Project Carp Report

Zhaoxu Zhang 11611308

***Abstract*—This is the second project of class SUSTech CS303, Artificial Intelligence. This project requires us to implement an algorithm to solve the CARP problem, which is a suffient-researched filed. Each student's program will be upload to an online platform to fight aginst others'.**

I. Preliminaries

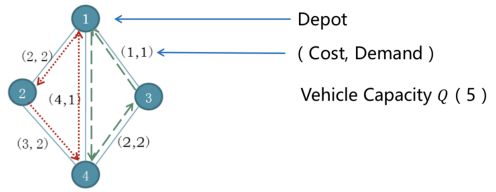
A. Problem Description

CARP(Capacitated Arc Routing Problem) can be described as follows: consider an undirected connected graph G = (V,E), with a vertex set V and an edge set E and a set of required edges(tasks) T $\subseteq$ E. A fleet of identical vehicles, each of capacity Q, is based at a designated depot vertex $v \in V$. Each edge $e \in E$ incurs a cost c(e) whenever a vehicle travels over it or serves it (if it is a task). Each required edge(task) $\tau \subseteq T$ has a demand $d(\tau) > 0$ associated with it.

The objective of CARP is to determine a set of routes for the vehicles to serve all the tasks with minimal costs while satisfying: a)Each routes must start and end at v_0; b)The total demand serviced on each must not exceed Q; c)Each task must be served exactly once(but the corresponding edge can be traversed more than once) This part describes the representation, the more detail of algorithm and the architecture I used in the codes.

B. Applications

The ARC routing problem is a classic problem with many applications in the world, such as urban waste collection, post deliver, sanding and salting the streets.CARP problem is a very classic example for ARC problem which is about urban waste collection. Since CARP is NP-hard [4], exact methods are only applicable to small-size instances. As a result, heuristics and meta- heuristics are often considered in the literature. For example, Augment–Merge, Path-Scanning, the "route first-cluster second" type heuristic proposed in, and Ulusoy's Heuristic are typical heuristic methods.[1]



C. Software

This project is written by python 3.6 using IDE pycharm. The library include numpy, random, functools, copy, operator, sys, time.

D. Algorithm

This project mainly use MEANs[1].

II. Methodology

A. Model Design

This problem is mainly solved by MEANS algorithm. Before that, I first implements Floyd algorithm to obtain the minCost matrix. MEANS algorithm is a modified genetic algorithm. Firstly, I use randomly path scaning to get a initial population of size 1000. And for each generation, I produce 5 children using crossover , and mutates according to the probability. For every generation, we maintain same population 1000 which are the best 1000 ones.

- CrossOver: Given two parent solutions S1 and S2, CrossOver randomly selects two routes R1 and R2 from them, respectively. Then, both R1 and R2 are further randomly split into two subroutes, say R1 = (R11, R12) and R2 = (R21, R22). After that, a new route is obtained by replacing R12 with R22.[1]
- localSearch: the single insertion, double insertion, and swap, are separately applied to the individual to conduct a best improvement local search. That is, for each move operator, all solutions that can be reached by it from Sx are examined, and the current solution will only be updated when a better solution has been found. After that, the MS operator is applied to this local optimal solution, forming the second stage of local search.
- How to judge a solution : here I make some simplification of the method provided by the paper as in my implementation, I removed all invalid solution(the one that exceeds capacity) The fomula is provided following:

$$value = \sum_{n=1}^{N} cost\,of\,route_i \tag{1}$$

B. Data and data structure

Here I mainly use dict and list in python.

C. Representation

To represent the chess in Gomoku , we define some rules:

- minCost: it is a matrix representing the min cost from one place to another place
- allDemands: it is a matrix representing the demand from one place to another place
- alldemandedge: it is a dict that contains all task as format "(a,b):demand"
- psize: size of initial population

- opsize: the amount of children of each generation
- ubtrial: the max time to find a solution each time
- P_{ls}: the probablity to perform a local search

D. Architecture

Here is some functions in the program :

- ***pathScaning***: It is the method that used to construct initial solution using path scaning method
- ***crossOver***: It is the method to achieve cross over between two parent solution
- ***MS***: It is the Merge-Split operator in MEANS algorithm
- ***localSearch***: It is the local search algorithm
- ***MEANS***: It is the main algorithm for MEANS.

E. Detail of Algorithm

Here describe the detail of algorithm in some vital functions. For some unimportant and well-known algorithm like "Floyd Algorithm","singleInsertion operator", they will not be presented below.

- MEANS: It is the main part of the MEANS algorithm.

function MEANS($psize, opsize, ubtrial, P_{ls}$)
  //Initialization.
  Set the current population $pop = \emptyset$
  while $|pop| < psize$ **do**
    Set the trial counter $ntrial = 0$;
    $ntial = ntrial + 1$
    while $S_{init} isnotacloneofanysolution S \in poporntrial = ubtrial$ **do**
      Generate an initial solution S_{init} ;
      $ntrial = ntrial + 1$
    end while
    if $S_{init} isacloneofanysolution S \in pop$ **then**
      break;
    end if
    $update\, his\_max\, and\, his\_values$
  end while
  $psize = |pop|$
  //Main Loop:
  while stopping criterion is not met **do**
    Set an intermediate population $pop_t = pop$
    for $ifrom1toopsize$ **do**
      Randomly select two different solutions S_1 and S_2 as the parents from pop;
      Apply the crossover operator to S_1 and S_2 to generate S_x ;
      Sample a random number r from the uniform distribution between 0 and 1;
      if $r < P_{ls}$ **then** Apply local search to S_x to generate S_{ls};
        if $S_{ls} isnotacloneofanyS \in pop_t$ **then**
          $pop_t = pop_t \cup Sls$
        else
          if $S_x isnotacloneofanyS \in pop_t$ **then**
            $pop_t = pop_t \cup Sls$
          end if
        end if
      else
        if $S_x isnotacloneofanyS \in pop_t$ **then**
          $pop_t = pop_t \cup Sls$
        end if
      end if
    end for
    Sort the solutions in popt using stochastic ranking;
    Set pop = the best psize solutions in popt ;
  end whilereturn the best feasible solution Sbf in pop;
end function

