Genetic Algorithm for the Capacitated Arc Routing Problem

Pengnan Lai 11610702

School of Computer Science and Engineering Southern University of Science and Technology

Email: 11610702@mail.sustc.edu.cn

1. Preliminaries

The Capacitated Arc Routing Problem(CARP), It is a difficult vehicle routing problem. can be described as follows: consider an undirected connected graph G=(V,E), with a vertex set V and an undirected edge set E and a set of required edges(Tasks) , A fleet of identical vehicles, each of capacity Q, is based at a designated depot vertex D. There are server requirement. (1) each route should starts and end in the depot. (2) each required edge needed served at least once. (3) the total demand of one route should not exceed the vehicle capacity Q. we need to minimize the total cost as lower as possible with finish all the road which has the demand

The CARP problem has several applications. For example, it can be used in the express delivery optimization, or the street garbage collection system. the good solution for the CARP problem can cut down the a mount of cost and save a lot of resource.

1.1 Software

In this project, we use the python 3.7 with IDE Pycharm to solve this problem. and with the libraries of numpy, multiprocessing, random, and copy to help me solve this problem.

1.2 Algorithm

The core algorithm of this project is path-scanning with five way to random choose edge, and after that, I use the genetic algorithm to improve the initial solution better.

2. Methodology

2.1 Representation

NOTATIONS

- **C** : Cost of the two vertices
- **D** : Demand of the edge
- **R** : Route for a one task
- **Q** : Vehicles capacity

DATA STRUCTURE

- Solution(tuple with two component)
    - **Route** : the route of one task, for example: [(0, 1), (2, 5), (5,4)]
    - **Cost** : the cost of this task, for example: 314
- Given_Information:
    - **VERTICES** : the number of vertices
    - **DEPOT** : the depot vertex
    - **CAPACITY**: the max capacity of one route
    - **REQUIRED DEMAND**: the demand of each edge
- Graph
    - **EDGE_SET**: a dictionary the contain the edge set and its corresponding cost and demand
    - **DISTANCE_TABLE**: the minimum distance between two vertices
- Evolution Algorithm
    - **POPULATIONS**: a group of based solution
    - **TIME_LIMITED:** the limited time for the evolution procedure
    - **BEST_CHOICES**: the lowest cost solution in the populations

2.2 Architecture

I design the 4 python file, and this is their function and usage

- Graph.py
    - **Class: Graph**:
        - **make_graph** : make a undirected graph for given the edge and vertices information
        - **calculate_cost**: This method can calculate the cost of route.
    - **floyd**: The classical algorithm that can calculate the minimum distance of two vertices.
- Evolution.py
    - **Class EvoSolves**:
        - **evolution**: Implement the evolution algorithm
        - **crossover**: The part of evolution that take two solution to biological crossover and reproduction a child, which contain the genes of its parents.
        - **mutate**:
            - Similar to local search function, mutate a solution to give it a better result after it has been reproduction after crossover
            - there are five 4 different way to mutate
- Population.py
    - **sort_population**: To sort the order of the population depend on its cost.
    - **find_best**: Find the minimum cost of solution in the population.
    - **get_group_number**: Select a best size of the group depend on the average cost of the group, this is very useful because some low cost graph only need a few solution to do a evolution and can get better solution in the end.
- CARP_Solver.py

- **main**: use multiprocessor to handle the whole procedure
- **read_file**: read the instruction from command line and read the according file into a dictionary
- **init_population**: initial population for given the population number
- **path_scanning**: use the Path-scanning algorithm to solve the initial solution

2.3 Model design

Since the CARP is an NP-hard Problem. therefore, how to design a good heuristics is the essential way to solve this problem. The Genetic Algorithm (GA) is a metaheuristic inspired by the process of natural selection, John Holland introduced Genetic Algorithm (GA) in 1960 based on the concept of Darwin's theory of evolution[1]. for this algorithm, the ideal is to evolve the populations by the bio-evolution. we need a initial populations, the way to find a lots of different solutions is path-scanning, with the different to find the next edge that satisfy out requirements by greedy algorithm. so my model is design as follows

1. initial populations by path-scanning
2. crossover and mutate
3. selection the better solutions
4. repeat 2-3 steps

2.3. Detail of Algorithm

Algorithm 1 Floyed

this is a classical algorithm that be used to find the min distance for arbitrary two vertices

```
Input: dis: a distance list
output: dis: a distance list
---------------------------------------------------------
length <- length of dis
for i in length:
  for j in length:
    for k in length:
      if dis(i,j) > dis(j,k) + dis(k,j)
        dis(i, j) = dis(j,k) + dis(k,j)
    end
  end
end
retutn dis
```

Algorithm 2 Path-Scanning

The random decide the diversity of the path-scanning solution cause we want to a lot of different solution, so random operator is must. I use four ways to select the next edge and make it more diversity. In fact, there are 5 ways to do that, and this method is proposed by Golden et al[2]

1. Maximize the return cost from j to the depot
2. Minimize this return cost
3. Maximize the ratio C/D
4. Minimize the ratio C/D

```
Input:
  free_edge: the edge dict that contain all edge that has its demand
  cost_table: the minimum distance between two vertices
  capacity : the vehicle capacity
Ouput:
  solution: a tuple with route and cost
----------------------------------------------------------
route_list = list
cost = 0
while free_edge is not empty:
  route_list.append([])
  edges = get_the_edge_that_satisy_the_capacity_now()
  capa <- capacity
  while edges is not empty
    next_node = find_nearest_node(next_node)
    next_edge = random use four ways to select edge
    route_list[-1] <- next_edge
    vertices_dis <- cost_table(node_now next_node)
    cost <- cost + cost of next_edge + vertices_dis
    remove edge from edges, free_edge
    edges <- get_the_edge_that_satisy_the_capacity_now()
  end
end
return (route, cost)
```

Algorithm 3 Evolution

I have mention this algorithm in previous model design description. and I want to explain some key point of this algorithm I used.

