SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Project for CARP Solver

Yuxing Hu
11510225

November 26, 2017

CONTENTS

1 PRELIMINARIES

1.1 INTRODUCTION

The CAPR solving problem is basically an optimize routes analysis. Suppose we have a graph that is fully connected, and it have several demanded edges for us to go through. And there is also some edges are not demanded, just staying there for us to use as a path to certain vertices and demanded edges. The goal is focus on cost: every edge has a cost, and the cost are different in different edges. Every time we go though an edge, no matter that edge is demand or not, we will cost some value then the total cost is increased. So we are not only to pass all the path, but also to minimize the cost of total path.

And there is also a big differ between CARP and ARP: capacitated. Every time we pass over some path, suppose we have a capacity n, when the total demand of this trip comes to capacity, we must come back to initial point called depot to " get some supply in order to keep going. This constrain makes whole thing more trickier: the edges we needs to go through definitely become bigger, since the path amount is huge, we need to think deeper about how to gets over it.

1.2 SOFTWARE

The project is written in Python 2.7, as the guide document requires. I use Terminal to run and test the program and Sublime Text to write in. And I also use PyCharm to debug the program and see the structure of the project. The only outer library being used is NumPy, as for inner library built in Python I used including getopt, time, random, multiprocessing, functools and sys.

2 METHODOLOGY

In this section, I will go through mainly method I used in this project. The reference book that teaching assistant Yao Zhao gave me a lot of help[1], especially in Chapter 7. Also in order to be more familiar with the whole problem, I also read some different article and journalist [2] [3] to find more detail and get inspired.

2.1 GRAPH CLASS

This class is designed to present a graph type for Dijkstra algorithm to use. And in the initial part we can notice that this graph is self organized from NumPy array, which is a different type of data structure. Initialization method will create vertices, edges and the value of each edges existed in the graph.

2.2 MAIN METHOD

In the main method part we manage to set global parameter like start time, terminate time and file name. Also, I store the input variables like random seed. To maximum the solution, I

use all 8 threads and call the multiprocessing from Python inner library. To share the result, I use the Manager() method to store data from different threads.

2.3 Set Globals

This method is used to set the globals in the whole program and for the future use. Also, to cast the data from initial file into NumPy array for calculation, including capacity, edges, vertices, cost, demand, depot and so on.

2.4 Solver Method

First the solver program will deal with some global parameters, and then calculate the cost of every two vertices and store them into a NumPy array list called path cost. Then is the main path scanning method, in this part the program will deal with different situation and calculate the right vertex to go, and the next vertex it should go. Every time a demand path is passed, the remain list which stored all the edges it supposed to pass will delete the target.

Scanning method will be shut down if all the demanded edge is gone through. And it will go through several times since I wrote many type of method to face the different situation. After every way has been went through or the deadline time is about to come, every possible path and related cost will be store in a dictionary structure and sort itself to find the smallest cost and return it with the path to main method to print.

2.5 Cal Dijk

This method is used to cast the NumPy array which store all the data in to readable type to create graph and for Dijkstra method to use. The order in this NumPy array is row by row, and that makes a v*v matrix.

2.6 Dijsktra

Dijsktra algorithm is used to calculate the minimum distance between every single two vertices' distance. The algorithm can be described as $Algorithm1$.

2.7 Prior-key Path Scanning

I wrote several tactics to find the best path, so then I will show them one by one. Path scanning is the main progress in this project, the overall structure is firstly find whether some of the demanded edge is started from current vertex. If so, the next step should be choose one of the path to go on. Because there are several ways to choose them, the mainly strategy are written as follow. And the algorithm are described as $Algorithm2$.

If there aren't any demanded path starting from current node, then the program will call for the path cost that Dijkstra algorithm created to find out which of the node is both nearest to current node(car's current position) and also is one of the end point of the demand node.

Algorithm 1 Dijkstra algorithm

Input: The graph created by class graph, G;

Output: The visited vertex and edge from the starting vertex, V; The distance from starting vertex to every other node,P

```
1:  procedure DIJKSTRA PROCEDURE
2:      create vertex set Q
3:      for each vertex v ∈ G do
4:          dist v = INFINITY
5:          prev v = UNDEFINED
6:          add v to Q
7:      end for
8:      dist[source] ← 0
9:      while Q is not empty do
10:         u ← min dist vertex in Q
11:         remove u from Q
12:         for each neighbor v of u do
13:             alt ← dist[u] + length(u,v)
14:             if string(i) = path(j) then
15:                 dist[v] ← alt.
16:                 prev[v] ← u;
17:             end ifreturn dist[], prev[];
18:         end for
19:     end while
20: end procedure
```

Algorithm 2 Path Scanning

Input: The remain edges list from the road graph, $remain\_list$; The position of the car,$car\_position$; The remain capacity of the car,car_cap; The cost matrix created by Dijkstra algorithm, $path\_cost$; All undirected edges,$graph\_edge$

