Capacitated Arc Routing Problem Report

12110817 Zhang Zhanwei

***Abstract*—The capacitated arc routing problem (CARP) is a difficult combinatorial optimization problem that has been intensively studied in the last decades. A heuristic algorithm based on tabu search and a local refine approach Merge-Split are applied and tested on various sets of benchmark instances. The computational results show that the proposed algorithm produces high-quality results within a reasonable computing time.**

I. Introduction

The origins of arc routing can be traced back to the 18th century and the work of Leonhard Euler, a Swiss mathematician who tackled the Konigsberg bridges problem while serving as Chair of Mathematics at the St. Petersburg Academy of Sciences. Euler's efforts in solving this problem laid the foundation for modern graph theory. While several 19th-century mathematicians also showed interest in arc routing problems, it was not until 1960 that the study of modern arc routing began, with the publication of the first work on the Chinese postman problem. Since then, arc routing has grown into a comprehensive research area within combinatorial optimization, with numerous contributors over the years.

The capacitated arc routing problem (CARP) has been extensively studied in the last few decades due to its widespread use in logistics, including waste collection, product distribution, winter gritting, and postal deliveries.[1] In this model, a graph is given with vertices and edges, where each edge has a traversal cost, and a subset of edges is associated with service cost and demand for some vehicles. A fleet of identical vehicles with a limited capacity is stationed at the depot vertex, and the goal is to find a set of vehicle routes with a minimum cost such that every required edge is serviced, each route must start and end at the depot vertex, and the total demand serviced on the vehicle's route doesn't exceed its capacity.

II. Notation and solution representation

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. A heuristic algorithm for CARP

This section presents the heuristic algorithm(HA) for CARP, which includes a description of the main procedure, the approach for generating initial solutions, a specific route-based crossover technique, and a population managerment procedure. The local refinement procedure will be presented in Section 4.

A. main scheme

The HA can be considered as a steady-state evolutionary algorithm which updates only one population solution at each generation of the evolution process.[2] Algorithm 1 shows the main scheme of our algorithm. It starts with an initial population of solutions which are first generated by random path generation and further improved with the local refinement procedure.

At each generation, HA randomly selects two parent solutions S^1 and S^2 from the population and performs a route-based crossover (RBX) operation [3] to generate an offspring solution S^0 and repairs it if needed. Then apply the

local refinement procedure to improve S^0, mainly including small-step-size operators and large-step-size operators.

Finally, HA ends a generation by updating the population and the best solution.

Algorithm 1 HA for CARP

Input: P - a CARP instance; pop_size - population size;
Output: the best solution $S*$ found

```
pop ← population_initialization(pop_size)
S* ← best_feasible_solu(pop)
while stopping condition not reached do
    Randomly select two solution S^1 and S^2 from pop;
    S^0 ← crossover(S^1, S^2) /* Route-based crossover */
    S^0 ← local_refine(P, S^0)
    pop ← update_population(pop, S^0)
    S* ← best_feasible_solu(pop)
end while
return S* =0
```

B. Population Initialization

In this procedure, we do not use the path-scanning or heuristic algorithm but randomly select an unserved task that does not violate the capacity constraint. If no task is satisfiable, complete the current route by following the shortest deadheading path to the depot vertex and start a new route. Otherwise, select one satisfactory task to extend the current route. This process continues until all tasks are served.

C. Route-based Crossover Operator [3]

Given two parent solutions $S^1 = \{R_1^1, R_2^1, \ldots, R_m^1\}$ and $S^2 = \{R_1^2, R_2^2, \ldots, R_m^2\}$, crossover copies S^1 to an offspring solution S^0 and replaces a route of S^0 with a route from S^2, and then repairs S^0 to establish feasibility if needed. The RBX crossover procedure consists of three main steps:

- Step 1: Copy S^1 to an offspring solution S^0 and replaces a route of S^0 with a route from S^2. Collect the unserved tasks U and duplicated tasks D.
- Step 2: Remove the duplicated tasks. Select the task which reduces more cost after removing.
- Step 3: Insert the unserved tasks. For each t in U, Scan all possible positions of S^0 to insert t, and calculate the saving (change of the total cost). RBX finally inserts the task into a position that causes the minimum saving.

