Genetic Algorithm with Heuristics and Merge-Split for Capacitated Arc Routing Problems

12110813 Liu Shengding

***Abstract*—Since Golden and Wong proposed the Capacity Constrained Arc Routing Problem (CARP) in 1981, the capacitated arc routing problem (CARP) has attracted much attention due to its wide applications in real life. Since CARP is NP-hard and exact methods are only applicable to small instances, heuristic and metaheuristic methods are widely adopted when solving CARP. In this paper, we take advantage of the genetic algorithm with heuristics and MS to solve the CARP. First,we use Path-Scanning algorithm and Random-Pick algorithm to initialize a set of valid solutions, which is thought of the population. And then we take advantage of a novel local search operator, namely Merge-Split(MS) and some tranditional local search operators to generate new solutions. Among these operators, the MS operator is capable of searching using large step sizes, and thus has the potential to search the solution space more efficiently and is less likely to be trapped in local optima. At last, we use the genetic algorithm to find the best feasible solution for CARP.**

***Index Terms*—Capacitated arc routing problem (CARP), path optimization, merge-split operator, local search, genetic algorithm, metaheuristic search**

I. Introduction

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 [1], [2], etc. It is a combinatorial optimization problem that requires determining the least cost routing plan for vehicles subject to some constraints [3]. The capacitated arc routing problem (CARP), which is the most typical form of the arc routing problem, is considered in this paper. It can be described as follows: a mixed graph $G = (V, E, A)$, with a set of vertices denoted by V, a set of edges denoted by E and a set of arcs (i.e., directed edges) denoted by A, is given. There is a central depot vertex $depV$, where a set of vehicles are based. A subset $E_R \subseteq E$ composed of all the edges required to be served and a subset $A_R \subseteq A$ composed of all the arcs required to be served are also given. The elements of these two subsets are called edge tasks and arc tasks, respectively. Each edge or arc in the graph is associated with a demand, a cost. Both the demand and the serving cost are zero for the edges and arcs that do not require service. A solution to the problem is a routing plan that consists of a number of routes for the vehicles, and the objective is to minimize the total cost of the routing plan subject to the following constraints:

1) each route starts and ends at the depot;

2) each task is served in exactly one route;

3) the total demand of each route must not exceed the vehicle's capacity Q.

4)the total number of the used vehicles must not exceed the number of the original vehicles V

II. Preliminary

CARP involves seeking a minimum cost routing plan for vehicles to serve all the required edges $E_R \subseteq E$ and required arcs $A_R \subseteq A$ of a given graph $G = (V, E, A)$, subject to some constraints. Each edge (i, j) is considered as a pair of arcs $< i, j >$ and $< j, i >$, one for each direction. Each arc a is associated with four features, namely $begin(a), end(a), cost(a), demand(a)$, standing for the begin and end vertices of the arc a, the cost of the arc a and the demand of the arc a. We are given a graph $G(V, E)$ with a set of vertices (V), a set of edges (E), a set of required edges ($ER \subset E$), and a fleet of identical vehicles with a capacity of Q that is based at the depot vertex v_d ($v_d \in V$). Each edge $e = (i, j) \in E$ is represented by a pair of arcs $< i, j >$ and $< j, i >$. A required edge is said to be served if and only if one of its two arcs is included in one vehicle route of the routing plan. For the sake of simplicity, we use the term task to represent a required edge hereafter. Let n be the number of tasks, i.e., $n = |E_R|$. Each arc of a task, say u, is characterized by four elements: the source vertex (s(u)), the target vertex (t(u)), the traversal cost ($cost(u)$) and the demand ($d(u)$).

To represent a CARP solution, we assign to each task (i.e., a required edge) two IDs $(i, i+n)$ where i is an integer number in $[1, n]$, i.e., one ID for each arc of the task. We also define a dummy task with 0 as its task ID and both its head and tail vertices being the depot vertex vd. This dummy task is to be inserted somewhere in the solution as a trip delimiter.

