Capacitated Vehicle Routing Problem

Larsen Cundrič, Domen Mohorčič

10. januar 2021

1 Introduction to problem

Vehicle Routing Problem (VRP) is a well known generalization of the famous Traveling Salesman Problem (TSP), which is proven to be NP-hard. In VRP we try to optimise a set of trips such that every node in a graph is visited at least once and all trips start and end in a starting node (depot). Capacitated VRP (CVRP) is extension of VRP such that all vehicles have a limited carrying capacity.

In our case we have a CVRP, in which we try to optimise route for collecting different types of garbage (organic, plastic, paper) from nodes. Our problem also has some additional constraints to basic CVRP:

- each truck can only pick up one type of garbage,
- some roads have maximum load capacity; a vehicle with a higher current load cannot use those roads,
- if a vehicle passes a node with garbage and has available capacity for picking it up, it has to pick it up; if a vehicle does not have available capacity, it just travells by and does not service it,
- when picking up garbage a truck must pick up everything; it cannot partially empty a location.

As an input we have a directed graph of nodes and edges. Nodes represent sites with different garbage amounts. Edges represent one- or two-way roads with maximum load capacities and their lengths. Number of trucks is infinite.

We try to minimize the cost of picking up all the garbage at all sites. The cost of a trip includes:

- a fixed starting cost,

- cost for travelled distance in a trip,
- paycheck for the employee that drives the truck.

Each trip has a starting cost of 10. The distance travelled is priced at 0.1 per one unit of travelled distance. Hourly cost of an employee stands at 10 per hour for first 8 hours and 20 per hour for remaining overtime. Time of each trip is calculated using speed of truck (50 units per hour) with distance travelled and time spend servicing nodes (0.2 hours for picking up garbage and 0.5 hours for emptying a truck at the end of the route).

We try to minimize previously described cost function for minimizing cost of picking up all trash. Since one truck can only carry one type of garbage we can treat each garbage type as its own problem.

We used two different approaches for solving this problem. One is a greedy algorithm and the other is simulated annealing with probabilistic local search.

2 Greedy algorithm

The idea of our greedy algorithm is to travel to the nearest site with garbage, so that we can pick it up (we have space for it). If no such site can be found, we make a trip to starting node and continue. We stop when all garbage is collected.

```
solution is an empty set
current location is starting location
start new trip

while( exists a location with uncollected garbage ) {

  do {
    find nearest uncollected location such that we can collect it
    find shortest path from current location to target location
    pick up garbage
    update current trip with visited location and picked up garbage
  } while ( exists a location with uncollected garbage we can pick up )

  find a path back to starting location
  add current trip to solution set
  start new trip
}
```

When finding a path from one location to other the truck doesn't pick anything up. This is garanteed by finding closest location from which it can

collect. If a location on its path happens to be collectable, the previously mentioned target location is by definition not the closest, since there is even closer location. Therefore there cannot exist a collectible location on its path. When there are no closest locations it can collect from the truck can return to starting node using any path provided the roads can carry it. On its path it will also collect nothing, as there is no location from which it can collect. Therefore it can return to starting node without any need for additional pickup computation. Then it can begin a new trip.

The above pseudocode is implemented in funciton greedySolution. Its parameter is a vector of garbage data from locations. It returns a solution in two different forms: a complete route for all trips combined (e.g. 1 2 3 1 5 1 4 1 meaning 3 trips with ones separating them) and list of pickups in order (e.g. 2 3 5 4). Our implementation uses precalculated matrices for shortest paths for every node combination with respect to carrying capacity of a vehicle. We calculated it using modified Floyd Warshall algorithm in function allPairsShortestPath This metrices make finding solutions easy and quick but take a lot of time to calculate, so it cannot be done as part of the greedy algorithm. Computation of all pairs shortest path metrices must therefore take part when reading the data, making it independant from our algorithm.

3 Simulated annealing with probabilistic local search - SAPLS

For our local search solution we implemented simulated annealing (function AlgorithmSA) with probabilistic local search. Simulated annealing starts with an initial solution and iteratively switches to better random neighboring solution or with a probability to a worse one. This probability depends on temperature and is high at the beginning. Temperature decreases every time our random neighbor is not better. When temperature drops below a certain value we find a local solution using our best current solution from simulated annealing and probabilistic local search. We decided to go with probabilistic so that we can control how big of a portion of the neighborhood is evaluated as it can become very big.

Our solutions are represented in permutation form with fixed length. Solution 2 3 5 4 means we have to find shortest paths 2-3, 3-5 and 5-4 such that nothing is collected in between. This representation simplifies neighborhood calculation since we can use simple swaps of elements similarly as in TSP.

The neighborhood is calculated in function neighborhoodPerm and gene-

rates a set of all possible swaps between elements of the provided solution. Since not all generated solutions are valid, cost function later assigns value of infinity to easily distinguish them from valid ones.

Our probabilistic local search (function AlgorithmLS) takes a certain fraction (passed as parameter) of the calculated neighborhood, calculates the cost for all neighboring solutions and takes the best neighbor. This cost is then compared to the best one known so far. If it is better than the current best, it does the whole process again, otherwise it returns the known best solution.

Our cost function (costVector for complete route and costPerm for routes in permutation notation) for evaluating solutions also uses precalculated matrices for shortest paths for every node combination with respect to carrying capacity of a vehicle.

The key to calculating good solutions relatively fast is in starting soltion. If it is bad, neighboring solutions are also bad and it can take some time to reach good solutions. First we tried to randomly generate initial solution, but later realised we can use the solution of our greedy algorithm. This produced good results with relatively fast convergence on local maxima.

4 Results

We got the following results for problems using greedy algorithm and SAPLS:

Problem	greedy alg.	SAPLS
1	178.70	159.57
2	965.89	920.49
3	4815.14	4221.47
4	21143.48	18671.48
5	10562.16	9187.89
6	1368.44	1252.66
7	42290.67	38159.16
8	44148.05	39034.17
9	161712.63	158018.27
10	80045.22	66315.04

In all cases solutions of greedy algorithm were inputs of SAPLS, which could only further improve them. Greedy algorithm is deterministic, so a solution to a particular problem is always the same, while the same cannot be said for SAPLS. Although local search is deterministic, simulated annealing and probabilistic local search are not. That way we always get a different solution, although not very different as we always have the same starting solution.