CARP Project

Kai Lin 11610205
School of Computer Science and Engineering
Southern University of Science and Technology
Email: 11610205@mail.sustc.edu.cn

1. Preliminaries

For CARP problem , known as Capacitated Arc Route Problem, is an NP-hard problems in optimization. For carp, it envloves with the limit of capacity of each vehicle, which leads it to be more hard than traditional routing programming problem. The problem can be described as following: consider an undirected connected graph G = (V, E), with a vertex set V and edge set E and a set of required edges R E. A fleet of identical vehicles, each of capacity Q, is based at a designated depot vertex. Each edge of the graph (vi , vj) incurs a cost cij whenever a vehicle travels over it or services a required edge. When a vehicle travels over an edge without servicing it, this is referred to as deadheading. Each required edge of the graph (vi , vj) has a demand qij associated with it. A vehicle route must start and finish at the designated depot vertex and the total demand serviced on the route must not exceed the capacity of the vehicle.The aim is take the least cast do all demand in these rules.

1.1. Software

This project is implemented in Python using IDE PyCharm. The libraries being used includes Numpy , multiprocessing , math , getopt and Random.

1.2. Algorithm

Path scanning [1] is used in this code to generated initial population.For get the min cost for each point to point I use the Dijsktra to calculate one point to other point distance and do for all points. After get several initial solutions take each solutions to take Simulate Anneal for a better solution until time over.So the total algorithm is take Dijsktra first . Then get the min cost for Path scanning .Finaly , take the initial solutions to do Simulate Anneal for a better solution.

2. Methodology

For the NP hard problem is hard to find best or a good enough solution first . So the Mainstream method to detail the problem is find a good solution first and do other method like Tuba Search and Simulate Anneal to get a better and better solution . For CARP's solutions are like many mountains . That people in the mountains is hard to know weather he is in the highest mountain . The solving process is take several point in different mountains which like high enough mountains the most mountains . Then try the near mountains to find a higher and higher mountain[2].

2.1. Representation

Firstly , there are 7 global variables defined ahead of the file

- **vertices**: the number of vertices.
- **depot**: the depot vertex number .
- **required_num**: the number of required edges (tasks).
- **no_required_num**: the number of non-required edges .
- **sum_num**: required_num add no_required_num .
- **capacity**: the vehicle capacity .
- **need_work**: the total cost of all tasks .

Then the class Graph

- **Graph**: the class for main data need use .
  - node: a set for all node form 1 to vertices
  - edge: a list to remember all edge for the graph .
  - distances: a dict to remember all edge distance for the graph .
  - graph_cost: use a two dimensional array that [from][to] is cost .
  - graph_work: use a two dimensional array that [from][to] is demand.

Other use data in the algorithm

- **file_name**:the file name get from command line .
- **time_limit**:the time limit get from command line .
- **random_seed**:the random seed get from command line that set in random make the same random seed get same random result .
- **min_cost**:a two dimensional array remember [from][to] min distance .
- **path**:a list in list first is the car No and the path[i] is the i car go path (do work) .

2.2. Architecture

Here are the important functions in Python file CARP_solver with pesudocode format.

- **dijsktra**: get one nodes to each other node min distance(cost) return a dict(to , distance).
- **all_dij**: get all nodes min distance(cost) to other return a two dimensional array.
- **go_one_car**: take one car to do work until all work is done or car use all capacity.
  - **rule**: when get more then one choice how to select .
    1. maximize the ratio of the edge cost and demand .
    2. minimize the ratio of the edge cost and demand .
    3. maximize the return cost .
    4. minimize the return cost .
- **go_mountain_SA**: take one solution to other possible and choice if use this new solution .
- **swap_one**: take one task to another point in task list.
- **swap_two**: take two together task to another point in task list.
- **swap_self**: take one task (from , to) to (to , from) .

2.3. Detail of Algorithm

Here show the psudocode main use that dijsktra , all_dij , go_one_car and go_mountain_SA .

Algorithm 1 dijsktra:get one nodes to each other node min distance(cost) return a dict(to , distance).

```
Input: $Graph$ graph_get,$int$ begain_point
Output: $dict$ visited
function DIJSKTRA($graph\_get, point$)
    $visited \leftarrow \{begin\_point, 0\}$
    while $nodes$ do
        for node in nodes do
            if visited[node] ¡ visited[min_node] then
                min_node = node
            end if
        end for
        nodes.remove (min_node)
        for $i = 1 \rightarrow vertices + 1$ do
            if graph_get[min_node][i]¿0 then
                refresh or add visited[i] to the
                min distance
            end if
        end for
    end while
    return $visited$
end function
```

Algorithm 2 all_dij: get all nodes min distance(cost) to other return a two dimensional array.

```
Input: $Graph$ graph_get
Output: $array$ min_cost
function ALL_DIJ($graph\_get$) $min\_cost \leftarrow [\ ][\ ]$
    for $i = 1 \rightarrow vertices + 1$ do
        $visited \leftarrow dijsktra(graph\_get, i)$
        for $j = 1 \rightarrow vertices + 1$ do
            min_cost[i][j] = visited.get(j)
        end for
    end for
    return $min\_cost$
end function
```

