# Report for Project CARP

11610511 Shiyi Li

November 2018

## 1 Preliminaries

bibliographyreferences The capacitated arc routing problem(CARP) consists of demands placed on the edges, and each edge must meet the demand. The problem is to identify routes to service demand located along the edges of a network such that the total cost of the routes is minimized. An example is garbage collection, where each route might require both a garbage collection and a recyclable collection. The CARP is NP-hard.[1] The project is written in python and the IDE is PyCharm. No additional library is used.

## 2 Methodology

The basic definition of each variables will be introduced in the methodology.

### 2.1 Representation

Several variables should be defined in advance in the algorithm.

- given: a dictionary to store all the predefined value of CARP problem
  - VERTICES: number of vertices in the graph
  - DEPOT: number that represents depot
  - REQUIRED EDGES: number of edges with demand
  - NON-REQUIRED EDGES: number of edges without demand
  - VEHICLES: number of vehicles
  - CAPACITY: the maximum capacity of each vehicle
  - TOTAL COST OF REQUIRED EDGES: total cost of required edges
- cost_each: distance of each edge
- demands_each: demand of each edge
- distances: the shortest distance between arbitrary point
- paths: the shortest path between arbitrary point

### 2.2 architecture

First, construct a graph from the given file. Then, calculate the shortest distance between each pair of points. Third, obtain the initial solution from path scanning. Last, use deterministic tabu search algorithm to get the optimal route.

- Class Graph:
  - add_edge: record the edge, its cost and demand
- Class RouteType: record the path, cost and remain capacity of each route
- readfile: read .dat file and construct the graph
- dijkstra: calculate the shortest distance between arbitrary points
- path_scanning: obtain initial solutions
- single_insertion: remove a candidate edge from its current route and insert it in other route between any two serviced edges that are not adjacent

- double_insertion: remove a candidate consists of two connected required edges from its current route and insert it in other route between any two serviced edges that are not adjacent
- TSA: construct and maintain a tabu list to avoid local optimum and find the optimal route

## 2.3 Algorithm

Here shows the exact algorithm used in the CARP, which includes dijkstra algorithm, path scanning and tabu search algorithm.

**Algorithm 1** dijkstra

**Input:** graph, start \\ start is the starting point of the graph when calculating the distance
**Output:** distances, paths

```
1: create vertex set Q
2: for each vertex v in the graph do
3:    distances[v] ← INFINITY
4:    paths[v] ← UNDEFINED
5:    add v to Q
6: end for
7: distances [start] ← 0
8: while Q is not empty do
9:    u ← vertex in Q with min dist[u]
10:   remove u from Q
11:   for each neighbor vofu do
12:      alt ← distances[u] + cost_each(u, v)
13:      if alt < distances[v] then
14:         distances[v] ← alt
15:         paths[v] ← u
16:      end if
17:   end for
18: end while
```

**dijkstra:** find the shortest distances between arbitrary vertex.

**Algorithm 2** path_scanning for one priority rule

**Input:** graph
**Output:** route, cost

```
1: k ← 0
2: copy all required arcs in a list free
3: while free ≠ empty do
4:    k ← k + 1
5:    R_k ← emptyset
6:    cost(k), load(k) ← 0
7:    i ← 1
8:    while free ≠ empty and d ≠ ∞ do
9:       d ← ∞
10:      for each u ∈ free and load(k) + q_u ≤ Q do
11:         if d_{i,beg(u)} < d then
12:            d ← d_{i,beg(u)}
13:            u_n ← u
14:         end if
15:         if d_{i,beg(u)} = d and better(u, u_n, rule)
            then
16:            u_n ← u
17:         end if
18:      end for
19:      add u_n at the end of route R_k
20:      remove arc u_n and its opposite u_n + m from
         free
21:      load(k) ← load(k) + q_{u_n}
22:      cost(k) ← cost(k) + d + c_{u_n}
23:      i ← end(u_n)
24:   end while
25:   cost(k) ← cost(k) + d_{i1}
26: end while
```