1. I select the parent by Roulette algorithm that the lower cost solution has the higher possibility to become the parent. that make the genes of whole population are much better by time
2. the crossover keeps one part of one parent genes, for example, one parent has three routes to finish the problem, so we randomly choose one route entirely, and the left edge depend on the other parent. this means we can get the excellent genes from both two parents
3. if I only use crossover, the new child may have the limit possible for the better solution. In order to give the whole population more possible, mutation is necessary. so we choose three way to mutate the solution
   1. flip one edge
   2. swap two edge
   3. single insertion

   and if the solution after mutation are worse than previous, return the previous solution

4. I try to use the new child to replace the worse member of this population
5. the fitness is the cost of solution, the lower, the better.

```
Input:
  p : populations of solution
 Ouput:
  p : the populations after evolution
---------------------------------------------------------
limite_num <- 3/2 * populations size
whlie true:
  if time approach to limited time:
    break
  if populations size == limited_num:
    selection(p)
  pa, pb <- roulete_select()
  child1 <- crossover(pa,pb)
  child2 <- crossover(pb,pa)
  best_child = select(child1, child2)
  best_child = mutate(best_child)
  if fitness(best_child) < max(fitness(pa), fitness(pb))
    p.add(best_child)
return p
```

Algorithm 4 crossover

I do not show the pseudocode of that algorithm but just clarify the ideal of that.

the key point of crossover are follows

1. select one entire route(gene) from the first parent into child
2. remove the edge that contain in this gene from the free edge set
3. traverse the second parent. for those edge that in the free edge set, add it into child in order. that means we should try to keep the order of remind edge like the edge order in the second parent as much as possible, and the total demand of one route should keep not exceeding the capacity

3. Empirical Verification

I choose the dataset in the web site[3] that except for the data that carp platform supply and with my computer. the environment are follows

- CPU : Inter CORE i5 8th Gen
- Windows 10
- 4 processing
- Python 3.7

3.1 Performance

I test each dataset three times with different of random seed, and the time of dataset of 'egl' is 120 seconds, others is 60 seconds. I take he average cost to show the performance.

hyperparameters

The hyperparameters in my program includes the populations size. I discover that the different of dataset is that the size of solution. for example, the egl file usually has more than 1000 total cost, so the possibility of that are more than other dataset with lower cost, In this case, the populations size depends the size of steps we jump. sometimes we may not find the best solution easily, so we need to get much more populations to get the more possible to get better solution, but for the dataset with low cost, may be we need to jump with carefully, slowly steps. that is a key point, it decide the time distribution. time is limited, do not spend time on those function that has already convergence.

3.2 Result

because some data in the web site seems has error, so we only choose the several data that can run the solution

Dataset except the OJ platform

dataset	1	2	3	ave
val5D	666	662	664	664
val10C	499	498	495	497.333333333333
+ New				

Dataset in the OJ platform

dataset	1	2	3	ave
va7A	312	313	310	311.666666666667
egl-s1-A	5373	5401	5321	5365
gdb1	316	316	316	316
val1A	173	173	173	173
gdb10	275	275	275	275
egl-e1-A	3846	3825	3813	3828
val4A	428	428	432	429.333333333333
+ New				

3.3 Analysis

During to some algorithm mistakes, the result of path-scanning is not so good with small-scale problem. and the it can only guarantee to generate the large quantify different initial solution . the time for generate 1000 solutions of path-scanning is about 4 seconds since its complexity is O(t^2). And, the Genetic Algorithm's key point is to run out of time.

Although the initial solution is not very good compared with others. after the evolution of Genetic Algorithm. the best solutions is very good. there are still many aspect to improve it. one point is the local search. it can use more ways to do the mutate since I only implement 3 type of mutation. If there are more possible to mutate more different genes, the algorithm may leap to the local area. and another key point is to try to parallelize more proceedings because I fount that there are big variance for given different populations size in the Genetic Algorithm. It may lead to better solution.

3.4 Conclusion

In the conclusion, in this project, the evolutionary algorithm implementation is relative good compared to others algorithm. the result close to the best solution for most of the dataset. however the limitation of GA still existed, and the hyperparameters could be improved better for the later work.

4. References

[1]J. Sadeghi, S. Sadeghi and S. Niaki, "Optimizing a hybrid vendor-managed inventory and transportation problem with fuzzy demand: An improved particle swarm optimization algorithm", Information Sciences, vol. 272, pp. 126-144, 2014.

[2]B. Golden and R. Wong, "Capacitated arc routing problems", Networks, vol. 11, no. 3, pp. 305-315, 1981.

[3]"Capacitated Arc Routing Problem", Uv.es, 2018. [Online]. Available: https://www.uv.es/~belengue/carp.html?tdsourcetag=s_pctim_aiomsg. [Accessed: 22- Nov- 2018].