CARP Project

Yilin Zheng 11510506
School of Computer Science and Engineering
Southern University of Science and Technology
Email: 11510506@mail.sustc.edu.cn

1. Preliminaries

1.1. Software

This project is written in Python using editor Sublime Text and IDE PyCharm. The libraries being used includes NumPy, copy, re, getopt, etc.

1.2. Algorithm

Path scanning [1] is used in this code to generated initial population. Besides, Dijsktra is used to calculated the shortest distance betweent any two vertices. I also define three move operator to optimal the result, however, for some unknown reason they aren't work correctly. So I just put them in the source file.

2. Methodology

Before go into code detail, here we should go an overview about this NP hard problem CARP and some related work done by others. CARP, known as Capacitated Arc Route Problem, is an NP-hard problems in optimization. Arc routing problems arise in a wide variety of applications, including snow plowing, refuse collection, maintenance, and postal services. 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 $R \in E$. A fleet of identical vehicles, each of capacity Q, is based at a designated depot vertex. Each edge of the graph (v_i, v_j) incurs a cost c_{ij} 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 (v_i, v_j) has a demand q_{ij} 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, Q. [2] So far, many algorithms had been done on this problem, most of them can be divided into three types [1]

- Exact algorithm
- Heuristic methods
- Metaheuristic methods

For thie problem, exact algorithm does worse when the data go on to large scale while the heuristic methods mostly can give a more feasible solution. So most algorithms about solving this problem are heuristic based like GA, MA and some Greedy based modified algorithms.

2.1. Representation

Here we begins to talk the representation on the source code. There are 10 global variables defined ahead of the file:

- ARC: arc set including required and non-required edges
- AD_MATRIX: adjacent matrix
- AD_ARRAY: adjacent array using nested dict
- COST: cost record for each edge using nested dict
- DEMAND: demand record for required edges in nested dict
- REARC: requrie arc set
- NODE: node set
- SHORTEST_DIS: adjacent martix recording the calculated shortest paths between given two vertices
- REVERTICES: required vertices adjacent to required edges

Also the head of file contains some data which will be assigned value after data is read by function *read_file*.

- NAME: file name
- VERTICES: vertices number
- DEPOT: source and destination of all routes
- REQUIRED_EDGES: required edge number
- NON_REQUIRED_EDGES: non required edge number
- VEHICLES: vehicle number
- CAPACITY: capactity of each vehicle
- TOTAL_COST_OF_REQUIRED_EDGES: sum of the cost of required edges, which is also the service demand.

Some global variables are used to mark some status:

- CONNECTED: status of two vertices, 1 for connected but 0 for disconnected
- INFTY: max int as infinity
- NINFTY: negative max int

- UNDEFINED: undefined
- POPULATION_SIZE: population size
- ALPAH: rate for inverse arc
- SEED: seed of random

2.2. Architecture

Here list all functions in the Python file CARP_solver with their usage. Three main functions:

- **Dijkstra**: calculate the shortest path from source to destination
- ***RPSH***: random path scanning different from traditional path scanning, which is based on randomly choose better rule in the better function.
- **better**: five rules for choosing edges adding to route when facing the same cost. In the this function, five common rules are used [1]
    - 1. maximize the ratio c_{ij}/r_{ij}, c_{ij} being the cost of $[i, j]$ and r_{ij} being the remaining demand once $[i, j]$ is treated.
    - 2. minimize the ratio c_{ij}/r_{ij}
    - 3. maximize the return cost from j to the depot
    - 4. minimize the return cost
    - 5. Apply rule 3 if the vehicle is less than half full while apply rule 4

Some optimal functions doesn't work for result:

- SI: Single insertion
    - displace task u after task v (also before task v if v is the first task of $rt(v)$); both arcs of u are con- sidered when inserting u in the target position, and the one yielding the best solution is selected
- DI: Double insertion
    - displace a sequence (u, x) after task v (also before task v if v is the first task of $rt(v)$); similar to **SI**, both directions are considered for each task and the resulting best move is chosen
- SW: Swap
    - exchange task u and task v ; similar to **SI**, both directions are considered for each task to be swapped and the resulting best move is chosen

Also the source codes includes some other codes or functions to supply other functions like time control and command-line behaviour.

2.3. Detail of Algorithm

Here describe the detail of algorithm in some vital functions.

- ***RPSH***: A modified path scanning algorithm based on randomly choice of rules applying for next edges added to the route.

