Constructive Methods

Ronald Cardona Martínez y *Juan Carlos Rivera*
rcardo11@eafit.edu.co, jrivera6@eafit.edu.co

Departamento de Ciencias Matemáticas
Escuela de Ciencias
Universidad EAFIT
Medellín – Colombia

Abstract

Exact methods for solving Vehicle routing problem (VRP) are only usable in small problems because they become more computational expensive as the problem grows. The Clarke and Wright's savings method is the default used heuristics for the VRP. It is an effective method which reaches a reasonably good solution for small and medium size problems in a modest amount of time. For large-scale VRP, more complex heuristics have been proposed. In this report, we use an improved savings method proposed by *Xing et al.*. This improvement is used to solve a generalization of the VRP, the Heterogeneous Cumulative Capacitated Vehicle Routing Problem (HCCVRP). .

Keywords: heuristics, constructive methods.

1 Implemented algorithm description

The algorithm implemented is a slight variation of the savings algorithm proposed by Clark and Wright [1]. In this improvement [2], authors aim to optimize the process of selecting which savings and adding which customer to the route by changing the rules of selection as we will see below.

First we are going to review the classical savings algorithm and then we will see the differences with the improved proposed version.

1.1 Classical savings method

The savings method tries to minimize total distribution distance. But for our instance of the problem we are going to take into account other practical constraints, such as time of arrival at each demand point and the velocity of the vehicles.

If buses start from school D_0 to two pick up points C_1 and C_2. Distances between school to pick up points C_1 and C_2 are l_{01} and l_{02} and the distance between pick up points is l_{12}. The demand of two pick up points is no more than the capacity of the buses. The total distance of separate pick up is $L = 2(l_{01} + l_{02})$ while the total distance by combined pick up is $L' = l_{01} + l_{10} + l_{12}$. Then the saving is defined as follow $S = L - L' = l_{01} + l_{02} - l_{12}$. The saving means the distance that is saved by using a combined route rather than a separated one for more than one pick up point. Then the savings are ranked in descendent order and customers are added to routes according to the rank until the capacity of vehicle is reached.

1.2 Improved savings method

So far, in the savings algorithm some bigger savings couldn't be added to the route because of the constraint of vehicle capacity. So we have to choose to add another pick up point to the route with

smaller savings or even worse, select the same pick up point but assign it to another route with smaller savings.

According to this, the improved savings method aims to maximize the use of those bigger savings by changing the merging method as we'll see below. In the improved savings method the first two steps remain unchanged, that is, calculate distance matrix and savings. Here is the definition of the method as described in [2].

1 Calculate euclidean distance between every two pick up points.

2 Based on the distance, calculate the savings between every two pick up points as described above. Then, rank the savings in descendent order.

3 Choose two customers with maximum savings, satisfied the bus capacity constraint, this is the initial route.

4
- In the remainder of savings, choose the largest one that contains same pick up point of either the start or end of previous route.
- Decide whether this savings is the largest one for this point. If not, turn to the second largest savings.
- Add the point to previous route if the sum of the demand does not exceed the capacity constraint.

5 Repeat step four till no pick up point can be added to the route subjected to the capacity constraint.

6 Repeat steps three, four and five till no customer can be added to some route.

In terms of our implementation we used two different approaches: The first one aimed to solve the heterogeneous version of the problem in one single run. We then faced a problem in fulfilling the vehicles heterogeneous capacity constraints. This took us to the second approach used, where we tried to divide the heterogeneous problem in a set of homogeneous smaller problems. This resulted in a better version of the algorithm, but it still couldn't guarantee that all nodes were visited.

2 Results

In this section are described the experiments made and the results obtained. All the results obtained in this section are under the following software and hardware commodities:

- Python programming language
- 2.3 GHz Intel Core i5
- 16 GB 2133 MHz LPDDR3 memory

This section is divided as follows, first a comparison with a lower bound obtained by ignoring some constraints is presented, next a review of computational time obtained and order of complexity and finally we compare our solution with Google ORTools Solver [3].