Output: The visited path in this car journey, $path$; The cost of this path,$cost$

procedure PATH SCANNING PROCEDURE
while $remain$ is not empty **do**
create at_dic
for each edge$i$ in $remain\_list$ **do**
if one of the end of i =$car\_position$ **then**
add edge i into at_dic
end if
end for
if size of at_dic != 0 **then**
path choose function
car_position←selected_path
$cost$ += $path\_cost$ of $selected_path$h
car_cap -= $selected\_path$;
$path$ += $selected\_path$
end if
if size of at_dic = 0 **then**
create $passible\_path$
for each edge$i$ in $remain\_list$ **do**
if one of the end of i =$car\_position$; demand of that edge <= car_cap **then**
add edge i into $passible\_path$
end if
end for
if $passible\_path$ is not empty **then**
arrange $remain\_list$ by $path\_cost$ ascending
car_position←remain_list[0]
$cost$ += $path\_cost$ of $remain\_list$[0]
end if
if $passible\_path$ is empty **then**
car_cap← $full$;
car_position← $depot$
end if
end if
end while
end procedure

If these two requirements is satisfied, it will find out whether there is enough capacity to go through the demand path. Hopefully it can reach there and keep going, but if not, it will return to the starting point to full up(get some gas), and come back at the starting point.

2.7.1 PRIOR-KEY NEAR FIRST

This strategy allow me to choose the nearest, minimum cost depot to be next destination point.

2.7.2 PRIOR-KEY FAR FIRST

This strategy allow me to choose the maximum cost depot to be next destination point.

2.7.3 PRIOR-KEY HALF HALF

If the remain capacity is less than 50 percent, it will choose nearest, otherwise furthest.

2.7.4 PRIOR-KEY HALF HALF2

Same as half half, but opposite, more than 50 percent, it will choose nearest, otherwise furthest.

2.7.5 PRIOR-KEY RANDOM

Choose a random as the destination. This progress will repeated several times until time is about going to end. So the mainly time of program used is on this part.

2.8 ADD DES LIST

This method is used to determine the list that can be go through from current point, which means can be instantly went through from current point.

2.9 CHOOSE PATH

This method is used to determine which vertex to go next. It went through all the path that is in the remain edge list. For different way, it will return different end vertex for further going on.

2.10 GO SITUATION

This method is used to choose next step destination. If capacity of current car is about to be minus, it should set status to FALSE so then the car will go back to initial point to have next round.

2.11 RESET ALL

This method is used to reset all the initial value for next path scanning.

2.12 Sort Dic

This method is used to sort the result dictionary called pc list to find the minimize cost with its related path for main method to print.

3 Empirical Verification

The empirical verification is not easy because we are only given limited test example to test and to find the best path to go. However, on the other hand the test become easy and obvious to find because of all that matters to the solution is the cost that we used.

3.1 Design

The design of the test is pretty simple because of that we are suppose to type in the right value of input and the input format is fixed and should not be modified. So by calculating the time and the output of cost we can see how well is program designed.

3.2 Related Data

The data I use is what the teacher has offered. Although I find the origin data in the website and there are huge mount of data chart I could use, I didn't choose them because these 7 data charts we already had are representative and reasonable.

3.3 Performance Measurement

I use the time() function to measure the time we use for each of the path scanning function, and allow them to report the cost to test the best result they can get.

3.4 Results

Needs to point out all the results are under the circumstance of 60 seconds.

3.4.1 gdb1.dat

lifesavermac:desktop lifesaver python CARP_solver.py gdb1.dat -t 60 -s 1

---

s 0,(1, 7),(7, 6),(6, 12),(12, 7),(7, 8),0,0,(1, 10),(10, 8),(8, 11),(11, 9),(9, 2),0,0,(1, 4),(4, 3),(3, 5),(5, 6),(1, 2),0,0,(1, 12),(12, 5),(5, 11),(11, 10),(10, 9),0,0,(4, 2),(2, 3),0
q 299

3.4.2 gdb10.dat

lifesavermac:desktop lifesaver python CARP_solver.py gdb10.dat -t 60 -s 1

---

s 0,(1, 4),(4, 6),(6, 11),(11, 4),(4, 7),(7, 2),(2, 1),0,0,(1, 10),(10, 9),(9, 3),(3, 4),(4, 2),(2, 5),(5,

7),0,0,(1, 5),(5, 6),(12, 11),(11, 1),(1, 8),(8, 3),(3, 2),0,0,(1, 9),(9, 8),(8, 12),(12, 10),0
q 279

3.4.3 VAL1A.DAT

lifesavermac:desktop lifesaver python CARP_solver.py val1A.dat -t 60 -s 1

---

s 0,(1, 11),(11, 12),(12, 16),(16, 15),(15, 20),(20, 19),(19, 1),(1, 5),(5, 6),(6, 7),(7, 8),(8, 14),(14, 18),(18, 17),(17, 16),(13, 17),(17, 12),(12, 6),(6, 11),(11, 5),(5, 4),(4, 2),(2, 10),(10, 3),0,0,(1, 20),(20, 21),(21, 24),(24, 23),(23, 22),(22, 19),(19, 9),(9, 3),(3, 4),(5, 2),(9, 1),(23, 21),(21, 15),(7, 13),(13, 14),0
q 185