**path_scanning:** this method is to obtain initial route.[2]

Five rules are used to determine the next required edge, $e$, in the route to be served:
(1) maximise the distance to the depot
(2) minimise the distance to the depot
(3) use rule 1 if the vehicle is less than half-full, else use rule 2
(4) maximise the ratio $d(e)/c(e)$, where $d(e)$ and $c(e)$ are the demand and cost of e
(5) minimise the ratio $d(e)/c(e)$

**The tabu search algorithm**

- **Neighborhood moves**: In a single insertion move, a candidate edge (only required edges can be candidates) is removed from its current route and a trial insertion is made in any other route between any two serviced edges that are not adjacent, but are linked by a deadheading path of edges. The trial insertion considers both directions for the edge to be traversed when inserted in the new route. In a double insertion move, the operation is similar except that a candidate consists of two connected required edges in one route. [1]

- **Objective function**: The objective function to be minimised by the TSA includes the total cost of each route, $i$, plus a penalty cost $Pw(i)$, where $P$ is a penalty term and $w(i) = max(x(i) - Q, 0)$, and where $x(i)$ is the sum of the demands on the edges serviced in route $i$. For any candidate solution, $S$, the objective function is denoted by $f(S)$.

- **Admissibility of moves**: In the TSA, a trial move to solution S is regarded as admissible:
  (i) if it is not currently on the tabu list;
  (ii) or if it is tabu and infeasible, but the value of f(S) is less than the value of the best infeasible solution yet found;
  (iii) or if it is tabu and feasible, but the value of f(S) is less than the value of the best feasible solution yet found.

- **Outline of TSA**: The operation of the TSA can now be summarised in the following steps:
  *1. Initialise*:
  Set current solution $S$ to be an initial solution and let $f(s)$ be the total cost of $S$
  Set best solution $S_B = S$
  Set best feasible solution $S_{BF} = S$, if $S$ is feasible; otherwise set $f(S_{BF}) = \infty$
  Set iteration counter, $k = 0$
  Set number of iterations since best solution after Intensication step, $k_B = 0$
  Set number of iterations for applying Intensication step, $k_L = 8N$
  Set number of iterations since best feasible solution, $k_{BF} = 0$
  Set number of consecutive iterations that solution is feasible, $k_F = 0$
  Set number of consecutive iterations that solution is infeasible, $k_I = 0$
  Set number of iterations since best solution in total, $k_{BT} = 0$
  Set tabu list to be empty Set tabu tenure, $t = N/2$
  Set frequency parameters, $F_{SI} = 1$, $F_{DI} = 5$
  Set penalty parameter, $P = 1$.

  *2. Find neighbourhood move ($S'$) from S, using all required edges as a candidate list.*: Set $f(S') = \infty$.
  2.1. If $k$ is a multiple of $F_{SI}$ perform trial single insertion moves
  For each admissible move, $s$, do:
  If $f(s) < f(S')$, then
  (1) $S' = s$ and $f(S') = f(s)$;
  (2) If $s$ is feasible and $f(s) < f(S_{BF})$, or $f(s) < f(S_B)$ go to Step 3.
  Repeat until all the potential moves have been explored.
  2.2. If $k$ is a multiple of $F_{DI}$ perform trial double insertion moves
  For each admissible move, $s$, do:
  If $f(s) < f(S')$, then
  (1) $S' = sandf(S') = f(s)$;
  (2) If $s$ is feasible and $f(s) < f(S_{BF})$, or $f(s) < f(S_B)$ go to Step 3.
  Repeat until all the potential moves have been explored.

  *3. Improve changed routes*: For each of the two routes that have been changed, apply the route improvement procedure using Frederickson's heuristic to try to nd further cost reductions if possible, resulting in solution $S''$.