Algorithm 3 go_one_car: use one car to do work until no work can do.

```
Input: $Graph$ graph_get,$numpy$ min_cost,$int$ need_work
Output: $int$ cost,$list$ path,$int$ need_work
function GO_ONE_CAR($graph\_get, min\_cost, need\_work$)
    $cap \leftarrow capacity$
    $cost \leftarrow 0$
    $first \leftarrow depot$
    $choice\_list \leftarrow [\ ]$
    while $cap > 0 and need\_work > 0$ do
        for each work in graph_get.work_dict do
            if $work <= cap$ then
                $choice\_list.append(work.from, work.to)$
            end if
        end for
        if len(choice_list) == 0 then
            $break$
        end if
        if len(choice_list) == 1 then
            $work = choice\_list[0]$
            $cost+ = work\_cost$
            $cost+ = distance(first, work\_begin)$
            $need\_work- = work\_cost$
            $path.append(work\_begin, work\_end)$
            $first = work\_end$
        end if
        if len(choice_list) ¿ 1 then
            take work by random rule in 1-4
            $cost+ = work\_cost$
            $cost+ = distance(first, work\_begin)$
            $need\_work- = work\_cost$
            $path.append(work\_begin, work\_end)$
            $first = work\_end$
        end if
    end while
    return $cost, path, need\_work$
end function
```

Algorithm 4 go_mountain_SA :take one solution to other possible and choice if use this new solution .

```
Input: $Graph$ graph_get,$list$ path, $numpy$ min_cost,$int$ cost
Output: $int$best_cost,$list$best_path
function GO_MOUNTAIN_SA($graph\_get, path, min\_cost, cost$)
$T \leftarrow cost * cost$
    while $T > 0.15 * costortimeover$ do
        for $i = 0 \to 4$ do
            for all path do
                try swap_one
                if $new\_cost < cost$ then
                    end_path=new_path
                    end_cost= new_cost
                    cost=new_cost
                    path =new_cost
                end if
                if $new\_cost > cost$ then
                    flag=random.random()
                    number = -(new_cost-end_cost)/T
                    fire = math .exp(number)
                    if flag¡fire then
                        cost=new_cost
                        path =new_cost
                    end if
                end if
                try swap_two and swap_self like
                the swap_one
            end for
        end for
    end while
    return $end\_cost, end\_path$
end function
```

for the total algorithm is do the flow like 1.read data and save . 2 .use path scanning(go_one_car)to get processing number initial solution . 3. use SA to get better and better solution until T is so low or time over.

3. Empirical Verification

Empirical verification is compared with keep the same level of the result of path scanning try initial solution . The reason might be the trap of local minimal but path scanning not conduct efficient mutations to conquer this problem.After use SA always can get a better solution then the initial but also will take in a new trap that is like I say in the mountains we use SA may can go out one trap but may we find a more higher mountain and the near mountains are much lower then it or the temperature is so low that do not tolerate the higher cost and in the trap can not go out.

3.1. Design

For this project we can simple take this problem to two parts 1. get the initial solution. 2.optimize the initial. For part.1 I use path scanning we get one more choice I take a random rule in the four I give out to select. For part.2 I use SA to optimize that take initial T in cost*cost and do small in each 4 loop and take T to T*0.9 do the SA until time over or T is low enough finally take out the best soltion.

3.2. Data and data structure

Data used in this project is the sakai given that egl-e1-A.dat , egl-s1-A.dat , gdb1.dat , gdb10.dat , val1A.dat , val4A.dat and val7A.dat . Data structures has been given in the CARP_format.txt.For other data set use my algorithm test is all right .

3.3. Performance

About the path scanning time complex is about $\mathcal{O}(n^3)$.and the SA time complex is about $\mathcal{O}(n^3)$. The alogrithm will use 8 processe one is try path scanning and 7 is use different initial solutins to do the SA for better solution .Do these 8 processes until time over and print out the end best solution.

3.4. Result

the result of egl-e1-A.dat

```
s 0,(1,2),(2,3),(2,4),(4,69),(69,59),(59,11),(11,12),(76,20),(20,19),
(50,52),(51,21),0,0,(4,5),(9,10),(12,16),(16,13),(13,14),(15,17),(15,18),
(18,19),(19,21),(52,54),(47,46),0,0,(69,58),(58,60),(60,62),(62,66),(66,68)
(62,63),(63,65),(60,61),(58,59),(59,44),(44,45),0,0,(58,57),(57,42),(41,35)
(35,32),(32,31),(31,23),(51,49),0,0,(44,46),(44,43),(56,55),(32,34),(32,33)
(23,75),(75,22),(22,21),(49,50),(49,47),(47,48),0
q 3874
```

