Efficiency-based Path-Scanning heuristic and Memetic Algorithm for Capacitated Arc Routing Problems

Tianao Wang

Southern University of Science and Technology
12011014

1. Introduction

1.1. Background

Arc Routing Problem is the problem which we need to choose the best path based on different conditions of routes.[2] The capacitated arc routing problem (CARP) is one variant of Arc Routing Problem.The agent has a limited capacity while processing demand, which means that it has to go back to some warehouse to empty its current load before its load exceeds its capacity. It is an important combinatorial optimization problem that has been extensively studied in the last decades. [1]

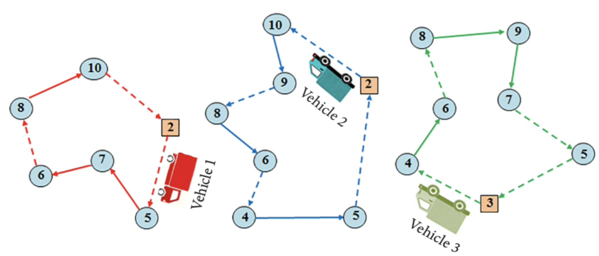

Figure 1. CARP

Because CARP is an NP-hard problem[3] for which many exact and heuristic methodologies have been proposed. In this project, we aim to implement an algorithm that can find a routing satisfying all of demand with the total cost as little as possible.

1.2. Algorithms

In this project, I have used the following algorithms:

1) *Path scanning*: The main algorithm used to making valid individuals for the CARP
2) *Efficiency-based Path-Scanning*: An optimization on Path scanning to quickly find high-quality solutions by bringing in some limitations.
3) *MAs*: An metaheuristic algorithm combining a local search algorithm with a genetic algorithm to produce new valid individuals with lower total cost.
4) *Local search*: A type of optimization algorithm in MAs that is used to find high-quality solutions by making small changes to the routes.

1.3. Application

The ARC routing problem is a classic problem with many applications in the real world, such as urban waste collection, post delivery, sanding or salting the streets, etc.[4][5]Moreover, many real-world complex operations encouraged the emergence of CARP variants.[1]

2. Methodology

2.1. Notation

There are some notations that I will use in my essay:

Symbol	Definition
$G(V, E)$	the whole undirected connetced graph
v_0	the depot for each vehicles start and end
v_h	the current location of the vehicle
E_R	the edge set need to required
ned	the number of required edges
td	the total demand of a edges set
tc	the total cost of a edges set
D	the initial capacity of vehicle
rvc	the remaining vehicle capacity
$SP(v_p, v_q)$	the shortest path cost from v_p to v_q
α	a real parameter for building threshold
P_{ls}	probability of mutation

TABLE 1. Notations List

2.2. Problem Formulation

The CARP can be formulated as: A route is the trajectory of a vehicle in G, represented by a set of passing edges, (1) starting and ending at the depot v_0. A set of paths l_e can be defined if (2) the sum of satisfied demands on each path e_{ij} is at most D ($\sum e_{ij} \leq D, e_{ij} \in l_e$) and (3) each edge to be served is served by only one vehicle.($e_{ij} \neq e_{mn}, e_{ij}, e_{mn} \in$

$l_e, \forall(i, j \neq m, n))$ The goal is to find a set of feasible paths with minimum cost.[1][6]

2.3. General workflow

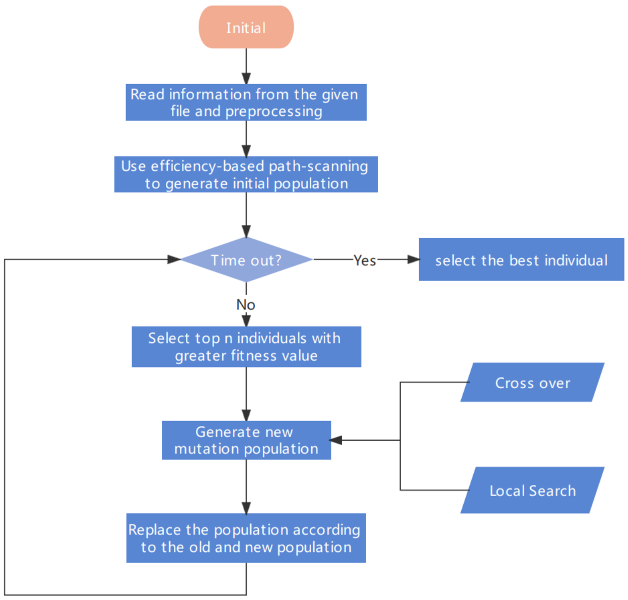

Figure 2. WorkFlow

2.4. Details of Algorithms

I will introduce the two core algorithms in this project.

