Capacitated Arc Routing Problems AI Report

Zinan Li
12011517

***Abstract*—This article describes the principles, implementation, and experimental results of a genetic algorithm aiming to solve Capacitated Arc Routing Problems(CARP).**

I. Introduction

Capacitated Arc Routing is a type of optimization problem that involves finding the most efficient routes for a fleet of vehicles to deliver goods to a set of customers. This problem arises frequently in logistics and transportation, where companies must find the most cost-effective way to deliver goods while also considering constraints such as vehicle capacity and time windows.

The history of Capacitated Arc Routing can be traced back to the 1960s, when researchers first began studying routing problems in transportation. Early solutions to these problems relied on linear programming and other mathematical techniques, but as the size and complexity of these problems grew, more sophisticated methods were needed. In the 1990s, researchers began exploring the use of genetic algorithms for solving Capacitated Arc Routing problems, and these methods have since been applied to a wide range of real-world applications.

Genetic algorithms are a type of optimization technique that is inspired by the principles of natural evolution. These algorithms involve generating a population of potential solutions, evaluating their fitness, and iteratively applying local search such as selection, crossover, and mutation to improve the solutions over multiple generations. This process allows for efficient exploration of large solution spaces and can often find near-optimal solutions to complex problems.

In this paper, I implement a solution to Capacitated Arc Routing using genetic algorithms. Our approach leverages the flexibility of these algorithms to find efficient routes for a fleet of vehicles while also considering constraints such as vehicle capacity. We demonstrate the effectiveness of our approach through a series of experiments and compare our results to other existing methods for solving Capacitated Arc Routing problems. Our findings show that our genetic algorithm-based approach is able to find high-quality solutions to these problems in a computationally efficient manner.

II. Preliminary

In this paper, I aim to solve the Capacitated Arc Routing problem using genetic algorithms. In order to formally define the problem and introduce the necessary terminology and notation, we begin by stating the problem as follows:

Given a directed graph $G = (V, E)$, where V is the set of vertices and E is the set of arcs, and a central depot vertex $dep \in V$, where a set of vehicles are based. Each arc $(i, j) \in E$ has an associated cost $c_{ij} > 0$ and a demand $d_{ij} \geq 0$. T is the set of non-zero-demand arc $(i, j) \in E$, $T \subseteq E$. The route r is an ordered set of non-zero demand arcs and represents the arcs that a vehicle needs to serve. The goal is to find a set of routes $R = r_1, r_2, \ldots, r_n$ such that the total cost of all routes is minimized while satisfying the following constraints:

- For each $(i, j) \in T$, there exists exact one route $r_k \in R$ such that $(i, j) \in r_k$
- For each route $r_k \in R$, the total demand of all arcs in r_k cannot exceed the vehicle's capacity Q
- For each route $r_k \in R$, the starting and ending vertices must be the depot vertex dep

$dis(i, j)$ represent the minimum cost from vertex i to vertex j, which can be calculate by Floyd algorithm. $cost(r)$ represent the minimum cost from depot to visit all arc $(i, j) \in r$ in order and back to depot.

Thus, the Capacitated Arc Routing problem can be formulated as the following optimization problem:

$$\min_{R} \sum_{r_k \in R} cost(r_k)$$

Subject to:

$$\sum_{r_k \in R} [(i, j) \in r_k] = 1, \forall (i, j) \in T$$

$$\sum_{(i,j) \in r_k} d_{ij} \leq Q, \forall r_k \in R$$

$$dep = r_k[1], dep = r_k[|r_k|], \forall r_k \in R$$

In the following sections, I will present a genetic algorithm approach to solving this problem.

III. Methodology

The proposed genetic algorithm for the Capacitated Arc Routing problem will consist of the following parts:

A. Preprocessing

According to the requirements of this project, we need to read data from a file. I store the cost and demand of each arc in a map whose key is a two-tuple representing the arc. Then I use the Floyd algorithm to calculate the minimum distance between any two vertices and store the result in a two-dimensional array.

B. Initialize the population

In this section, I use the path-scanning to find the initial solutions. Path-scanning is a heuristic method for finding a solution to the CARP in which a path for each vehicle is found by starting at the depot and selecting the next non-zero-demand arc to traverse based on the minimum distance from the current node. If there are multiple arcs with the same minimum distance, we will pick one at random. This process is repeated until the vehicle reaches its capacity or there are no more arcs to traverse. The initial population without duplicates is repeatedly generated using this method until the maximum number of iterations is reached.

C. Apply local search

In common genetic algorithms, the offspring is typically produced by applying crossover operation in two parents selected by a fitness-based selection. However, CARP is a restricted problem, and the crossover operation has great possibilities for producing ineligible offspring. The ineligible offspring need to be modified by deleting duplicate arcs and inserting missing arcs. After conducting my experiments, I discovered that crossover has high time consumption and poor results. Therefore, I deprecate the crossover operation. Instead, I use mutation(local search) to produce the offspring (the possibility of mutation is 100%), and if the mutation does not replicate an already existing solution, I add it to the population. Each generation produces a set number of offspring equal to the set population size.

The mutation operation will randomly choose a following method to modify the individual.

1) Flip: In the flip operation, a random arc in the individual is selected, and its order is reversed. This operation can help explore different solutions and potentially improve the fitness of the individual. As the image 1 shows, the arc $(5, 6)$ changes into $(6, 5)$ after the flip operation.