the result of egl-s1-A.dat

```
s 0,(1,116),(116,117),(117,2),(117,119),(118,114),(114,113),(112,110),
(110,107),(107,106),(105,104),(110,111),0,0,(113,112),(112,107),
(107,108),(108,109),(106,105),(104,102),(95,96),(96,97),(97,98),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),0,0,(124,126),(126,130),
(68,67),(67,66),(66,62),(62,63),(63,64),(64,65),(56,55),(55,140),(140,49),
(49,48),0,0,(67,69),(69,71),(71,72),(72,73),(73,44),(44,45),(34,139),
(139,33),(33,11),(11,12),(12,13),0,0,(43,44),(45,34),(11,27),(27,28),
(28,30),(30,32),(28,29),(27,25),(25,24),(24,20),(20,22),0,0,(55,54),(11,8),
(8,6),(6,5),(8,9),(13,14),0
q 5691
```

the result of gdb1.dat

```
s 0,(1,4),(4,2),(2,3),(3,5),(5,6),0,0,(1,2),(2,9),(9,11),(11,5),(5,12),0,0,
(1,7),(7,8),(8,10),(10,11),(11,8),0,0,(1,10),(10,9),(4,3),(6,7),(7,12),0,0,
(1,12),(12,6),0
q 316
```

the result of gdb10.dat

```
s 0,(1,11),(11,6),(6,5),(5,2),(2,4),(4,1),0,0,(1,8),(8,9),(9,3),(3,4),(4,7),
(7,2),(2,1),0,0,(1,5),(5,7),(2,3),(3,8),(8,12),(12,10),(10,1),0,0,(1,9),
(9,10),(12,11),(11,4),(4,6),0
q 275
```

the result of val1A.dat

```
s 0,(1,5),(5,4),(4,3),(3,10),(10,2),(2,4),(2,5),(5,6),(6,12),(12,17),(17,18),
(18,14),(14,8),(8,7),(7,6),(6,11),(11,5),(1,9),(9,3),(9,19),(19,1),0,0,
(1,11),(11,12),(12,16),(16,17),(17,13),(13,7),(13,14),(16,15),(15,20),
(20,19),(19,22),(22,23),(23,21),(21,15),(20,21),(21,24),(24,23),(20,1),0
q 173
```

the result of val4A.dat

```
s 0,(1,2),(2,3),(3,4),(4,5),(5,6),(6,12),(12,11),(11,16),(16,19),(19,26),
(26,32),(32,36),(36,35),(35,39),(39,38),(38,34),(34,35),(35,31),(31,30),
(30,24),(24,14),(14,13),(7,8),(9,3),0,0,(1,7),(7,13),(13,23),(23,29),(29,30),
(29,34),(31,32),(32,27),(27,20),(20,17),(17,18),(18,22),(22,28),(28,27),
(27,21),(21,20),(20,19),(16,17),(17,11),(11,5),(4,10),(10,9),(9,14),(15,10),
0,0,(2,8),(8,9),(10,11),(15,16),(15,25),(25,26),(26,27),(27,33),(33,37),
(37,41),(41,40),(40,36),(36,39),(39,40),(36,37),(21,22),(31,25),(25,24),
(24,23),(23,14),(14,15),0
q 418
```

the result of val7A.dat

```
s 0,(1,35),(35,36),(36,2),(2,3),(3,4),(4,5),(5,39),(39,32),(32,31),
(31,30),(30,29),(29,36),(36,37),(37,31),(31,38),(38,4),(3,37),
(37,30),(37,38),(38,39),(3,7),(7,13),(13,12),(12,6),(6,1),0,0,
(1,40),(40,8),(8,9),(9,15),(15,16),(16,11),(11,12),(12,17),
(17,13),(13,18),(18,22),(22,21),(21,20),(20,23),(23,24),(24,25),
(25,22),(21,24),(20,17),(17,18),(18,19),(19,13),(7,6),(6,2),
(2,1),0,0,(1,33),(33,26),(26,27),(27,34),(34,35),(35,28),(28,27),
(34,33),(34,26),(1,11),(1,10),(10,9),(10,16),(15,14),(14,8),(8,1),0
q 291
```

3.5. Analysis

This CARP project is not like the go_bang that can just use one person self though that can get a good result. For the NP hard problem has been study for many smart person like Xin Yao , so I think this project is main teach me how to study and use the knowledge that other has find and standing on the shoulders of giants to jump for a new high .Maybe my algorithm may also can not go out some trap but I study lot and try my best is enough .

Acknowledgments

I would like to thank my friends Peijun Ruan and Pujin Chen who together discuss the CARP's frame . Read and discuss the parper which the hard to understand part .Last I would like to thank SA who will check my codes and reports.

References

[1] Mei, Y. (2010). . [online] Wanfangdata.com.cn. Available at: http://www.wanfangdata.com.cn/details/detail.do?_type=degreeid=Y1705973 [Accessed 22 Nov. 2018].

[2] Ke, T., Yi, M. and Xin, Y. (2009). Memetic Algorithm With Extended Neighborhood Search for Capacitated Arc Routing Problems. IEEE Transactions on Evolutionary Computation, 13(5), pp.1151-1166.