VRP Solver Description

Although I wouldn't claim this to be state-of-the-art at the moment, I think it does reasonably well and should demonstrate how I approach these sorts of problems.

It supports CVRP and VRPTW problem instances and could be easily extended to support DARP, and VRPPDTW problem instances.

I've included a log of a sample run, run on 30 or so problem instances. It seems to be reaching 100% of best known on many problem instances and not as well on others. With a little work I believe I could bridge this gap.

As a little history, in late December, I was playing around with some exercises around the Unit Commitment problem, when I put together some simple solver tools (more a design pattern than anything) and I built the VRP solver on top of that… And included the Unit Commitment Problem solver as well, for reference. The design pattern enforces a separation of the Solver from the Problem.

The approach I took, was to define the variables as the customers and the domains as the neighboring customers or depots (i.e. vehicles) where this variable's (i.e. customer's) route can be inserted into. The number of vehicles is either an input from the problem instance, or its a multiplier on the total demand / vehicle capacity. So when we're searching a neighborhood, we can remove a customer from one route, add them to another, etc., etc..

The Solver tools (Iterated Local Search, Local Search, Depth First Search, Random Search) can then search neighborhoods of these variables.

I implemented incremental scoring, where unmet demand is highest priority, followed by lateness, distance travelled, and overCapacity. Although capacity should never be violated, I felt that if it would temporarily violate this constraint, it would work its way out of it.

There is also a hard constraint on capacity, which I relax a bit, and is mainly to reduce the # of solutions that have to be explored.

Unfortunately, handling time windows required pushing the arrival time forward through the end of a route. For future work, I believe I can collapse the time window constraint for route visits that are not part of a neighborhood search and thus eliminate the need to propagate the arrival times up the route -- for the portion that's not in the neighborhood being searched.

I defined the neighborhood in the following way.

- Travel distance is converted to time (duration),

- a latest departure time is defined as due time + service time,
- a min wait time is then the customer's ready time - (latest departure time + trip duration)
- an earliest departure time is defined as ready time + service time.
- then a neighbor's closeness is defined as Max(min wait, trip duration, min late)

Basically, this defines a neighborhood structure that takes time windows into account.

In terms of the algorithm, it loosely implements Iterated Local Search. The perturbation is just some random moves -- the size of the perturbation changes randomly as well. I think it's better than random restart, but I need to experiment with this. Local Search is a series of Depth First Searches on small neighborhoods of random size.

The variable neighborhoods / domains, return neighbors based on above closeness measure. If the algorithm gets stuck, then it will randomly return neighbors that are farther and farther away.

One thing to note is: we can't add a customer to a neighbor's route until the neighbor has a route. This means that initially many customers have no active neighbors unless one of their close neighbors is a depot, for this reason, until all demand is satisfied (hence all neighborhoods active), we build neighborhoods, so that neighborhoods with active neighbors get returned first. Once these neighborhoods get assigned routes, more neighborhoods become active, and so forth, and so forth.

I ran into some cases where there would be a little left over unmet demand. Since I didn't have time to dig into it deeply, I added an element of repair -- if algorithm is stuck and there's unmet demand, find a route that isn't full and add it to customer's domain.

The Solver toolbox talks to the problem mainly via GetNeighborhood, GetDomain, UpdateValue, and GetScore methods (which essentially directs the toolbox), and a few notifications (e.g. stuck).

There are a number of algorithm tuning parameters: num iterations, min/max neighborhood size, and max domain size. I tuned these a little to the problem instances.

You'll notice that I had to do a little trickiness when restoring a group of variables .vs. updating a single variables (as in during search). This is because the variables may be restored out of order and the linkages could be incorrect until the restore is completed. I may be able to clean this up but haven't had time to do so.

The file Tester.cs is where the entry point it. You can run it from there, and will generate a VRP Solver.log file which will contain the results.

Future work:

- Tune the algorithm
- Optimize code for performance. I was unable to find runtime numbers for the problem instances so don't know how well I'm doing.
- It would be easy to parallelize the iterations of Local Search. Then during acceptance, it could consider the results from all the threads.
- More difficult is to parallelize the neighborhoods, if multiple threads optimize neighborhoods that share a route.
- Clean up the code a bit.
- support more types of problems