CARP Solver

Peijun Ruan 11610214

School of Computer Science and Engineering
Southern University of Science and Technology
Email: 11610214@mail.sustc.edu.cn

1. Preliminaries

1.1. Introduction

The Capacitated Arc Routing Problem(CARP) problem is a classical combinatorial optimization problem.In this project, the problem can be described as: given a strong connected graph,each edge has a cost,and some of edges need to be serviced.Here are some number of vehicles to satisfy the demand which all start at a specific position(depot) and after finish its task, it will go back that position.what should be noticed is that each edge with demand should only be serviced once and each vehicle has a capacity to limit the demand it can satisfy.

So in this problem, there are three constraints:

- each vehicle should start from depot and go back depot at last.
- each demand edge should be serviced and serviced only for once.
- the total demand of edges a vehicle serviced should not exceed its capacity.

CARP has been proved to be a NP-hard problem and the time need to solve is $\mathcal{O}(n!)$, which means it is impossible to find the best solution in a polynomial time.Then the goal focus on: in given time, find a solution which satisfy all three constraints above and has minimum cost.

1.2. Software

This project is written in Python 3.6 as the guide document requires. I use Terminal to run and test this program and Pycharm to write and debug.The outer library I use in this project is Numpy and the built-in libraries I use include getopt, time, random, multiprocessing, sys and math.

1.3. Algorithm

In this project, I choose to use Dijkstra algorithm to build a two-dimension array for the graph and use Path-Scanning to build a original solution, then use Simulated Annealing[1] algorithm to change the original solution and formulate a new solution, comparing the new solution with the old one, if better then replace the old one with new one, else replace the old one with the new worse one with a certain possibility. After that ,repeat the solution formulation and replacement steps until the time is run out or the solution doesn't change for certain times, like 8 times, then return the final solution.

2. Methodology

In this section, I will show the method I use specifically and show how I represent the data in data structure.

2.1. Representation

Graph representation

- edge_cost: a two-dimension numpy array, store the cost from one vertex to another. If two vertices are not connected directly, the value will be 9999999.

- graph: a two-dimension numpy array, store the value of demand edge. If two vertex form a demand edge, the value should be larger than 0, if only form a edge, the value should be 0, if they are not connected directly, the value will be small than 0.

- min_cost: a two-dimension numpy array, store the minimum cost from one vertex to any other.

Answer representation

- solution: a list whose elements are lists, each vehicle's route are represent as a list consisted of tuples, and combine all the vehicles' list to a list. That's the solution of this project.

.

2.2. Functions

Here shows the main functions I use in the carp_solver.py

- get_input: get the input from Terminal and store the given data in variables.
- get_graph: read the data in given file and build the graph.
- dijkstra: find the shortest cost from one vertex to any others.
- get_min_cost: use dijkstra to generalize the min_cost array.
- path_scanning: use some strategy to find a valid original solution.
- fitness: calculate the cost of given solution.
- operators: the methods to create a new solution base on the original solution.
  - swap: exchange two elements.
  - insertion: choose one element in the solution and randomly insert it into one position.
  - MS[2]: the Merge-Split operator that use pathscanning to create a new solution.
- sub_thread: simulated annealing

2.3. Detail of Algorithm

Here show the psudocode of the main functions.

Algorithm 1 Dijkstra

Input: $vertex\ v$, $graph\ G$
Output: $one-dimension\ array\ min\_cost$

function DIJKSTRA(G , v)
$min\_cost \leftarrow [\ ]$, $min\_cost[v] \leftarrow 0$
  for $vertex\ s$ in G **do**
    $min\_cost \leftarrow G[v][s]$
  end for
  for i in range $vertices'\ number$ **do**
    $min \leftarrow 9999999$
    $p \leftarrow closest\ vertex\ to\ v\ that\ unvisited$
    $set\ p\ visited$
    for $vertex\ t$ in G **do**
      if p and t are directly connected **then**
        if $min\_cost[t] > min\_cost[p] + G[p][t]$ **then**
          $min\_cost[t] \leftarrow min\_cost[p] + G[p][t]$
        end if
      end if
    end for
  end for
  return min_cost
end function

