch its times. This goes in between 3 and 4.

Results

This algorithm successfully solves cases n = 25, 26, and 28. But these are the simple cases, and it didn't take too much time...nothing to say, really.

The two cases that really test our algorithm are n = 37 and 73.

n = 73

The answers given by PT NyVA >= and PT NyVA == are different. The former produces 5'041 without route-splitting, while the latter produces 5'047 with. Notably, job 33 is a loner and was unable to be tacked onto a longer route.

The code accounts for this possibility and deliberately marginalizes loner nodes, because they are rare and discouraged. See the commentary at "[4.4] Test for loner nodes".

The SP is forced to contend with 16'434 routes, because the stem-and-blender approach contains large blenders. In particular:

stem 1 63 → blender 4 5 6 10 11 12 13 14 15 64 65 66 67 68 69 70 71 (size 17)
stem 16 73 2 → blender 3 4 5 6 66 67 68 69 70 71 (size 10)
stem 21 7 8 → blender 4 5 6 9 10 11 12 13 14 15 64 65 69 70 71 (size 15)

Eventually it did yield a route with 5'041 objective, consistent with n = 73 on PT NyVA >= (though this is expected). It confirms that the stem-and-blender method takes care of hybridization effectively.

n = 37
This was the only case in which both PT NyVAs failed to yield the correct answer! In fact, PT y >= gave 3'756 with route-splitting, PT y == gave 3'773 with route-splitting, PT NyVA >= gave 3'765 without route-splitting (wrong, and without a chance to do better, because 3'765 > 3'756), and PT NyVA == gave 3'761 with route-splitting.

It meant a route-splitting answer was actually optimum among these four PTs.

By using PT NyVA == combined with stem-and-blender, we were able to reduce the final cost to 3'751. This proves the effectiveness of stem-and-blender in conjunction with a route-splitting-producing algorithm. More subtly, it affirms the worth of PT NyVA == over PT NyVA >=. All route-splitting answers, if sufficiently 'good', can be corrected to the true minimum with stem-and-blender. But if an answer is not route-splitting and wrong, nothing can change its wrongness. Still, PT NyVA >= is significantly faster than PT NyVA == by a factor of 1.5 - 2, so tradeoffs should be evaluated if this thing goes to market. Below, n = 37 using PT NyVA == plus stem-and-blender. Integer routes.

```
Route Any[0, 14, 15, 16, 17, 18, 19, 20, 38] was used 1.0 times!
Route Any[0, 5, 6, 7, 8, 9, 38] was used 1.0 times!
Route Any[0, 10, 11, 12, 13, 38] was used 1.0 times!
Route Any[0, 21, 22, 23, 24, 28, 38] was used 1.0 times!
Route Any[0, 1, 2, 3, 4, 32, 33, 34, 35, 38] was used 1.0 times!
Route Any[0, 25, 26, 27, 38] was used 1.0 times!
Route Any[0, 36, 37, 29, 30, 31, 38.0] was used 1.0 times!
```

```
println("OBJECTIVE: ", objective_value(model_set))
```

```
OBJECTIVE: 3751.494919249296
```