Algorithm 1 RPSH

```
Input: required arc $REARC$, source of routes $DEPOT$, demand record $DEMAND$, cost record $COST$
Output: a solution $S$
 1: $k \leftarrow 0$; $S \leftarrow \emptyset$
 2: copy all required arcs in a list $rearc$
 3: while $rearc \neq \emptyset$ do
 4:     $k \leftarrow k + 1$; $R_k \leftarrow \emptyset$; $load(k) \leftarrow 0$;$i \leftarrow 1$
 5:     while $rearc \neq \emptyset$ or $\overline{d} \neq 0$ do
 6:         $\overline{d} \leftarrow \infty$
 7:         for $u \in rearc|load(k) + q_n \leq Q$ do
 8:             if $d_{i,beg(u)} \leq \overline{d}$ then
 9:                 $\overline{d} \leftarrow d_{i,beg(u)}$
10:                 $\overline{u} \leftarrow u$
11:             else if $d_{i,beg(u)} = \overline{d}$ and $better(u, \overline{u}, rule)$ then
12:                 $\overline{u} \leftarrow u$
13:             end if
14:         end for
15:         add $\overline{u}$ at the end of the route $R_k$
16:         remove arc $\overline{u}$ and its opposite $\overline{u} + m$ from $rearc$
17:         $load(k) \leftarrow load(k) + q_n$
18:         $cost(k) \leftarrow cost(k) + \overline{d} + c_{\overline{u}}$
19:         $i \leftarrow end(\overline{u})$
20:     end while
21:     $cost(k) \leftarrow cost(k) + d_{i1}$
22:     add $R_k$ into $S$ // store routes
23: end while
24: return $S$
```

3. Empirical Verification

Empirical verification is compared with keep the same level of the result of path scanning because of failure of optimal. The reason might be the trap of local minimal but I did not conduct efficient mutations to conquer this problem.

3.1. Design

This problem is a NP hard problem which usually needs to be solved by some stochastic algorithms so I need do more on the design of the algorithm to overcome the obstacle getting improved result.

3.2. Data and data structure

Data being used in this project is five txt files for test the codes.

The data structures used in this project include list, tuple, dict, adjacent matrix and adjacent array.

3.3. Performance

The main time complexity is the path scanning which take $O(n^3)$ running time.