Algorithm 2 Path_Scanning

Input: $Graph\ G, Edge\_cost\ E$
Output: $solution$

function PATH_SCANNING(G, E)
  $solution \leftarrow [\ ]$
  for i in range vehicles's number **do**
    $arcs[] \leftarrow free\ arcs$
    $route \leftarrow [\ ], pos \leftarrow 1, cap \leftarrow capacity$
    while $True$ **do**
      $d \leftarrow 9999999$
      for arc in arcs & $cap - G[arc] > 0$ **do**
        if $dis[pos][beg(arc)] < d$ **then**
          $d \leftarrow dis[pos][beg(arc)]$
          $u \leftarrow arc$
        else if $dis[pos][beg(arc)] = d$ **then**
          if $G[arc]/E[arc] > G[u]/E[u]$ **then**
            $u \leftarrow arc$
          end if
        end if
      end for
      if d doesn't change **then**
        break
      end if
      delete u and uîn $arcs$
      route.append(u),pos=beg(u),cap-=G[u]
    end while
    solution.append(route)
  end for
  return $solution$
end function

Algorithm 3 Sub_Thread

Input: $solution\ S$
Output: $best\ solution\ B$

function SUB_THREAD(S)
  $t \leftarrow fitness(S)$
  $best \leftarrow S$
  $local \leftarrow S$
  $temperature \leftarrow t^2$
  $break\_time \leftarrow 8\ times \leftarrow 0$
  while $temperature > 0.01$ **do**
    if $times >= break\_time$ **then**
      break
    end if
    for i in range(4) **do**
      new_solution $\leftarrow$ creat_new_route(local)
      a $\leftarrow$ fitness(new_solution)
      b $\leftarrow$ fitness(local)
      if $a < b$ **then**
        $local \leftarrow new\_solution$
      else if $exp((a - b)/temerature) > random(0.0, 1.0)$ **then**
        $local \leftarrow new\_solution$
      end if
    end for
    if $fitness(local) < fitness(best)$ **then**
      $best \leftarrow local$
    else
      $times \leftarrow times + 1$
    end if
    $temperature* = 0.85$
  end while
  return $best$
end function

Algorithm 4 MS

Input: $solution\ S$
Output: $a\ new\ solution\ N$

function MS(S)
  $p \leftarrow random\ integer$
  sub_route_set $\leftarrow$ select p route in S
  apply path_scanning on sub_route_set
  modify the solution S get N
  if $N\ is\ valid$ **then**
    return N
  else
    return S
  end if
end function

3. Empirical Verification

For this project, it is easy to do the empirical verification. Since we already have the test sets, we can use them and calculate the cost of the final solution the algorithm return and compare the final solution's quality with the original solution's quality to judge how much the solution has improved.

3.1. Design

To design test is pretty simple due to the only thing we need to do is the type right input in the Terminal and wait it give a final solution and its total cost

3.2. Data

The Data set I use in test is what teacher has offered on Sakai. Because the data set given by teacher is enough for testing , I did not find other data sets on Internet.

3.3. Performance

Teacher offers 7 data sets on this project and I will show the result on all of them.The test environment uses i5-7300HQ 2.50GHZ, 8GB RAM, and use all the 4 processing to run the code.

gdb1.dat RUNNING TIME: 60s RANDOM SEED: 2
s 0,(12,7),(7,8),(8,10),(10,11),(11,5),0,0,(1,12),(12,6), (6,7),(7,1),0,0,(1,4),(4,3),(3,2),(9,11),(11,8),0,0,(12,5), (5,3),(5,6),0,0,(1,2),(2,4),(2,9),(9,10),(10,1),0 q 316

gdb10.dat RUNNING TIME: 60s RANDOM SEED: 2
s 0,(1,10),(10,12),(10,9),(9,3),(3,4),(4,2),(2,1),0,0,(1,4), (4,6),(6,11),(11,12),(12,8),(8,1),0,0,(1,11),(11,4),(4,7), (7,2),(2,5),(5,1),0,0,(1,9),(9,8),(8,3),(3,2),(7,5),(5,6),0 q 275

val1A.dat RUNNING TIME: 60s RANDOM SEED: 2
s 0,(1,11),(11,6),(6,7),(7,8),(8,14),(14,13),(13,17), (17,18),(18,14),(13,7),(17,12),(12,6),(6,5),(5,11), (11,12),(12,16),(16,17),(16,15),(20,21),(21,24),(24,23), (23,22),0,0,(1,20),(20,15),(15,21),(21,23),(22,19), (19,20),(1,19),(19,9),(9,3),(3,10),(10,2),(2,4), (4,3),(9,1),(1,5),(5,4),(2,5),0 q 183

