Report for capacitated arc routing problem (CARP)

Ziqiang LI

Computer Science and Engineering,

Southern University of Science and Technology

11510352

1. Preliminaries

The capacitated arc routing problem (CARP) is an important and practical problem. I follow a memetic algorithm to solve this problem[1], and get the idea of mutation with in single route[2]. I set a trigger that will restart the population mutation when it trapped in local minimum.

In general, I get a group of initial feasible solution from path scanning, and then the group repletely reproduce and select the best offsprings. Besides, I also use Dijkstra algorithm to find shortest path linking the required edges.

2. Methodology

Main function in CARP.py

- **read data:** read data from formatted file, and convert it into list *D*.
- **matrix tran:** transform the above *D* into adjacent matrix, and store cost and demand of all edge.
- **free edge set:** pick and return all required edge from *D*.
- **path scanning:** generate using five rules to offer initial feasible solution.
- **better:** using following rules to decide whether choose an edge in *path scanning*.
- **class Routing:**
  - **mutation:** following the four rules to generate new route.
    - **Single insertion**
    - **Double insertion**
    - **Swap**
    - **2-opt**
  - **fix:** if a route is overloading, it will invoke this to try to fix it. This function is similar to *path scanning*. The major difference is that this function finds the overload vehicles and rearrange redundant required edges. The majority sequence of current solution will be same.
  - **calc cost:** calculate the cost of route.
- **class Population:**
  - **generate:** using above mutation method to generate a give size offsprings.
  - **select:** select the best offsprings.
  - **check mature:** check the population if is mature.
  - **resize:** resize the current population into a given size. This function is used after the population is considered as matured.

Algorithm 7.2 – Path-Scanning for one priority rule

1. $k \leftarrow 0$
2. copy all required arcs in a list $free$
3. **repeat**
4. $\quad k \leftarrow k+1;\ R_k \leftarrow \emptyset;\ load(k), cost(k) \leftarrow 0;\ i \leftarrow 1$
5. $\quad$ **repeat**
6. $\quad\quad \bar{d} \leftarrow \infty$
7. $\quad\quad$ **for each** $u \in free \mid load(k) + q_u \leq Q$ **do**
8. $\quad\quad\quad$ **if** $d_{i,beg(u)} < \bar{d}$ **then**
9. $\quad\quad\quad\quad \bar{d} \leftarrow d_{i,beg(u)}$
10. $\quad\quad\quad\quad \bar{u} \leftarrow u$
11. $\quad\quad\quad$ **else if** $(d_{i,beg(u)} = \bar{d})$ and $better(u, \bar{u}, rule)$
12. $\quad\quad\quad\quad \bar{u} \leftarrow u$
13. $\quad\quad\quad$ **endif**
14. $\quad\quad$ **endfor**
15. $\quad\quad$ add $\bar{u}$ at the end of route R_k
16. $\quad\quad$ remove arc $\bar{u}$ and its opposite $\bar{u}+m$ from $free$
17. $\quad\quad load(k) \leftarrow load(k) + q_{\bar{u}}$
18. $\quad\quad cost(k) \leftarrow cost(k) + \bar{d} + c_{\bar{u}}$
19. $\quad\quad i \leftarrow end(\bar{u})$
20. $\quad$ **until** $(free = \emptyset)$ or $(\bar{d} = \infty)$
21. $\quad cost(k) \leftarrow cost(k) + d_{i1}$
22. **until** $free = \emptyset$

better function – using in *path scanning* line 11

1. Maximize the ratio c_{ij} / r_{ij}, 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. If the vehicle is less than half-full, then apply rule 3, otherwise rule 4.

- How to deal with local minimum?

If I find the temporary solutions in population that all routes have same cost, I suppose it trap into local minimum. So, I will restart the population with just some top routes.

3. Empirical Verification

- 3 python files are used in my project.
- Dijkstra.py gives the functions to find shortest path between given two edges.
- CARP_solver.py provides the command line parser to get all needed arguments for solve the problem.
- CARP.py is the most important part in my project. Its includes the implement to memetic algorithm. I also follow the rule of encapsulation to make the function readable.

4. Perfermance

I run my algorithm on gdb1, gdb10, val1A, val4A, val7A, egl-e1-A, egl-s1-A.

Data set	*My solute.*	*Best solute.*
gdb1	316	316
gdb10	275	275
val1A	173	173
val4A	402	400
val7A	290	279
egl-e1-A	3683	3548
egl-s1-A	5194	5018

For the beginning 3 sets which is not complicated, I can get a solution that is the same as the best or very close to the best. For last 2 complicated set, my errors are within 4%.

5. References

[1] A. C. C. Corberán and G. Laporte, *Arc routing: problems, methods*, and applications. Philadelphia: Society for Industrial and Applied Mathematics, 2014.

[2] K. Tang, Y. Mei, and X. Yao, “Memetic algorithm with extended neighborhood search for capacitated arc routing problems,” *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 5, pp. 1151–1166, 2009.