Multi Vehicle Routing under Capacity, Time and Scheduling Constraints

M. Donselaar, R. Koot

March 24, 2009

1 Analysis

We note that out final plan will consist of a number of closed routes from the base location, past several pick-up points and back to the base. We will call each such a closed route a *loop*. In our optimal solution it will not matter in which relative order the loops on the same day are serviced.

2 Design

Our design based around a tabu search algorithm, as it is easy to implement and *Metaheuristics for the Vehicle Routing Problem: Fifteen Years of Research* suggests is performs well.

2.1 Datastructure

Our current solution is stored as a tuple of two lists. The first lists corresponds to the loops in our solution and are tagged with the current truck and day assigned to it as well as the first order of the loop. The second list contains the orders and linked on the order that follows is in the loop. A special value indicates we have reached the end of the loop en must return to the base. Loop 0 is reserved for the orders that we have decided to skip. When moving an order to loop 0, we therefore do not consider its position in the loop.

2.2 Neighbour space

Our neighbour space consist of a single operation performed on the current solution:

Move order x from loop a to loop b before order y. This is a simple operation which encompasses almost everything we want to do. It allows us the choose to skip or not skip an order by moving it from or to loop 0. In our implementation we decided to not allow that $a = b$. This would have required checking for several corner cases (expensive) and this can still be achieved indirectly by first moving an order to another loop and then back to the original loop but in another position. It is also allowed that b is currently an empty loop, in effect meaning that a new loop is added to the solution.

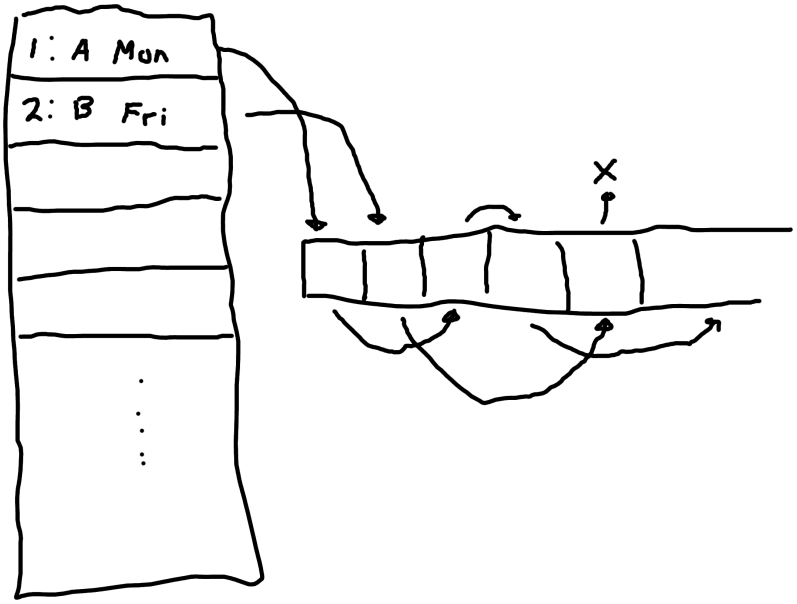

Figure 1: Datastructure representing a solution.

In our original analysis we had also considered the operations:

Assign a loop a to day i and truck j. However we experimentally found that it is possible to schedule nearly all orders using only a single loop per truck per day, with loops taking usually over 6 hours to service, making this operation ineffective.

Swap order x with order y. This might be beneficial over simply moving orders between loops, when a loop is against its capacity constraint. Be decided not to implement is, as it adds additional computational complexity, out local search algorithm seems to perform good without it and its effect could be simulated by moving orders and (temporarily) allowing the capacity constraints to be violated.

2.3 Tabu space

We had originally started very conservatively, making only specific solutions tabu. This made our local search algorithm cycle, making an insignificant change somewhere and then selecting the same repetitive pattern of orders to move between the same loops. This suggested we should instead make the orders we select to move tabu instead. This worked well and had the additional advantage of being computationally much less expensive that checking whether or not an entire solution was tabu or not. We initially started out with a tabu list of length 32 (a power of two that is approximately the square root of the number of orders) but experimentally found that increasing it to 256 would cause our local search algorithm to converge the fastest to the best solution.