RBX has the potential to generate a solution that has a different structure from its parent solutions. Additionally, the quality of the generated solution is not compromised as a result of utilizing greedy heuristics during Steps 2 and 3. RBX can be realized in $O(n^2)$, where n is the number of tasks.

D. Population Updating

1) Fitness Function: The fitness function of a solution S contains two parts:

- Total cost of the solution $f(S)$.
- Penalty factor PF_u related to the amount of unserved demand.
  Under the limited amount of vehicles, there are some infeasible solutions in the population. In the case of a constrained optimization problem, such as CARP, it has been established that permitting a controlled exploration of infeasible solutions can assist in navigating between structurally diverse solutions and lead to the discovery of high-quality solutions that might be hard to find if the search is restricted to the feasible region.[3] However, we still need a penalty factor so that the infeasible solutions do not overly affect the health of the population.

Assuming an offspring solution S^0 that has undergone both crossover and local refinement, then calculate the fitness value for each individual in the population and subsequently remove from POP the solution S^{worst} that has the lowest fitness value.

2) Tabu Search: Tabu search is a type of metaheuristic random search algorithm that starts from an initial feasible solution and selects a series of specific search directions (movements) for trial, aiming to achieve the maximum change in the target function value. To avoid getting trapped in local optimum solutions, a flexible "memory" technique is used in TS search to record and select the optimization process that has already been carried out, guiding the next search direction.[6]

In the population updating, we maintain a tabu list containing the solutions which cause the successful population updating in the previous n iterations. In the next iteration, the offspring in the tabu list will be discarded.

IV. Local Refinement

The local refinement procedure to improve mainly includes small-step-size operators and large-step-size operators. Select one of the operations based on pre-given probability P_{ss} and P_{ms}.

A. Single insertion(SI)

It is a small-step-size operator. During a single insertion move, a task is extracted from its current position and placed into a new empty route or a different position within the existing solution. If the selected task is part of an edge task, both of its directions are examined when inserting it into the "target position". The direction that results in a superior solution will be selected.

B. Double insertion(DI)

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.

C. Merge-Split(MS) [4]

It is a large-step-size operator. After selecting a solution, the Merge component combines $p(p > 1)$ randomly chosen routes to create an unordered list of tasks that includes all of the selected routes' tasks. The Split component then utilizes the unordered list generated by Merge. First, it applies the path scanning (PS) heuristic [5]. PS begins by initializing an empty path and then, at each iteration, finding tasks that don't violate the capacity constraints. If no task adheres to the restrictions, the end of the path links to the depot with the shortest path to form a route, and a new empty path is initialized. If a single task follows the limits, PS connects that task to the end of the current path with the shortest path. If there are multiple tasks that satisfy the constraints, the one closest to the end of the current path is selected. If there are multiple tasks that satisfy both the capacity constraints and are similarly close to the end of the current path, the split component employs five rules to determine which one to choose:

1) maximize the distance from the head of task to the depot
2) minimize the distance from the head of task to the depot
3) maximize the term $d(t)/cost(t)$, where $d(t)$ and $cost(t)$ are demand and serving cost of task t, respectively;
4) minimize the term $d(t)/cost(t)$
5) use rule 1) if the vehicle is less than half full, otherwise use rule 2).

Once all the tasks in the unordered list have been chosen, PS terminates. It's worth noting that PS doesn't utilize the five rules in a rotational manner. Instead, it scans the unordered list of tasks five times - each time using only one rule. As a result, PS will create a total of five ordered task lists and choose the best task list to re-insert into the solution. PS can be realized in $O(n)$, where n is the number of tasks.