2.4.1. Path Scanning with Efficiency-based.

Path scanning is a type of optimization algorithm that is used to solve the capacitated arc routing problem (CARP). The idea of this algorithm is try to find the edge closest to the vehicle position v_h each time. If there is not only one edge satisfied, randomly use *FiveRules* to choose one. The pseudocode of the *FiveRules* is listed as follow:

Algorithm 1 *FiveRules*

Input: rule, edges, rvc
Output: edge(v_i,v_j)

```
rule = 1, edge ← edges: max(SP(v_j,v_0))
rule = 2, edge ← edges: min(SP(v_j,v_0))
rule = 3, edge ← edges: max(d(edge)/c(edge))
rule = 4, edge ← edges: min(d(edge)/c(edge))
rule = 5, if rvc>D/2, choose rule1, else choose rule2
return edge
```

The path-scanning with efficiency rule(PS-Efficiency) is a heuristic based on the path-scanning with ellipse rule. It constructs a set of feasible routes in a greedy fashion, where each route is created starting from the depot and then sequentially selecting the nearest unserviced edge $(v_i, v_j) \in ER$ for which the demand $d(v_i, v_j)$ does not exceed the remaining vehicle capacity. Similarly to PS-Ellipse, when the vehicle reaches a certain load threshold it activates an efficiency rule which restricts the vehicle to service only edges considered "cost-efficient".[1]

The function $near(v_h)$ is given by Equation 1 and defines the set of unserviced edges close enough to v_h. The route efficiency index $eff(R)$ is given by Equation 2 , which is the ratio of the demand serviced by distance traversed of route R.

$$\begin{aligned} \operatorname{near}(v_h) = \{ & (v_i, v_j) \in E_R \mid \min\{SP(v_h, v_i), SP(v_h, v_j)\} \\ & \leq \operatorname{tc}(E_R) / \operatorname{ned}(E_R) \text{ and } (v_i, v_j) \text{ is unserviced}\} \end{aligned} \tag{1}$$

$$\operatorname{eff}(R) = \frac{\sum_{i=1}^{n} d(u_i, v_i)}{SP(v_0, u_1) + \sum_{i=1}^{n} c(u_i, v_i) + \sum_{i=1}^{n-1} SP(v_i, u_{i+1}) + \operatorname{SP}(v_n, v_0)} \tag{2}$$

The load threshold that activates the efficiency rule is given by the triggering criteria (Equation 3). If $near(v_h)$ is empty, the algorithm adopts in Equation 4 instead of Equation 3.

$$\operatorname{rvc} \leq \alpha \times \operatorname{td}(\operatorname{near}(v_h)) / \operatorname{ned}(\operatorname{near}(v_h)) \tag{3}$$

$$\operatorname{rvc} \leq \alpha \times \operatorname{td} / \operatorname{ned} \tag{4}$$

Once the vehicle achieves the load threshold, the vehicle is constrained to service only edges $(v_i, v_j) \in ER$ satisfying the Equation 5.

$$\frac{d(v_i, v_j)}{SP(v_h, v_i) + c(v_i, v_j) + SP(v_j, v_0) - SP(v_h, v_0)} \geq \operatorname{eff}(R) \tag{5}$$

Figure 3 gives an example with three required edges $(v_i, v_j), (v_k, v_l)$ and (v_m, v_n). In this example, only the edges (v_i, v_j) and (v_m, v_n) are satisfying the efficiency rule.[1]

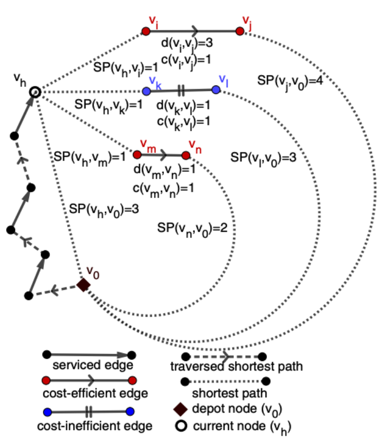

Figure 3. Example of efficiency rule for a route R with $eff(R)$=1 [1]

Regarding computational complexity, PS-Efficiency can be implemented in the same worst-case time complexity as

PS-Ellipse.[1] From the follow algorithm, we can get the whole procedure has $O(k|E_R|^2)$ time complexity.