3.4. Result

- ’gdb1.dat’:
  - s 0,(1, 12),(12, 6),(6, 7),(7, 1),(1, 4),0,(1, 10),(10, 9),(9, 2),(2, 4),(4, 3),0,(1, 2),(2, 3),(3, 5),(5, 11),(11, 8),0,(12, 7),(7, 8),(8, 10),(10, 11),(11, 9),0,(12, 5),(5, 6),0
  - q 337
- ’gdb10.dat’:
  - s 0,(1, 9),(9, 3),(3, 4),(4, 7),(7, 5),(5, 6),(11, 4),0,(1, 2),(2, 3),(3, 8),(8, 9),(9, 10),(10, 12),(12, 11),0,(1, 4),(4, 6),(6, 11),(11, 1),(1, 8),(8, 12),(10, 1),0,(1, 5),(5, 2),(2, 7),(2, 4),0
  - q 289
- ’val1A.dat’:
  - s 0,(1, 9),(9, 19),(19, 22),(22, 23),(23, 24),(24, 21),(21, 20),(20, 19),(19, 1),(1, 5),(5, 4),(4, 3),(3, 10),(10, 2),(2, 5),(5, 6),(6, 11),(11, 5),(2, 4),(3, 9),0,(1, 20),(20, 15),(15, 21),(21, 23),(15, 16),(16, 17),(17, 12),(12, 16),(17, 18),(18, 14),(14, 13),(13, 17),(13, 7),(7, 8),(8, 14),(7, 6),(6, 12),(12, 11),(11, 1),0
  - q 179
- ’val4A.dat’:
  - s 0,(1, 2),(2, 3),(3, 4),(4, 5),(5, 6),(6, 12),(12, 11),(11, 17),(17, 20),(20, 27),(27, 21),(21, 20),(20, 19),(19, 26),(26, 32),(32, 31),(31, 25),(25, 24),(24, 14),(14, 23),(23, 13),(13, 7),(7, 8),(8, 2),(3, 9),(9, 8),(7, 1),0,(9, 14),(14, 13),(23, 24),(24, 30),(30, 29),(29, 34),(34, 38),(38, 39),(39, 40),(40, 36),(36, 37),(37, 41),(41, 40),(39, 36),(36, 32),(32, 27),(27, 26),(26, 25),(25, 15),(15, 16),(16, 19),(16, 11),(11, 5),(10, 15),0,(9, 10),(10, 11),(17, 16),(15, 14),(23, 29),(30, 31),(31, 35),(35, 36),(39, 35),(35, 34),(27, 28),(28, 22),(22, 21),(27, 33),(33, 37),(22, 18),(18, 17),(10, 4),0
  - q 429
- ’val7A.dat’:
  - s 0,(1, 35),(35, 28),(28, 27),(27, 26),(26, 33),(33, 1),(1, 2),(2, 6),(6, 7),(7, 3),(3, 2),(2, 36),(36, 37),(37, 38),(38, 31),(31, 37),(37, 30),(30, 31),(31, 32),(32, 39),(39, 5),(5, 4),(4, 3),(7, 13),0,(1, 8),(8, 40),(40, 1),(1, 10),(10, 16),(16, 11),(11, 12),(12, 13),(13, 17),(17, 18),(18, 13),(13, 19),(19, 18),(18, 22),(22, 25),(25, 24),(24, 21),(21, 20),(20, 23),(23, 24),(21, 22),(20, 17),(17, 12),(12, 6),(6, 1),0,(1, 11),(16, 15),(15, 9),(9, 8),(8, 14),(14, 15),(9, 10),(33, 34),(34, 26),(27, 34),(34, 35),(35, 36),(36, 29),(29, 30),(37, 3),(4, 38),(38, 39),0
  - q 305
- ’egle1A.dat’:
  - s 0,(1, 2),(2, 3),(2, 4),(4, 69),(69, 59),(59, 58),(58, 69),(59, 11),(10, 9),0,(4, 5),(11, 12),(12, 16),(16, 13),(13, 14),(15, 17),(15, 18),(18, 19),(19, 20),(20, 76),(50, 52),(52, 54),(50, 49),(49, 47),0,(59, 44),(44, 46),(46, 47),(47, 48),(49, 51),(51, 21),(21, 22),(22, 75),(75, 23),(23, 31),(31, 32),(32, 34),(56, 55),0,(58, 60),(60, 62),(62, 63),(63, 65),(62, 66),(66, 68),(60, 61),(58, 57),(57, 42),0,(44, 43),(41, 35),(35, 32),(32, 33),(21, 19),(45, 44),0
  - q 3955
- ’egls1A.dat’:
  - s 0,(1, 116),(116, 117),(117, 2),(117, 119),(118, 114),(114, 113),(113, 112),(112, 110),(110, 107),(107, 108),(107, 112),0,(110, 111),(107, 106),(106, 105),(105, 104),(104, 102),(66, 67),(67, 69),(69, 71),(71, 72),(72, 73),(73, 44),(44, 45),(45, 34),(43, 46),(86, 87),0,(86, 85),(85, 84),(84, 82),(82, 80),(80, 79),(79, 78),(78, 77),(77, 46),(43, 37),(37, 36),(36, 38),(38, 39),(39, 40),(67, 68),0,(108, 109),(66, 62),(62, 63),(63, 64),(64, 65),(56, 55),(55, 54),(55, 140),(49, 48),0,(124, 126),(126, 130),(43, 44),(34, 139),(139, 33),(33, 11),(11, 12),(12, 13),(20, 22),0,(95, 96),(96, 97),(97, 98),(140, 49),(11, 27),(27, 28),(28, 30),(30, 32),(28, 29),(27, 25),(25, 24),(24, 20),0,(11, 8),(8, 6),(6, 5),(8, 9),(13, 14),0
  - q 5953

3.5. Analysis

This project is far more difficulty than the former one Go project. Although there are many present research results can be re-conducted, but the process is not easy to do and I failed but I do think the main problem is the time schedule and person ability. I will try hard next time. Great pity I cannot finished this project in time. And we do see there is a long way to go on even I tried this for some day and nights.

Acknowledgments

I would like to thank Xizi Ni, Ziqiang Li, Yulian Mao for exchanged some thoughts with me on some vital algorithms although I did not successfully finished this project. But I have gained a lot through paper reading and algorithm implementations.

References

[1] Chen, Y., Hao, J. and Glover, F. (2016). A hybrid metaheuristic approach for the capacitated arc routing problem. European Journal of Operational Research, 253(1), pp.25-39.

[2] Brando, J. and Eglese, R. (2008). A deterministic tabu search algorithm for the capacitated arc routing problem. Computers & Operations Research, 35(4), pp.1112-1126.