An improved path-scanning for CARP

1. Preliminaries

The capacitated arc routing problem (CARP) is a challenging combinatorial optimization problem with many real-world applications, e.g., salting route optimization and eet man- agement. Given a graph with some edges and arcs required tobe served (called tasks) and a number of vehicles with limited capacity, a CARP is dened as seeking an optimal routing plan for the vehicles under the following conditions.[1]

- Each vehicle starts and ends at a predened vertex, namely depot.
- Each task is served by exactly one vehicle.
- The total demand of the tasks served by each vehicle does not exceed its capacity.

1.1. Software

This project is written in Python using IDE PyCharm. The libraries being used includes NumPy, Copy, random, sys, geopt, os, re

1.2. Algorithm

The algorithm being used in this project includes Dijkstra and path-scanning.

2. Methodology

Path-sanning is similar to greedy algorithm. First, put all the arcs in a set F, pick up an arc which is near the node at the end of the route every time. The new heuristic differs fromthe Belenguer et al. one in that it utilizes an ellipse rule when the vehicle load is near capacity. Intuition suggests that as a vehicle's load approaches its capacity, its route should stay closer to the depot to reduce its cost to return to the depot when full.[2]

2.1. Representation

This project contains some main data need to be maintain during the process:

- information:all the information of the data, including the capacity, vehicles' amount.
- cost :adjacent matrix to sotre the cost information of two joined nodes
- demand:adjacent matrix to sotre the demand information of two joined nodes
- nodes_amount:nodes amount
- initial_route :dictionary to store the routes
- initial_load :dictionary to store the load every time
- all_distance:dictionary to store the minimum cost of two joined nodes
- total_demand:total demand of every arcs
- total_cost:total cost of every arcs
- all_cost:total cost of the routes

2.2. Architecture

Here list all functions in the Python file CARP_solver.y with their usage.

- ***open_file***: read the .dat file.
- ***dijkstra***:dijkstra algorithm, return the minimun distance between arbitrary points on the map.
- ***path_scanning***:main algorithm in this project, return detail routes.
- ***command_line_reader***: read command line from cmd.

The last of the Python file, use *main()* to test all the codes. I also write a simple crossover function to improve the performance of the result.However it doesn't work.

- Duplicate 20 copies of parent-route set as route
- Random find a segment seg1 in the route[i][k1], and random decide whether to switch the direction
- Random find a segment seg2 in the route[i][k2], and random decide whether to switch the direction
- Insert seg1 into route[i][k2] and insert seg1 into route[i][k1]
- Calculate the minimum cost of these 20 routes, set the route as the next parent-route
- Run this program for 1000 times

2.3. Detail of Algorithm

Algorithm 1 path-scanning

Input: $information$, $nodes$,$cost$,$demand$
Output: the routes of the problem

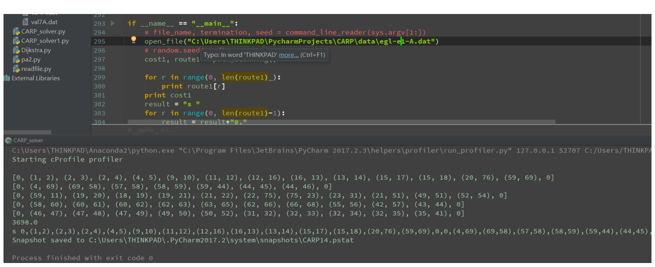
$k \leftarrow 0$
copy all required arcs which demand are larger than 0 in a list $free$
while $free \neq \emptyset$ **do**
  $k \leftarrow k+1$,$R_k \leftarrow \emptyset$,$load(k), cost(k) \leftarrow 0$,$i \leftarrow 1$
  while $free \neq \emptyset$ **do**
    for each $u \in free$ **do**
      if $rvc > \aleph \times td/ned$ **then**
        if $d_{i,beg(u)} < \bar{\text{d}}$ and $load < capacity$ **then**
          $\bar{\text{d}} \leftarrow d_{i,beg(u)}$
          $\bar{\text{u}} \leftarrow u$
        else if $d_{i,beg(u)} = \bar{\text{d}}$ and $better(u, \bar{\text{d}})$ **then**
          $\bar{\text{u}} \leftarrow u$
        end if
      else if $SP(v_i, v_p) + C_{pj} + SP(v_j, v0) \leq tc \div ned + SP(v_j, v_0)$ **then**
        if $d_{i,beg(u)} < \bar{\text{d}}$ and $load < capacity$ **then**
          $\bar{\text{d}} \leftarrow d_{i,beg(u)}$
          $\bar{\text{u}} \leftarrow u$
        else if $d_{i,beg(u)} = \bar{\text{d}}$ and $better(u, \bar{\text{d}})$ **then**
          $\bar{\text{u}} \leftarrow u$
        end if
      end if
    end for
    add $\bar{\text{u}}$ at the end of route R_k
    remove arc $\bar{\text{u}}$ from $free$
    $load(k) \leftarrow load(k) + q_{\bar{\text{u}}}$
    $cost(k) \leftarrow cost(k) + \bar{\text{d}} + c_{\bar{\text{u}}}$
    $i \leftarrow end_{\bar{\text{u}}}$
  end while
end while
return $points\_status \leftarrow is\_not\_alive$