Algorithm 2 *PS-Efficiency*

Input: $G(V, E)$: instance graph; D: vehicle capacity; c: cost vector; d: demand vector; SP: matrix of shortest path costs; α: real parameter; t': the time now; s: start time; t: termination time
Output: $bestSol$: best routing solution found;
$bestSol \leftarrow \emptyset$;
while $t' - s < t$ **do**
    $sol \leftarrow \emptyset$; $rvc \leftarrow D$; $v_h \leftarrow v_0$;$sumDist \leftarrow 0$;
    $R \leftarrow \{v_0\}$; $rule$ ←false;
    for n = 1 to ned **do**
        if $near(v_h) \neq \emptyset$ and inequation(5) **then**
            $rule \leftarrow$ true;
        else if $near(v_h) = \emptyset$ and inequation(1) **then**
            $rule \leftarrow$ true;
        end if
        if $rule$ is false **then**
            Determine set of candidate edges $F \subseteq E_R$
        else
            Determine set of candidate edges $F \subseteq E_R$
            satisfying (6);
        end if
        if $F = \emptyset$ **then**
            $sol \leftarrow sol \cup (R \cup \{v_0\})$; $rvc \leftarrow D$; $v_h \leftarrow v_0$;
            $R \leftarrow \emptyset$; $rule$ ←false; $sumDist \leftarrow 0$;
        else
            R append (v_i, v_j) randomly from F
            $sumDist \leftarrow sumDist + SP(v_h, v_i)$
            $+c(v_i, v_j)$; $rvc \leftarrow rvc - d(v_i, v_j)$;
            $v_h \leftarrow v_j$; $eff(R) \leftarrow (D - rvc)/(sumDist$
            $+SP(v_h, v_0))$
        end if
    end for
    if $(cost(sol) < cost(bestSol))$ or $bestSol = \emptyset$ **then**
        $bestSol \leftarrow sol$
    end if
end while
return $bestSol$

2.4.2. MAs.

I have tried to use the MAs method in my code. I will briefly summarize MAs including initialization, crossover, and local search[6].

Firstly, It takes me half the time given to initialize the population using the method *PS-Efficiency* described above. Then, crossover is implemented by applying the sequence based crossover (SBX) operator to two parent individuals randomly selected from the current population at each iteration.[6] During each iteration to produce new individual, there is a possibility to mutate by using local search method above. I have used four local search methods show in the follow pseudocode. Finally, use a efficient fitness function to sort the solution.

Algorithm 3 *LocalSearch*

Input: $route$
Output: $best\_route, cost\_min$
$route1, cost1 \leftarrow flip(route)$
$route2, cost2 \leftarrow Swap(route)$
$route3, cost3 \leftarrow TwoOptSinglePath(route)$
$route4, cost4 \leftarrow TwoOptDoublePath(route)$
$cost\_min \leftarrow min(cost1, cost2, cost3, cost4)$
$best\_route \leftarrow$ the route of the $cost\_min$
return $best\_route, cost\_min$

The overall framework process of MAs is shown in the following pseudocode:

Algorithm 4 *MAs*

Input: A CARP instance,$psize, opsize, ubtrial, P_{ls}$
Output: A feasible solution S_{bf}
initialize population pop by *PS-Efficiency*
$psize = |pop|$;
while $t' - s < t$ **do**
    Set an intermediate population pop_t = pop;
    for $i = 1 \rightarrow opsize$ **do**
        Randomly select two solutions S_1 and S_2 as
        parent from pop
        Apply the crossover operator to S_1 and S_2 to
        generate S_x ;
        if random $r < P_{ls}$ **then**
            Apply local search to S_x to generate S_{ls};
            if $S_{ls} \notin pop_t$ **then**
                $pop_t = pop_t \cup S_{ls}$
            end if
        end if
        if $S_x \notin pop_t$ **then**
            $pop_t = pop_t \cup S_x$
        end if
    end for
    Sort the solutions in pop_t using stochastic ranking;
    Set pop = {the best $psize$ solutions in pop_t}
end while
return the best feasible solution S_{bf} in pop

3. Experiment

3.1. Software and Hardware

3.1.1. Software.

- CARP code: Python (Editor: Pycharm CE 2022.2.2)
- Online usability and robin test: SUSTech Infinity OJ
- Report writing: LATEX (Editor: Overleaf)
- Python: Version 3.9

3.1.2. HardWare.

- Basic code and report writing: *MacBook Pro Intel Core i5 CPU*
- Test Platform: *Sustech OJ Server CPU with 32 cores 64 threads*

3.2. Dataset

We are given seven datasets in this CARP, and they are from about three different scale types (gdb:tiny, val:samll, egl:large) There are the details of the datasets:

group	$\|V\|$	$\|E_R\|$	$\|D\|$
val1A	24	39	200
val4A	41	69	225
val7A	40	66	200
gdb1	12	22	5
gdb10	12	25	10
egl-s1-A	140	75	210
egl-e1-A	77	51	305

TABLE 2. DATASET

3.3. Experimental Results

For the small graphs like val and gdb dataset, I use 60s to test the performance. For the large graphs like egl dataset, 600s is allowed to show the performance as sufficient as possible.