Suppose a solution S involves m vehicle routes, S can then be encoded as an order list of $(n + m + 1)$ task IDs among which $(m + 1)$ are dummy tasks:

$$S = \{S(1), S(2), ..., S(n + m + 1)\}$$

where $S(i)$ denotes a task ID (an arc of the task or a dummy task) in the i^{th} position of S. S can also be written as a set of m routes (one route per vehicle):

$$S = \{0, R_1, 0, R_2, 0, ..., 0, R_m, 0\}$$

where R_i denotes the i^{th} route composed of $|R_i|$ task IDs (arcs), i.e.,

$$R_i = \{R_i(1), R_i(2), ..., R_i(|R_i|)\}$$

with $R_i(j)$ being the task ID at the j^{th} position of R_i.

Let $sp(u, v)$ denotes the shortest path distance between the node u and node v, the total cost of the solution S can be calculated as:

$$f(S) = \sum_{i=1}^{n+m} (cost(S(i)) + dist(S(i), S(i + 1)))$$

III. METHODOLOGY

A. General workflow

In short, our algorithm is achieved through the following operations :

$Prepare$: read in the file and find the minimum distance between any two nodes using Dijkstra

$Initialize$: use Path-Scanning algorithm and Random-Pick algorithm to initialize a set of valid solutions, which is thought of the population.

$Local\_Search$: take advantage of a novel local search operator, namely Merge-Split(MS) and some tranditional local search operators, such as single insertion , double insertion and so on to generate new solutions

$Find\_the\_best$: we use the genetic algorithm to find the best feasible solution for CARP until the stopping condition is reached.

Algorithm 1 Workflow for CARP

Require: P - a CARP instance; pop_size - population size;
Ensure: the best solution $S*$ found
pop ← population_initialization(pop_size)
while stopping condition not reached **do**
  Randomly select one solution S_1 from pop;
  new=Single_Insertion(S_1)
  new=Choose_Better(new)
  new=Double_Insertion(new)
  new=Choose_Better(new)
  new=Merge_Split(new)
  new=Choose_Better(new)
  pop ← update_population(pop, new,S_1)
end while
S* ← best_feasible_solu(pop)
return $S*$ =0

B. Detailed algorithm/model design

In preparation, we need to read in the file and find the minimum distance between any two nodes using Dijkstra or Floyd algorithm.

The following is the pseudo code for Dijkstra,

Algorithm 2 Dijkstra Algorithm for SSSP

add start node into heap
while heap is not empty **do**
  curnode=heap.poll();
  for for each child node i of curnode **do**
    if node i haven't been visited **then**
      the distance of node i = curnode.distance+ weight of edge(curnode, i)
      set node i is visited
      add node i into heap
    else
      if the distance of node i > curnode.distance+ weight of edge(curnode, i) **then**
        update distance of node i
        decrease key of node i in heap
      end if
    end if
  end for
end while=0

In initializing population, we use Path-Scanning to generate 5 quite well solutions and use RANDOM-Pick to generate 25 random solutions to obtain a population of size 30.

The following is the pseudo code for Path-Scanning,[4]

Algorithm 3 Path-Scanning

k ⟵ 0
copy all required arcs in a list $free$
while $free! = \emptyset$ **do**
  $k = k + 1; R_k \longleftarrow \emptyset;$
  $load(k) \longleftarrow 0, cost(k) \longleftarrow 0; i \longleftarrow 1$
  while $free! = \emptyset$ or $d! = \infty$ **do**
    $\overline{d} = \infty$
    for $eachu \in freeandload(k) + q_n \leq Q$ **do**
      if $d_{i,beg(u)} < \overline{d}$ **then**
        $\overline{d} \longleftarrow d_{i,beg(u)}$
        $\overline{u} \longleftarrow u$
      else if $d_{i,beg(u)} = dandbetter(u, \overline{u}, rule)$ **then**
        $\overline{u} \longleftarrow u$
      end if
    end for
  end while
end while=0