3.4.4 VAL4A.DAT

lifesavermac:desktop lifesaver python CARP_solver.py val4A.dat -t 60 -s 1

---

s 0,(1, 7),(7, 13),(13, 14),(14, 15),(15, 16),(16, 19),(19, 20),(20, 21),(21, 27),(27, 26),(26, 32),(32, 36),(36, 35),(35, 39),(39, 40),(40, 36),(36, 37),(37, 41),(41, 40),(36, 39),(39, 38),(38, 34),(34, 29),(3, 9),0,0,(1, 2),(2, 3),(3, 4),(4, 10),(10, 9),(9, 14),(14, 24),(24, 30),(30, 31),(31, 35),(35, 34),(32, 31),(31, 25),(25, 26),(26, 19),(11, 10),(10, 15),(15, 25),(25, 24),(24, 23),(23, 29),(29, 30),(14, 23),(23, 13),(8, 7),(9, 8),(5, 4),0,0,(8, 2),(11, 17),(17, 16),(16, 11),(11, 5),(5, 6),(6, 12),(12, 11),(20, 17),(17, 18),(18, 22),(22, 21),(32, 27),(27, 33),(33, 37),(20, 27),(27, 28),(28, 22),0
q 456

3.4.5 VAL7A.DAT

lifesavermac:desktop lifesaver python CARP_solver.py val7A.dat -t 60 -s 1

---

s 0,(1, 10),(10, 16),(16, 15),(15, 14),(14, 8),(8, 1),(1, 6),(6, 7),(7, 3),(3, 2),(2, 36),(36, 29),(29, 30),(30, 37),(37, 38),(38, 31),(31, 37),(37, 36),(36, 35),(35, 28),(28, 27),(27, 26),0,0,(1, 40),(40, 8),(8, 9),(9, 10),(15, 9),(11, 1),(1, 33),(33, 26),(26, 34),(34, 27),(33, 34),(34, 35),(35, 1),(1, 2),(2, 6),(6, 12),(12, 17),(17, 20),(20, 23),(23, 24),(24, 25),(25, 22),(22, 21),(21, 20),0,0,(16, 11),(11, 12),(12, 13),(13, 19),(19, 18),(18, 13),(13, 17),(17, 18),(18, 22),(24, 21),(7, 13),(4, 38),(38, 39),(39, 32),(32, 31),(31, 30),(3, 4),(4, 5),(5, 39),(37, 3),0
q 332

3.4.6 EGL-E1-A.DAT

lifesavermac:desktop lifesaver python CARP_solver.py egl-e1-A.dat -t 60 -s 1

---

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

60),(66, 62),(55, 56),(35, 41),(33, 32),(32, 34),0
q 4085

3.4.7 EGL-S1-A.DAT

lifesavermac:desktop lifesaver python CARP_solver.py egl-s1-A.dat -t 60 -s 1

---

s 0,(1, 116),(116, 117),(117, 2),(119, 117),(114, 118),(86, 85),(85, 84),(84, 82),(82, 80),(80, 79),(87, 86),0,0,(113, 112),(112, 110),(110, 107),(107, 112),(114, 113),(111, 110),(108, 109),(107, 106),(106, 105),(105, 104),0,0,(108, 107),(102, 104),(67, 66),(66, 62),(62, 63),(63, 64),(64, 65),(55, 56),(54, 55),(55, 140),(140, 49),(49, 48),0,0,(126, 124),(130, 126),(78, 77),(77, 46),(46, 43),(43, 37),(37, 36),(36, 38),(38, 39),(39, 40),(44, 43),(79, 78),0,0,(67, 69),(69, 71),(71, 72),(72, 73),(73, 44),(44, 45),(45, 34),(34, 139),(139, 33),(33, 11),(11, 12),(28, 27),0,0,(67, 68),(96, 97),(97, 98),(95, 96),(8, 6),(6, 5),(9, 8),(8, 11),(11, 27),0,0,(13, 12),(14, 13),(22, 20),(20, 24),(24, 25),(25, 27),(29, 28),(28, 30),(30, 32),0
q 5800

3.5 FURTHER THINKING

From the result we can observe that the final result is coming from the random path way give us the best solution. However, that is not the best method we should use. Because we know that random aren't the best, that is only the art of compromise. The theoretical lower bound is much smaller than what we just found. As for me, I think the next breakthrough point of CARP problem should be in math prove rather than spending time and resource to get.

REFERENCES

[1] Ángel Corberán and Gilbert Laporte. *Arc routing: problems, methods, and applications*. SIAM, 2013.

[2] Bruce L Golden and Richard T Wong. Capacitated arc routing problems. *Networks*, 11(3):305–315, 1981.

[3] Paolo Toth and Daniele Vigo. *The vehicle routing problem*. SIAM, 2002.