2.4 Initial solution

We initially start with all orders unplanned (meaning all orders are in loop 0). It turns out however that adding a new loop to the solution can be prohibitively expensive. A loop containing a single solution will have costs of travelling from the base to the order and back and an additional half hour of emptying the truck. This is often more expensive than than the penalty incurred by not scheduling an order at all. We have tried to overcome this by increasing that penalty, but found it was more effective by forcing orders to be planned in the beginning, by allowing only move from loop 0 during first n iterations, where n is some number approximately as large as the number of orders.

3 Implementation

3.1 Mutation

To achieve the maximum number of iterations per second, the speed of the finding the best neighbour is crucial. To achieve this we used the following tactics in our implementation of the local search algorithm:

Cheap evaluation function. The evaluation function is called on each element of the neighbourhood space, making its performance critical. We transformed this to a very simple function that has a runtime each to the number of loops, by caching the time and total volume of garbage for each loop instead of calculating this number over and over again.

Avoid updating the solution. After caching the total time and volume for each loop, it is no longer necessary to update the solution when evaluating each neighbour. We only update the cached value (which can be calculated easily) before calling the evaluation function and update the solution only once after having determined which neighbour is the optimal one.

We made another controversial decision:

Do not plan orders that have a frequency of more that one time per week. After inspecting the data set we found that the number of those orders was quite low (45 orders of frequency 2, 4 orders of frequency 3, 1 order of frequency 4, and no orders of frequency 5 at all.) The orders of frequency 3 and 4 are also quite insignificant, not justifying planning them knowing that our local search algorithm would already decide not to plan approximately 10 similar orders of frequency 1.

The orders of frequency 2 might be worth considering, especially given that it contains a single order that would be very expensive to skip. Some preliminary work (which can be found in `r5/`) on expanding our local search algorithm to support orders of frequency 2 show that this had adverse effects on the performance: the number of iterations per second decreased by over an order in magnitude, and the solutions it produced where lower in quality (after the same number of iterations) as our old solver.

It might be worth first planning (automatically or even manually some of) the orders of frequency 2 and then continue planning the orders of frequency 1 using our solve, this would require further research.

Using these tactics we were able to have our local search algorithm perform 100 000 iterations in several hours on a Core 2 Duo T7300 2.00 GHz.

4 Output

Here follows some sample output generated by our implementation each iteration:

```
I have decided to move order 303 from loop 8 to loop 1 (just before order 803).
****** 3027: 100:53
SKIPPED=75 (1: 25  2: 45  3: 4  4: 1  5: 0)

  1: A1 (volume=80410/100000) (time=4:42|4:42) (orders=90)
  2: A2 (volume=99490/100000) (time=6:31|6:31) (orders=94)
  3: A3 (volume=99910/100000) (time=6:27|6:27) (orders=116)
  4: A4 (volume=99630/100000) (time=6:46|6:46) (orders=96)
  5: A5 (volume=99610/100000) (time=5:45|5:45) (orders=123)
  6: B1 (volume=99410/100000) (time=6:46|6:46) (orders=137)
  7: B2 (volume=99960/100000) (time=6:39|6:39) (orders=109)
  8: B3 (volume=99150/100000) (time=7:20|7:20) (orders=111)
  9: B4 (volume=99780/100000) (time=6:41|6:41) (orders=106)
 10: B5 (volume=99990/100000) (time=6:50|6:50) (orders=103)
 11: A2 (volume=25370/100000) (time=1:27|1:27) (orders=17)

A1=  4:42    A2=  7:58    A3=  6:27    A4=  6:46    A5=  5:45
B1=  6:46    B2=  6:39    B3=  7:20    B4=  6:41    B5=  6:50

TOTAL=1177/1177
```

It shows the last mutation that was performed on the solution, the current iteration and valuation of the current solution, the number of skipped orders (split per frequency), the loops in the current solution with its attributes (truck, day, volume, time, orders), and the total time per truck per day.