3.3.1. Different hyperparameter.

From the Equation 3 and 4, we can find α is a hyper-parameter that can effect the performance of the algorithm. Therefore, I select different α (α=1.0, α=1.5, α=2.0, α=2.5, α=3.0) in *PS-Efficient* algorithm to compare how they affect the performance of the algorithm.

α	**1.0**	**1.5**	**2.0**	**2.5**	**3.0**
val1A	173	173	173	173	173
val4A	411	406	406	404	408
val7A	284	279	280	279	279
gdb1	316	316	316	316	316
gdb10	275	275	275	275	275
egl-s1-A	5412	5403	5403	5370	5411
egl-e1-A	3773	3696	3699	3699	3699

TABLE 3. DIFFERENT α

From the table, we can find that for simple graphs, the convergent solution can be found in extremely short time for different α. For complex graph, the performance of algorithm first gets better and then gets worse as α increases.

3.3.2. Different Methods.

What's more, I compare different methods to find a best one that can solve this problem as efficient as enough. The experimental result of Path-Scanning(PS), the Efficiency-based Path-Scanning(PS-Eff), the Path-Scanning with local search(PS-LS) and the Efficiency-based Path-Scanning with local search (PS-Eff-LS) are list as below:

group	PS	PS-Eff	PS-LS	PS-Eff-LS
val1A	173	173	173	173
val4A	410	406	408	408
val7A	288	280	290	287
gdb1	316	316	316	316
gdb10	275	275	275	275
egl-s1-A	5938	5403	5938	5430
egl-e1-A	3931	3699	3850	3699

TABLE 4. DIFFERENT METHOD

From the result table, we can find the Efficiency-based Path-Scanning far better than the origin Path-Scanning. What's more, we can find by adding local search at Path-Scanning, the performance is improved. But by adding local search at Efficiency-based Path-Scanning, we find sometimes the performance become worse. The reason I speculate that the local search cost too much time so that there is not enough time for the Efficiency-based Path-Scanning to find a better solution. The Efficiency-based Path-Scanning has more obvious improvement effect on solution than local search.

4. Conclusion

4.1. Advantages and Disadvantages of the Algorithms

My CARP program performs well when graphs are small or medium in size. It can be easy to find the best or better solution. However, when the size of the graph increases, it is difficult to find the optimal solution due to the stuck in the local optimal solution.

Advantage:

1) use a more efficient Path scanning method (Efficiency based Path-Scanning)

2) use MAs that incorporates the idea of heredity and the use of the local search mutation to find a better solution

Disadvantage:

1) when graph become large, the performance decrease

2) still difficult to get out of the local optimal solution

4.2. Deficiencies and Possible Improvement Directions

If given me more time, I will improve the MAs algorithm to MAENS by using *Merge-Split Operator*[6] and combine with the Efficiency-based Path-Scanning.What's more, I will read more research papers and find better solutions to solve the CARP.

4.3. Experience

From this project, I study a lot in how to solve the classic NP-Hard arc routing problem. Its problem formation and strategies in using heuristic and genetic algorithms provided me with a training opportunity. In addition, the algorithms were inspired by the papers I searched and the lectures I studied. This is my first attempt to reproduce the idea of the paper and it made me experience the feeling of research.

References

[1] Rafael Arakaki and Fabio Usberti. "An efficiency-based path-scanning heuristic for the capacitated arc routing problem". In: *Computers Operations Research* 103 (Nov. 2018). DOI: 10.1016/j.cor.2018.11.018.

[2] Wikipedia contributors. *Arc routing — Wikipedia, the free encyclopedia*. https://en.wikipedia.org/w/index.php?title=Arcrouting&oldid=950319399. 2022.

[3] Bruce L. Golden and Richard T. Wong. "Capacitated arc routing problems". In: *Networks* 11.3 (1981), pp. 305–315. DOI: https://doi.org/10.1002/net.3230110308. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/net.3230110308. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/net.3230110308.

[4] H. Handa, L. Chapman, and Xin Yao. "Robust route optimization for gritting/salting trucks: a CERCIA experience". In: *IEEE Computational Intelligence Magazine* 1.1 (2006), pp. 6–9. DOI: 10.1109/MCI.2006.1597056.

[5] H. Handa et al. "Robust Solution of Salting Route Optimisation Using Evolutionary Algorithms". In: *2006 IEEE International Conference on Evolutionary Computation*. 2006, pp. 3098–3105. DOI: 10.1109/CEC.2006.1688701.

[6] Ke Tang, Yi Mei, and Xin Yao. "Memetic Algorithm With Extended Neighborhood Search for Capacitated Arc Routing Problems". In: *IEEE Transactions on Evolutionary Computation* 13.5 (2009), pp. 1151–1166. DOI: 10.1109/TEVC.2009.2023449.