AAHPS Assignment 5

Janez Justin, Jaka Vrhovec

January 2021

Introduction

In all solutions it is assumed, that there is no more trucks needed than there is number of sites to visit.
All solutions split the vehicle routing problem into three separate independent problems - one for each type of trash to collect.

Program 1: Simulated annealing (SANN/solution.R)

When constructing graph in this solution, only the shortest path $u \to v$ is used. If there exists longer path between two sites it is discarded. There is a different approach (SANN/moreEdges.R) using same method, where all connections are considered. But due to high time complexity of getting all connections leading out of some site for some weight of the truck we omitted this idea.

Starting solution

The idea of this solution was to generate a starting solution in which each truck visits its own house, while traversing down the tree generated by dijkstra's algorithm starting at the disposal facility.
Example on problem #1:

```
Truck 1: 1 -> 2 -> 1
Truck 2: 1 -> 3 -> 1
Truck 3: 1 -> 4 -> 1
Truck 4: 1 -> 5 -> 1
```

Finding neighbours

After the starting solution is produced neighbours of current solution are generated in two ways.

- Add new site to random truck after random visited site and then find shortest path back to the route it is supposed to go down.
- Switch the order of two random trucks. Let's say truck x was sent out of the facility as k-th truck, truck y was sent out of the facility as l-th truck and k-th and l-th trucks are chosen to be switched. Then x is sent as l-th truck and y is sent as k-th truck.

Finding minimum with simulated annealing

For each problem there is a list of parameters to be set, that are taken into account while searching for solutions:

- *it* - after how many iterations of not finding any better solutions to take current best solution as best.
- *temp* - starting temperature
- r - probability of using the 'add new site' method to find neighbour

Parameters have been set based on some testing. In general we noticed, that for bigger problems(7, 8, 9) better results are produced when *temp* is higher and $r \approx 10$. And for problems with more connections relative to the number of sites(4, 10), small r does the trick of producing better results.
Temperature in each iteration is calculated using the following formula:

$$T_i = \frac{temp}{log(i+1)}$$

To get the results each problem has been started finite number of times. Out of all produced solutions best one has been taken.

Results

Cost of problem 1 : 159.5727
Cost of problem 2 : 988.1228

Cost of problem 3 : 5935.459
Cost of problem 4 : 25401.63
Cost of problem 5 : 12551.45
Cost of problem 6 : 1470.999
Cost of problem 7 : 53340.71
Cost of problem 8 : 55070.59
Cost of problem 9 : 193116.4
Cost of problem 10 : 91416.33

Comment

Compared to later solutions, results produced by this method can be considered bad, except for the first problem. The solution is quite time complex and can take a lot of time to produce "good" results. Compared to later solutions this one is worst time and cost wise.

Program 2

Program 2 consist of three parts:

- First greedy algorithm
- Second greedy algorithm
- Differential evolution

Third part expands on first part. Second part's methods are included as effectiveness comparison to first part.
Part 1 and 2 are not using any type of optimization method after first solution is produced.

Program 2.1: Greedy (g/Main.java)

Learning from the previous method, where calculating shortest paths approximately every third iteration took a lot of time, we decided to try to construct as good of a solution when generating starting solution. This lead to much better starting solutions than Program 1 gives us in a few seconds.

The idea behind this solution is to use Floyd-Warshall algorithm to pre-calculate all shortest paths between all nodes for all integer weights between 0 and maximum possible trash on single truck.

Starting solution

When constructing the starting solution we always start from 1 and call findBestPathFor(). This function gives us the shortest path to the closest node, that still has some trash to collect and can be collected(current truck has enough space in it).
Found path is appended to currently known path until path and process is repeated until path leading to the disposal facility is returned.

Path finding

```
findBestPathFor(fromSite, curentStateOfSites, currentTruckWeigth){
    // distances from fromSite that can handle ciel(currentTruckWeigth)
    dist = distances[ciel(currentTruckWeigth)][fromSite]
    goto = argmin(distances)
    return path(fromSite, goto)
}

path(u,v){
    return reconstruction of shortest path from u to v from floyd-warshall
}
```

Results

Problem 1 cost: 163,699
Problem 2 cost: 965,889
Problem 3 cost: 4869,331
Problem 4 cost: 21143,477
Problem 5 cost: 10646,226
Problem 6 cost: 1368,440
Problem 7 cost: 42295,583
Problem 8 cost: 44148,046
Problem 9 cost: 161712,632

Problem 10 cost: 80051,770

Comment

Compared to previous solution, this one is much better, except for first problem. Counting in the reduced time complexity the solution is miles better (and could possibly be improved by introducing some parameters on how to choose new paths, which can be optimizied using one of the search methods).

Program 2.2: Greedy2 (g2/Main.cpp)

Main difference to the program 2.1's solution is that this solution constructs a few paths and then picks the best from all computed ones. Also for picking the next best node there's 0.5 probability that it will pick node that has highest $\frac{weight}{distance}$ ratio or 0.5 probability that It will pick the closest not cleared node. In first case there's possibility that in the path from current node to the chosen node there's node that truck will have to pick garbage, in that case we pick the garbage and reevaluate the most optimal node. In both cases we continue until we can't pick any more garbage.

Results

Problem 1 cost: 164,485
Problem 2 cost: 961,663
Problem 3 cost: 4712,962
Problem 4 cost: 20681,004
Problem 5 cost: 10412,414
Problem 6 cost: 1300,358
Problem 7 cost: 41565,509
Problem 8 cost: 43556,916
Problem 9 cost: 162389,296
Problem 10 cost: 79849,972

Program 2.3: Differential evolution (de/Main.java)

This solution is an extension of Program 2.1.
It expands on findBestPathFor(...) by implementing parameters for:

- Max hours a truck can wander around for when constructing a new path.
- Max length of single truck route
- Minimum required amount of empty space in truck when returning to facility
- Chance of choosing a random site when choosing what site to visit next

If there is no path to choose from, because of the constrains then shortest possible path out of all paths is chosen (just like in Program 2.1).
Parameters listed above are then used as values to evolve on during differential evolution.

Differential evolution

We used the algorithm that was introduced on the tutorials.
Rate of crossover and mutation scale factor are chosen randomly before starting differential evolution for each of the three subproblems.
In each iteration new solution is generated based on the newly mutated parameters. In comparison to program 1 changes are not done on solutions that have good costs, but it generates entirely new solutions in each iteration.

We noticed that parameter of randomness is used only in problems 1,2,10.
Limiting the amount of trash that can be picked up by a single truck might seem like a dumb idea at first (and in most cases it is) but problem 4's solution ran with ≈ 14 units of free space.

Results

Problem 1 cost: 159,573
Problem 2 cost: 939,241
Problem 3 cost: 4626,993
Problem 4 cost: 19862,932

Problem 5 cost: 9693,645
Problem 6 cost: 1307,397
Problem 7 cost: 39823,420
Problem 8 cost: 41322,645
Problem 9 cost: 160202,471
Problem 10 cost: 71880,977

Comment

This solution is a lot more time consuming than 2.1 and 2.2. But the results are the best of all the solutions.

Conclusion

Programs 2.1 and 2.2 are are very time (but a lot less space) efficient compared to Program 1 and they produce quite good results. Program 2.3 produces slightly better results than greedy algorithms, but it needs a lot more time to find these solutions.