  *4. Update*:
  Update tabu list.
  If $S''$ is feasible and $f(S'') < f(S_{BF})$ then
  (1) apply Frederickson's heuristic to each route individually (except the two routes that have just been updated in Step 3) to get a new solution $S'''$

(2) set $S_{BF} = S'''$, $S'' = S'''$ and set $k_B = k_{BF} = k_{BT} = 0$.
If $f(S'') < f(S_B)$ then set $S_B = S''$ and $k_B = k_{BT} = 0$.
Set $k = k + 1$, $k_B = k_B + 1$, $k_{BF} = k_{BF} + 1$, $k_{BT} = k_{BT} + 1$.
If $k$ is a multiple of 10, then if $k_F = 10$, then set $P = P/2$ else if $k_I = 10$ then set $P = 2P$
If $k_F = 10$ or $k_I = 10$ then recalculate $f(S_B)$ with the new value of $P$ and set $k_F = k_I = 0$.

*5. Change of parameter values*:
If $k_B = k_L/2$, then set $F_{SI} = 5$, $F_{DI} = 1$.

*6. Intensication*:
If $k_B = k_L$ , then
(1) if $f(S_{BF}) < \infty$ then set $S = S_{BF}$ else set $S = S_B$ ,
(2) set $k_B = 0, P = 1, kF = 0, kI = 0, F_{SI} = 1, F_{DI} = 5, k_L = k_L + 2N$,
(3) recalculate $f(S_B)$ with the new value of $P$, and empty tabu list.

*7. Stopping criterion*:
If ($k >= 900\sqrt{N}$ and $k_{BF} >= 10N$) or $k_{BT} = 2kL$ then stop, otherwise go to Step 2.

# 3 Empirical Verification

## 3.1 Dataset

Dataset source:
*http : //logistik.bwl.unimainz.de/benchmarks.php*

## 3.2 Performance measure

The version of Python is 3.7.0 and the run time is set to be 60s. Performance is measured by comparing the best known lower bound and my optimal result.

## 3.3 Hyperparameters

The main optimal algorithm is based on a tabu search algorithm (TSA). However, it differs from CARPET in many of the details of the implementation. In particular, the algorithm is deterministic and does not require the use of random parameter values (which are also needed for MA), so making the results fully reproducible. [1]

Table 1: Performance

| Instances | Best Known LB | Test(60s) |
|---|---|---|
| val1A | 173 | 285 |
| val1B | 173 | 242 |
| val1C | 245 | 460 |
| val2A | 227 | 318 |
| val2B | 259 | 309 |
| val2C | 457 | 512 |
| egl-e3-A | 5898 | 9466 |
| egl-e3-B | 7744 | 11472 |
| egl-e3-C | 10244 | 13342 |
| egl-s4-A | 12153 | 18385 |
| egl-s4-B | 16113 | 22847 |
| egl-s4-C | 20430 | 27349 |
| gdb13 | 536 | 571 |
| gdb20 | 121 | 138 |
| gdb21 | 156 | 163 |
| gdb22 | 200 | 220 |
| gdb23 | 233 | 241 |

## 3.4 Experimental results

Experimental result is showed in Table 1.

## 3.5 Analysis

The path scanning give the initial solution for CARP and TSA optimize the solution. Five rules of path scanning are all used to get the optimal initial result. However, the TSA seems to be stuck in local optimum and the final solution does not performs well, especially the large scale dataset. The problem still remains unfixed. Further more, a new ellipse rule can be applied to path scanning to obtain a better initial solution.[3]

# References

[1] José Brandão and Richard Eglese. A deterministic tabu search algorithm for the capacitated arc

routing problem. *Computers & Operations Research*, 35(4):1112–1126, 2008.

[2] Bruce L Golden, James S DeArmon, and Edward K Baker. Computational experiments with algorithms for a class of routing problems. *Computers & Operations Research*, 10(1):47–59, 1983.

[3] Luís Santos, João Coutinho-Rodrigues, and John R Current. An improved heuristic for the capacitated arc routing problem. *Computers & Operations Research*, 36(9):2632–2637, 2009.