3. title

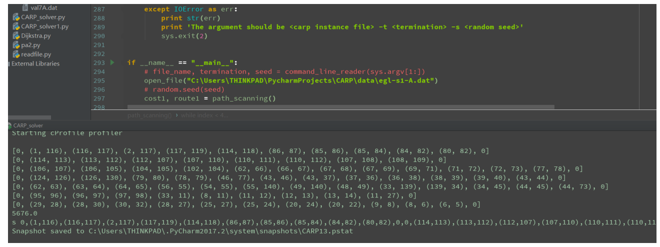
4. Empirical Verification

Empirical verification is compared with the given ans in the dat file:

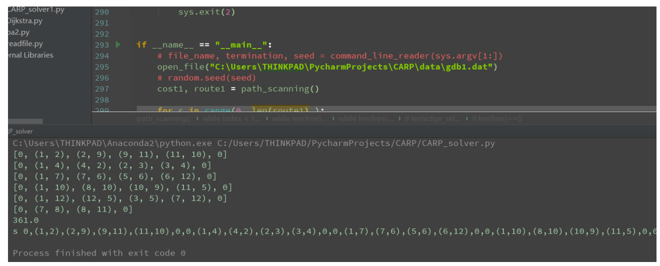
. **egl-s1-A**



. **egl-e1-A**

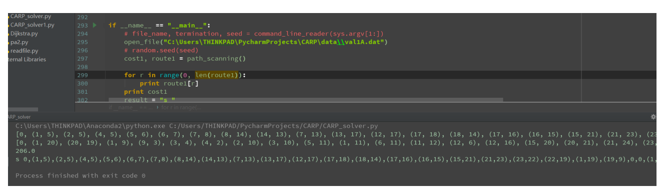


. **gdb1**

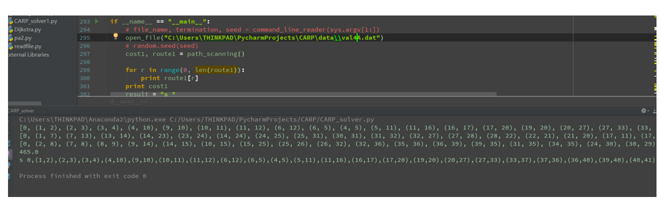


. **gdb10**

. **val1A**



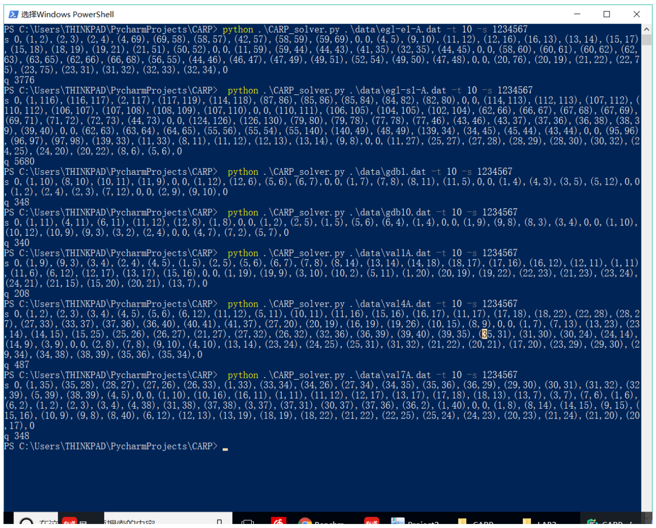
. **val4A**



. **val7A**

Test in cmd: time 10s, seed:1234567

. **Test in command**



I find that the improved path-scanning algorithm performs well at egl-e1-A rather than egl-s1-A although their data amount are similar. This may because the limitation of this kind of heuristic search.

Acknowledgments

I would like to thank Ni, Li, Zheng 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. Maybe I have to manage my time in rest projects. It is very regrettable of not having enough time to finish my optimize part.

References

[1] Y. Mei, K. Tang and X. Yao, ”Decomposition-Based Memetic Algorithm for Multiobjective Capacitated Arc Routing Problem”, IEEE Transactions on Evolutionary Computation, vol. 15, no. 2, pp. 151-165, 2011.

[2] L. Santos, J. Coutinho-Rodrigues and J. Current, ”An improved heuristic for the capacitated arc routing problem”, Computers & Operations Research, vol. 36, no. 9, pp. 2632-2637, 2009.

[3] ”Dijkstra’s algorithm”, En.wikipedia.org, 2017. [Online]. Available: https://en.wikipedia.org/wiki/Dijkstra