V. Computational Experiments

A. Datasets Description [3]

191 classical instances are very popular and widely used in the CARP literature. They cover both random instances and real-life applications, and are typically classified into eight sets:

- gdb: 23 instances randomly generated by DeArmon, with 7-27 nodes and 11-55 required edges.
- val: 34 instances derived from 10 randomly generated graphs proposed by Benavent et al., with 25-50 nodes and 34-97 required edges.
- egl: 24 instances proposed by Eglese, which originate from the data of a winter gritting application in Lancashire (UK), with 77-140 nodes and 98-190 edges that include 51-190 required edges.
- C : 25 instances generated by Beullens et al. based on the intercity road network in Flanders, with 32-97 nodes and 42-140 edges that include 32-107 required edges.
- D: 25 instances modified from the instances of set C by doubling the vehicle capacity for each instance.
- E: 25 instances, also generated by Beullens et al. based on the intercity road network in Flanders, with 26-97 nodes and 35-142 edges that include 28-107 required edges.
- F: 25 instances modified from the instances of set E by doubling the vehicle capacity for each instance.
- EGL-G: 10 large-sized CARP instances, which like the set egl, were also generated based on the road network of Lancashire (UK) [4], each having 255 nodes and 375 edges with 374 to 375 required edges.

B. Environment

- CPU: AMD Ryzen 5 2600 Six-Core Processor
- OS: Windows 10
- IDE: VSCode
- Python version: 3.8
- NumPy version: 1.23.5

C. Parameters

Name	Description	Value
pop_size	population size	20
PF_u	Penalty factor related to unserved demand	1.5
p	Number of routes involved in Merge-Split operator	2
P_{ss}	probability of small-step-size operator	0.7
P_{ms}	probability of merge-split operator	0.3

D. Comparative results

The table displays the cost of the final solutions of some datasets. Here are a description of the columns:

1) $|V|$, $|R|$, $|E|$ indicate the number of vertices, required edges, and total edges.
2) LB indicates the lower bounds found so far for the instances.
3) MAENS column indicates the result from an existing algorithm. [4]
4) MY column indicates the result of my implementation.
5) Time column indicates the running time.

Name	$\|V\|$	$\|R\|$	$\|E\|$	LB	MAENS	MY	Time(s)
egl-s1-A	140	75	190	5018	5018	5181	180
egl-e1-A	77	51	98	3548	3548	3548	180
egl-g1-A	255	347	375	-	992045 [3]	1273031	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	406	120
val10A	50	97	97	428	428	428	120

For the small-size instances and some median-size instances, the performance of the implementation is almost comparable to the best results reported in the literature.

However, for large-size instances, his performance is still 20%-30% away from LB. The reasons may be:

- Infeasible solutions are not dealt with more carefully.
- Calculation time is not long enough.
- There are still fewer ways to operate on a solution.
- Easy to fall into local optimal solutions.

VI. Conclusion

In conclusion, the capacitated arc routing problem (CARP) is a challenging combinatorial optimization problem that has been extensively studied in the last few decades due to its frequent use in logistics industries. In this paper, we proposed a heuristic algorithm based on tabu search and a local refinement approach Merge-Split to solve the CARP problem. The computational results show that the proposed algorithm produces high-quality solutions within a reasonable computing time.

The proposed algorithm uses a steady-state evolutionary approach with the route-based crossover operator, which was effective in generating a structurally different offspring solution from its parent solutions. The population initialization method is simple and effective, and the fitness function includes a penalty factor to avoid the impact of infeasible solutions in the population.

The local refinement procedure includes a small-step-size operator and a large-step-size operator. Merge-Split is a powerful large-step-size operator that combines the results of multiple random routes, and path-scanning techniques are used to find optimal routes among the unordered list of tasks. SI and DI are single and double insertion moves that generally lead to local improvements in the solution.

Overall, the proposed algorithm is highly effective in solving capacitated arc routing problems, and its performance is comparable to the best results reported in the literature. The algorithm can be further extended by incorporating other heuristics and metaheuristics, and it can be applied to solve real-world inspiring applications in logistics and transport industries.

VII. References Section

[1] Dror, M. Arc Routing: Theory, Solutions and Applications. Kluwer Academic Publishers 2000.

[2] Glover F., Kochenberger G. (Eds.). Handbook of Metaheuristics, Kluwer, Norwell, Massachusetts, USA. 2003.

[3] Chen, Yuning et al. “A hybrid metaheuristic approach for the capacitated arc routing problem.” Eur. J. Oper. Res. 253 (2016): 25-39.

[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.