- pathScaning : we use it to generate a list of initial solutions

function PATHSCANING(method,alldemandege)
  tempt = copy os alldemandege
  while tempt's length != 0 **do**
    $\emptyset \Rightarrow routes$;
    capacity $\Rightarrow$ leftCap;
    depot $\Rightarrow$ start;
    while leftCap¿0 **do**
      IFINITY $\Rightarrow$ mincost;
      $\emptyset \Rightarrow closest$;
      for all task(a,b) in tempt **do**
        if minCost[start][a] == mincost **then**
          closet.add(task);
        end if
        if minCost[start][a] ¡ mincost **then**
          $\emptyset \Rightarrow closet$;
          closet.add(task);
          refresh mincost;
        end if
      end for
    end while
    for all task in closet **do**
      if it's load exceeds the leftCap **then**
        remove task from closet;
      end if
    end for
    if len(closest) == 0 **then**
      break;
    end if
    if len(closest) == 1 **then**
      result = closest[0];
    else
      if method == 1 **then**
        result = the task that maximize the distance to the depot;
      end if
      if method == 2 **then**
        result = the task that minimize the distance to the depot;
      end if
      if method == 3 **then**
        result = the task that maximize the term (load)/(serve cost);
      end if
      if method == 4 **then**
        result = the task that minimize the term (load)/(serve cost);

```
            end if
            if method == 5 then
                if current capacity¡= 0.5capacity then
                    use method1;
                else
                    use method2;
                end if
            end if
            if method == 6 then
                result = random choose one from closest;
            end if
        end if
        routes.append(result);
        remove result from tempt;
        finalresult.append(routes);
    end whilereturn finalresult
end function
```

- crossOver : In this method , we achieved crossover part of the algorithm

```
function CROSSOVER(S1, S2)
    R1⇐randomly choose one route from S1;
    R2⇐randomly choose one route from S2;
    R11,R12⇐randomly split R1;
    R21,R22⇐randomly split R2;
    R1loss⇐the one that is in R12 and not in R22;
    R22⇐remove the task in R22 that have appeared in R1 except R12;
    R1new⇐R11+R22;
    for task in R1loss do
        insert task into R1new at the position that will minimize total cost;
    end for
    if R1new is better than R1 then
        remove R1 from S1;
        insert R1new into S1;
    end ifreturn S1
end function
```

- localSearch : We achieve local search part in this method

```
function LOCALSEARCH(S_x)
    newsol1⇐using singleInsertion operator to generate a new solution that better or equal to Sx;
    newsol2⇐using doubleInsertion operator to generate a new solution that better or equal to Sx;
    newsol3⇐using swap operator to generate a new solution that better or equal to Sx;
    Sls⇐the best one of newsol1,newsol2,newsol3;
    Sls⇐using MS operator to generate a new solution that better or equal to Sx; return Sls;
end function
```

III. Empirical Verification

A. Dataset

Expect for the dataset provided by the carp platform provided, I also use some eglese dataset from reference of the textbook. Here is the name of the dataset: egl-e1-B.dat, egl-e1-C.dat, egl-e3-B.dat, egl-e3-C.dat

B. Hyperparameters

Here, the parameters that are needed to test is the input for MEANS function.

In the paper, the autho assigns following values to them:

psize	opsize	ubtrial	P_{ls}
30	180	50	0.3

However, I modify the parameter to the following. For the reason that our calculating resouce is limited and the most contributing part is Initialization, I increase the size of initial population and decrease the number of childrens for each generation.

psize	opsize	ubtrial	P_{ls}
1000	10	50	0.3

C. Performance Measurement

Here as this program always run until the end few second of the restricted time, it means nothing here to measure the program using time. Therefore, I use the total cost of the final solution to measure the performance of the program.

D. Experimental Results

Here is the performance of the dataset provided by the platform.

	egl-e1-A	val4A	gdb1	val7A	egl-s1-A
Cost	3785	412	316	285	5361

Here is the performance of the dataset from the reference.

	egl-e1-B.dat	egl-e1-C.dat	egl-e3-B.dat	egl-e3-C.dat
Cost	4662	6256	8246	10822

E. Conclusion

Actually, the performance is quite ok. As the size of initial population is large enough, it can get a result that closest to optimal solution at an early stage. However, it also has some defects. For example, it can not always find out the optimal solution. For this, I think it is might because that there are not enough generations as I didn't implements multithreads.

IV. Acknowledgements

I want to thanks for Doc.Tang ,Doc.Mei ,and Doc.Yao who created MEANS I also want to thanks for TA Yao Zhao who clarify my confusion about this project. At last I would like to thank forward to all the student assistances who share themselve's experience and maintain the online judge system.

References

[1] Tang, K., Mei, Y. and Yao, X. (2009). Memetic Algorithm With Extended Neighborhood Search for Capacitated Arc Routing Problems. IEEE Transactions on Evolutionary Computation, 13(5), pp.1151-1166.