val4A.dat RUNNING TIME: 60s RANDOM SEED: 2
s 0,(2,8),(8,7),(8,9),(9,14),(14,13),(23,24),(24,25), (25,31),(32,26),(26,27),(27,28),(28,22),(22,21),(21,27), (27,33),(33,37),(27,32),(27,20),(20,21),(22,18),(18,17), (17,20),(20,19),(19,16),(16,15),0,0,(1,2),(2,3),(3,4), (4,5),(5,6),(6,12),(12,11),(11,17),(17,16),(16,11),(11,5), (4,10),(10,11),(10,15),(15,25),(25,26),(26,19),(15,14), (14,24),(24,30),(23,14),(10,9),0,0,(1,7),(7,13),(13,23), (23,29),(29,34),(34,38),(38,39),(39,40),(40,36),(36,39), (39,35),(35,36),(36,37),(37,41),(41,40),(36,32),(32,31), (31,35),(35,34),(31,30),(30,29),(9,3),0 q 428

val7A.dat RUNNING TIME: 60s RANDOM SEED: 2
s 0,(1,11),(11,16),(16,15),(15,9),(9,10),(10,16),(15,14), (14,8),(8,9),(10,1),(1,35),(35,36),(36,2),(2,6),(6,1),(1,40), (40,8),(8,1),0,0,(1,33),(33,26),(27,34),(34,35),(35,28), (28,27),(27,26),(26,34),(34,33),(11,12),(17,20),(20,21), (21,22),(22,25),(25,24),(24,23),(23,20),(21,24),(22,18), (18,17),(4,38),(38,39),0,0,(1,2),(2,3),(3,4),(4,5),(5,39), (39,32),(32,31),(31,30),(30,29),(29,36),(36,37),(37,38), (38,31),(31,37),(37,30),(37,3),(3,7),(7,13),(13,17),(17,12), (12,6),(6,7),(13,18),(18,19),(19,13),(13,12),0 q 304

egl-e1-A.dat RUNNING TIME: 60s RANDOM SEED: 2
s 0,(60,61),(60,62),(62,66),(66,68),(62,63),(63,65),(56,55), (41,35),(35,32),0,0,(44,46),(46,47),(47,48),(47,49),(51,21), (21,22),(22,75),(75,23),(23,31),(31,32),(32,34),(32,33),0,0, (1,2),(2,3),(2,4),(4,5),(9,10),(11,59),(59,69),(69,4),0,0, (11,12),(12,16),(16,13),(13,14),(15,17),(15,18),(18,19), (19,21),(51,49),(49,50),(50,52),(52,54),(19,20),(20,76),0,0, (69,58),(58,59),(59,44),(44,45),(44,43),(42,57),(57,58), (58,60),0 q 3839

egl-s1-A.dat RUNNING TIME: 60s RANDOM SEED: 2
s 0,(66,67),(67,68),(67,69),(69,71),(71,72),(72,73),(73,44), (44,45),(45,34),(44,43),(124,126),(126,130),0,0,(104,102), (66,62),(62,63),(63,64),(55,140),(140,49),(49,48),(139,33), (33,11),(139,34),0,0,(1,116),(116,117),(117,2),(117,119), (118,114),(114,113),(113,112),(112,107),(107,110),(110,111), (110,112),0,0,(107,108),(108,109),(107,106),(106,105),0,0, (64,65),(55,54),(11,8),(8,6),(6,5),(8,9),(13,14),(13,12), (12,11),0,0,(105,104),(95,96),(96,97),(97,98),(56,55),(11,27), (27,25),(25,24),(24,20),(20,22),(27,28),(28,30),(30,32), (28,29),0,0,(87,86),(86,85),(85,84),(84,82),(82,80),(80,79), (79,78),(78,77),(77,46),(46,43),(43,37),(37,36),(36,38), (38,39),(39,40),0 q 5428

3.4. Improvement

Now let's compare the performance improvement between original solution and the final solution.

	gdb1	gdb10	val1A	val4A
original	350	304	188	431
final	316	275	183	428
improvement	9.7%	9.5%	2.6%	0.69%

	val7A	egl-e1-A	egl-s1-A
original	326	4263	6382
final	304	3839	5428
improvement	6.7%	9.9%	14.9%

3.5. Analysis

From the result above we can find, for some small scale data set, my algorithm can easily find the optimal solution, but for these medium scale data set, my algorithm seems to be trapped at the local optimal and makes little improvement. But for the two large scale data set, my algorithm can improve a lot compared to the original solution. One reasonable explain is due to the scale of data set. The range of local search for better solution changes according to size, for medium scale data set, the range is small and my algorithm can not jump out the local optimal trap. While for the large size data set, the search space is very large so I can find a better solution.To draw a conclusion, the key problem in my algorithm is that it can not generate a solution to jump out of the local optimal trap.

References

[1] 李庆华, 李波. CARP问题的元启发式算法综述[J]. 2013.

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