2.1 Lower bound

The proposed objective function is defined as:

$$\min Z = \sum_{i \in V_0} q_i * t_i$$

Table 1: Lower Bound comparison for HCCVRP

Instance	***n***	$Z(S^h)$	$Z(L_b)$	% L_b
1	50	847.512	833.423	98.3
2	50	1367.227	988.503	72.2
3	50	731.093	858.512	117.4
4	75	742.923	858.512	115.5
5	75	1825.593	1747.034	95.6
6	75	982.182	876.619	89.2
7	100	1109.237	1076.977	97.0
8	100	1185.046	1256.775	106.0
9	100	969.521	876.619	90.4
19	100	1620.707	1878.100	115.8
20	100	1701.766	1747.681	102.6
21	100	1931.130	1747.034	90.4
16	120	2173.401	1748.030	80.4
17	120	2175.563	1847.713	84.9
18	120	1322.339	1571.723	118.8
10	150	1015.298	964.781	95.0
11	150	1126.212	1057.256	93.8
12	150	835.069	908.906	108.8
13	199	1283.992	1221.839	95.1
14	199	1087.374	1221.839	112.3
15	199	2047.361	1899.720	92.7

The objective is to minimize the weighted sum of arrival times at demand nodes.

To calculate a good lower bound to compare our solution we decided to cut off one of the most important constraints in the problem definition: the vehicle's maximum capacity constraint, so that in some cases the maximum capacity of a given vehicle could be surpassed.

As we can see in Table 1, the $\%L_b$ reached by the algorithm depends on the detail of each specific instance. In some cases S^h was better than L_b which means that we cannot entirely trust the results given by our implementation of the improved savings method. Mainly because in the experiments we observed that for some instances, the algorithm failed to reach a feasible solution. The algorithm couldn't guarantee for some cases visiting all nodes in the route while generating one route for each vehicle.

But in general, S^h and L_b are fairly close.

2.2 Time and Complexity

As seen in the algorithm description in section 1.2, our implementation of improved savings method only differ from the classical one in the criteria used for selecting and merging routes.

The algorithm begins by calculating a distance matrix and then a savings matrix which in terms of runtime scaling corresponds to an $O((n+1)^2)$, being n the quantity of nodes to visit. We then have to sort the flattened n^2 long savings matrix, for this purpose we used a built-in quick-sort algorithm which yields $O(\log n^2)$, actually this sorting phase is the most important one, because in a latter step we will search for a new connection in a sorted array which speeds the process. And finally we have to add all the nodes to a route, this is done by checking at the savings matrix for a node which haven't been visited and checking feasibility, plus the merging $O(n^2)$ phase. This is the core of the process and takes $O(n * n^2)$. This leaves us with an overall runtime scaling of $O(n^3)$. One can be pessimistic about this order of magnitude, but, actually in our experiments (Table 2) the runtimes we found depended primarily on each specific instance. For example, a 200 node instance could be resolved in far less time than a 100 node instance.

Table 2: Runtime for HCCVRP

Instance	***n***	***time***
1	50	0.029641151
2	50	0.216788054
3	50	0.025573969
4	75	0.027072906
5	75	0.426993132
6	75	0.247075796
7	100	0.118534088
8	100	0.196800947
9	100	0.286124945
19	100	3.265247107
20	100	0.280575991
21	100	0.363944054
16	120	0.820427895
17	120	0.876606941
18	120	3.073361874
10	150	0.126315117
11	150	0.120033979
12	150	3.154281855
13	199	0.193500996
14	199	0.196749926
15	199	0.873731136

3 Conclusions

- As we have seen in the algorithm description section, this improved version differs from the classical one only in the criteria it uses for selecting the next nodes in the route and the merging.
- As this changes are just rules in step 4 of the method , potentially $O(1)$, we can just say that in terms of runtime scaling the classical and the improved version we used are the same.
- In terms of our implementation, we haven't achieved a version which can satisfy all the constraints for every instance. Mainly because we couldn't fulfill vehicle capacity and number of routes constraints at a time.
- The improved version of the savings algorithm couldn't be trustfully compared with the classical version beyond that its definition, mainly because we haven't implemented it.

References

[1] G. Clarke and J.V. Wright. Scheduling of vehicles from a centraldepot to a number of delivery points. Operations Research, 1964 (12) , pp. 568–581

[2] Xing, W., Shu-zhi, Z., Xing, W., Hao, C. (2016). An Improved Savings Method for Vehicle Routing Problem. 2016 2nd International Conference on Control Science and Systems Engineering (ICCSSE), 1–4. https://doi.org/10.1109/CCSSE.2016.7784340

[3] OR-Tools is an open source software suite for optimization, tuned for tackling the world's toughest problems in vehicle routing, flows, integer and linear programming, and constraint programming. https://developers.google.com/optimization/