And we also define the $better(\overline{u}, u, rule)$ method accoring to the following rules:

(1) maximize the distance from the task to the depot;

(2) minimize the distance from the task to the depot;

(3) maximize the term $dem(t)/sc(t)$, where $dem(t)$ and $sc(t)$ are demand and serving cost of task t, respectively;

(4) minimize the term $dem(t)/sc(t)$;

(5) use rule 1) if the vehicle is less than half- full, otherwise use rule (2)

C. Traditional OPERATORs FOR LOCAL SEARCH

There are four commonly used move operators for CARP, namely single insertion, double insertion, swap, and 2-opt [11].

1) Single Insertion: In the single insertion move, a task is removed from its current position and re-inserted into another position of the current solution or a new empty route. If the selected task belongs to an edge task, both its directions will be considered when inserting the task into the "target position." The direction leading to a better solution will be chosen.

2) Double Insertion: The double insertion move is similar to the single insertion except that two consecutive tasks are moved instead of a single task. Similar to the single insertion, both directions are considered for edge tasks.

3) Swap: In the swap move, two candidate tasks are selected and their positions are exchanged. Similar to the single insertion, both directions are considered for edge tasks.

4) 2-opt: There are two types of 2-opt move operators, one for a single route and the other for double routes. In the 2-opt move for a single route, a subroute (i.e., a part of the route) is selected and its direction is reversed. When applying the 2-opt move to double routes, each route is first cut into two subroutes, and new solutions are generated by reconnecting the four subroutes.

D. MERGE-SPLIT OPERATOR FOR LOCAL SEARCH

The MS operator aims to improve a given solution by modifying multiple routes of it. As indicated by its name (MergeSplit), this operator is composed of two components, i.e., the Merge and Split. Given a solution, the Merge component randomly selects p(p ¿ 1) routes of it and, combines them together to form an unordered list of task IDs, which contains all the tasks of the selected routes. The Split component directly operates on the unordered list generated by the Merge,[4].

To summarize, the MS operator first merges multiple routes to obtain an unordered list of tasks, and then employs PS to sort the unordered list. After that, Ulusoy's splitting procedure is used to split the ordered lists into new routes in the optimal way. Finally, we may obtain five new solutions of CARP by embedding the new routes back into the original solution, and the best one is chosen as the output of the MS operator, [4]. Fig. 1 demonstrates the whole process of MS operator.

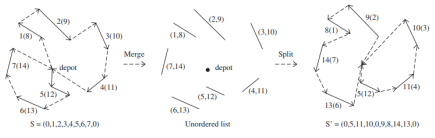

Fig. 1. the whole process of MS operator

In an easy way to understand, MS actually operates Path-Scanning on the sub-graph obtained by merging the part routes of all the routes of the original sub-graph. And the fact proves it is quite powerful !

E. Analysis

Clearly, Dijkstra is the fastest algorithm to find the shortest distance in a non negative weight graph. The time complexity of Dijkstra is $O(n+m)logn$, which n is the number of the vertices and m is the number of edges.

The time complexity of single Path-Scanning is $O(n^2)$ so the time complexity of single Merge-Split is also $O(n^2)$. And the time complexity of single traditional operations is $O(n)$

Therefore, the total time complexity of the algorithm is $O(n^2) * O(n) = O(n^3)$, which is the local search period. Therefore, it is the deciding factor of the performance.

IV. Experimental Study

A. Experimental Setup

(1)Data Set All the experiments were carried out on three benchmark test sets of CARP instances, referred to as the gdb set , the val set , the egl set . The gdb set was generated by DeArmon and consists of 23 instances. The val set was generated by Benavent et al. It contains 34 instances based on 10 different graphs. Different instances based on each graph were generated by changing the capacity of the vehicles. The egl set was generated by Eglese based on data from a winter gritting application in placeLancashire . It consists of 24 instances based on two graphs, each with a distinct set of required edges and capacity constraints.

