Vehicle Routing with Time Windows – Tutorial

Philip Kilby

17 June 2017

Note: My answers for these exercises are in the *answers* directory. You can cheat if you want, but you'll obviously get more out of it if you have a go yourself first.

1 Exercise 1

In class, we used the *bin_packing_capa* global constraint to enforce the capacity constraint. A more natural model is to limit the load at the *end* visit.

Change the model in `vrp.mzn` file to model this constraint.
Hint You will want to define a variable q to hold the load after completing the visit.
Hint Remember, the index of the end visit is `NumCusts + NumRoutes + k`.
Hint Performance issue: A problem in MiniZinc means that *unbounded* variables do not perform well. To define q, use

```
array[Visits] of var 0..max(Capacity):  q;
```

Setup: On configuration tab:

- Choose datafile `A-n10-k2.dzn` from directory `data`
- Set *Time limit* to 60 seconds
- Select *Solver* Gecode (bundled)
- Select *User-defined behavior*
- Select *Print intermediate solutions*
- Set *Compress solution output...* to 2000
- Set *Solver flags* to `-restart luby -restart-scale 1000`
- Select *Statistics for solving*

2 Exercises 2

You will recall (I am sure) that in the *Orienteering* variant of the VRP, we don't have to visit all customers, and the objective is to maximize the *value* of customers visited.

Unlike the model presented in class, this allows some customers to not be visited.

Change the model presented in class (using the bin-packing version of the capacity constraint) to allow customers to not be visited. Also change the objective.

- For the *value* of a customer i, we will use *ValueMult* $\times$ Demand[i].
- The data files do not contain a value for *ValueMult*, so the system will prompt you for a value. This will let you play around with different values for the multiplier.
- Rather than limiting time, as is usual for Orienteering problems, we will limit distance
- Define a parameter *MaxDist* for the maximum distance. Since we do not provide a value in the data files, the system will prompt for a value of this too
- Note that in the initialisation of *dist*, the distances are multiplied by 1000 (to avoid rounding errors). So, don't forget to multiply MaxDist by 1000 in the constraint.
- Data for this exercise is in `data`, and there are also infeasible problems in `veh-1` that have 1 too few vehicles

Hint You may want to define an extra route (route `NumVehicles + 1`) where the *unassigned* visits reside

Hint You will want to define a Value array to store the calculated value for each customer

Hint You can limit the application of a list comprehension in MiniZinc with a *where* expression. E.g.

```
sum (i in Visits where routeOf[i] != NumRoutes)
```

3 Exercise 3

In another variant – the *Profitable Tour Problem* – we wish to select the customers to visit in order to maximize profit, defined to be the sum of the value of visited customers, less the cost of visiting them.

Modify the model from Exercise 2 to model this problem.

- Use *Value* as defined in Exercise 2
- Use the total distance of a tour as the cost (i,e, one unit of cost == one unit of distance.
- Remember when setting *ValueMult*, that the distance is multiplied by 1000.

4 Excessive 4

So the tutorial is called VRP with Time Windows, but where are the time windows?

Here!

Modify the original model to cater for time windows.

- Use *dist* between customers for the time also.
- Define *Early, Late* and *Dur* (duration) for each customer. (The data files use these names, so make life easy for yourself and use these too.
- Data for this exercise is in `tw`.

Hint I found it useful to define *VisitDur* (in the same way we defined *VisitDemand*) for all visits (not just customers). Duration of start and end visits is 0.

Hint Like q for load, it will be useful to define t for the time service begins at each visit.

Spoiler alert!

Hint to be read only if you can't find solutions to simple problems I spent a frustrating time trying to find a solution until I realised I had set service time at my succ *equals* my service time + travel-time to succ + my duration. But of course, to satisfy the early time constraint, the vehicle may have to wait at a customer. Hence, the service can start any time at *or after* that time. Doh!