And what the instances we choose are as followings:

$$egl-g1-A, egl-e1-A, egl-s1-Agdb1, gdb2, gdb3, gdb4$$

$$gdb10, gdb11, val1A, val4A, val10A$$

(2)Description of the Environment

- CPU: 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz
- OS: Windows 10
- IDE: Pycharm
- Python version: 3.10
- NumPy version: 1.24.2

B. Experimental Results

And the detailed instances and test results are as followings.

Name	$\|V\|$	$\|R\|$	$\|E\|$	LB	MAENS	MY	Time(s)
egl-s1-A	140	75	190	5018	5018	5340	180
egl-e1-A	77	51	98	3548	3548	3638	180
egl-g1-A	255	347	375	-	992045 [3]	1174182	300
gdb1	12	22	22	316	316	316	60
gdb2	12	26	26	339	339	339	60
gdb3	12	22	22	275	275	275	60
gdb4	11	19	19	287	287	287	60
gdb10	12	25	25	275	275	275	60
gdb11	22	45	45	395	395	395	120
val1A	24	39	39	173	173	173	120
val4A	41	69	69	400	400	407	120
val10A	50	97	97	428	428	428	120

C. Analysis

The experimental results are actually quite good. They have met my expectation.

Next, I will analyze the effect of different components and hyperparameters by comparing the results of basic utility test cases.

Fig. 2. just Path-Scanning

Fig. 3. Path-Scanning and Merge-Split

The fact proves that Merge-Split is quite efficient.

Fig. 4. PS, MS and Single-Insertion

Fig. 5. PS, MS, SI and Double-Insertion

Through the above, We find that the single insertion also has a considerable effect on the final solution, but the double-insertion has a very little improvement for the solution obtained by PS,MS,ID.

V. Conclusion

During this project, I have gained a lot.

First, I learn a novel local search operator, the MS operator in CARP, which was used in my project. Unlike existing local search operators, which only modify a small part of the candidate solutions, the MS operator may result in significantly different solutions. Hence,the MS operator is capable of searching using large step size.

Second, it is well acknowledged that a good combination of large and small step sizes can lead to better solutions to numerical optimization problems.

Third, I find that if you put your heart into the project, the project may be not that difficult!

But I Still have much to improve, such as the master of the embeded function of python and the time planning. And the biggest problem on myself is that it is hard for me to concentrate on paper reading and think what the paper says. The ability of reading paper is still to be improved!

VI. References

[1] H. Handa, D. Lin, L. Chapman, and X. Yao, “Robust solution of salting route optimisation using evolutionary algorithms,” in Proc. IEEE Congr. Evol. Comput. 2006, Vancouver, BC, Canada, pp. 3098–3105.

[2] H. Handa, L. Chapman, and X. Yao, “Robust route optimization for gritting/salting trucks: A CERCIA experience,” IEEE Comput. Intell. Mag., vol. 1, no. 1, pp. 6–9, Feb. 2006.

[3] M. Dror, Arc Routing, Theory, Solutions, and Applications. Boston, MA: Kluwer, 2000.

[4] K. Tang, Y. Mei and X. Yao, ”Memetic Algorithm With Extended Neighborhood Search for Capacitated Arc Routing Problems,” in IEEE Transactions on Evolutionary Computation, vol. 13, no. 5, pp. 1151-1166, Oct. 2009, doi: 10.1109/TEVC.2009.2023449.

[5] B. L. Golden, J. S. DeArmon, and E. K. Baker, “Computational experiments with algorithms for a class of routing problems,” Comput. Oper. Res., vol. 10, no. 1, pp. 47–59, 1983.

[6] Hertz, Alain Laporte, Gilbert Mittaz, Michel. (2000). A Tabu Search Heuristic for the Capacited Arc Routing Problem. Operations Research. 48. 10.1287/opre.48.1.129.12455.