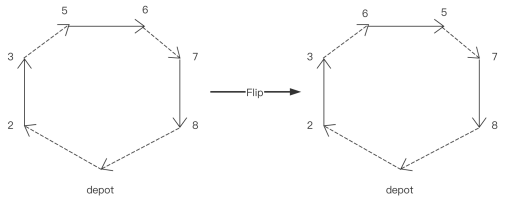

Fig. 1. Flip

2) Single insertion: The single insertion operation is performed by first randomly choosing an arc from a route, then identifying a potential insertion point in the existing route and calculating the cost or benefit of inserting the new arc at that point. The new arc is then removed from the origin route and inserted into the route at the chosen insertion point, and the overall performance of the route is evaluated to determine if the insertion has improved the solution. The single insertion may produce an ineligible individual, which will be discarded. As the image 2 shows, the arc $(7, 9)$ is inserted into another route.

Fig. 2. Single insertion

3) Double insertion: The double insertion is mostly the same as the single insertion. The difference is that double insertion chooses and inserts two contiguous arcs at a time. As the image 3 shows, the arc (7,9), (3,5) is inserted into another route with the same order.

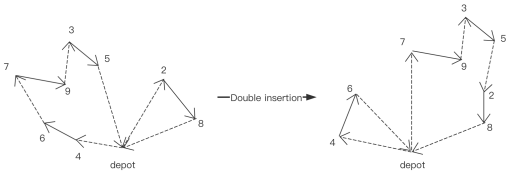

Fig. 3. Double insertion

4) 2-opt for single route: In 2-opt for single route, the route is improved by reversing pairs of arcs in a route in order to reduce the overall cost of the route. It is performed by randomly choosing a subroute (a part of a route) from a route and reversing it. As the image 4 shows, the subroute $(5, 6), (7, 8)$ is reversed after the operation.

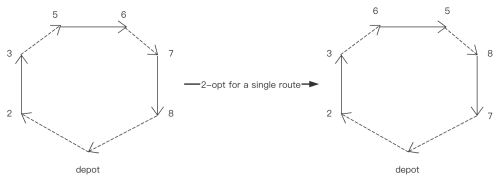

Fig. 4. 2-opt for single route

5) 2-opt for double route: In a 2-opt algorithm for double routes, the route is improved by swapping sets of arcs between two routes in order to reduce the overall cost of the route. The operation first chooses two routes and splits each of them into two parts. It then swaps their latter part and checks if they still meet the restriction. Each part of two route can be reversed or not, so there exist $C_3^1 * C_3^1 = 9$ possible results. The operation will calculate the total cost of each result and take the best one. As the image 5 shows, the subroute $(5, 6), (7, 8)$ and $(11, 12)$ is reversed and swapped.

D. Select the fittest solutions

The fitness of each individual is represented by its total cost. The smaller the cost, the better. In order to reduce the time

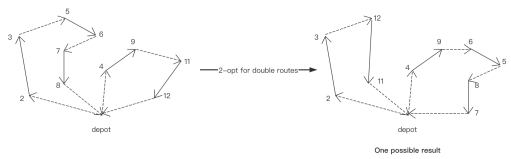

Fig. 5. 2-opt for double route

consumption of sorting, I use the *heapq.nsmallest()* function to heapsort the individuals in the population by their cost and find the n smallest individuals where n equals the preset population size. These individuals will be the next generation.

E. Implementation of genetic algorithm

After testing, I found that the number of generations is more important than the population size. Therefore, I adopt the strategy of a large initial population and a small population size to ensure evolution speed. I set the population size to be 5 and the initial population to be 100.

Algorithm 1 Genetic algorithm

Input: Map g, depot d, population size $psize$, time t
Output: Best solution s

```
//Generate initial population
population = []
for i = 1 → 100 do
    solution ← PathScanning(g, d)
    if solution ∉ population then
        population ← population ∪ solution
    end if
end for
//Evolution
while running time not exceed t do
    for i = 1 → psize do
        Random select an individual x from population
        Produce x̄ by applying a mutation operation to x
        if x̄ is valid and x̄ ∉ population then
            population ← population ∪ x̄
        end if
    end for
    Using heapsort find the psize smallest solution as p
    population ← p
end while
return population[0] //population is ordered
```

F. Algorithm analysis

The time complexity of the preprocess is $O(n^3)$ since it uses nested for loops to compute the shortest distances between all pairs of vertices using the Floyd-Warshall algorithm. It is difficult to determine the exact time complexity of the genetic algorithm because the time complexity of a genetic algorithm depends on several factors. In general, however, we can say that the time complexity of a genetic algorithm is typically $O(n * g)$, where n is the population size and g is the number of generations.

IV. Experiments

A. Experiments setups

The test was conducted using my own Macbook Pro, whose CPU is Intel Core i5-8257U with Python3.9.13 and Numpy1.23.3.

To assess the algorithm's efficiency, define R as the number of generations per second. Let the algorithm with different population size play a total of three rounds at 7 test maps with time limit 10 second, respectively, and then count their R and final solution.

I plot the evolution of the population's minimum cost and average cost for each generation in order to more clearly observe the population's evolution. In this part, the time limit is 10 second and population size 5.

B. Experiments results

The test result is shown below:

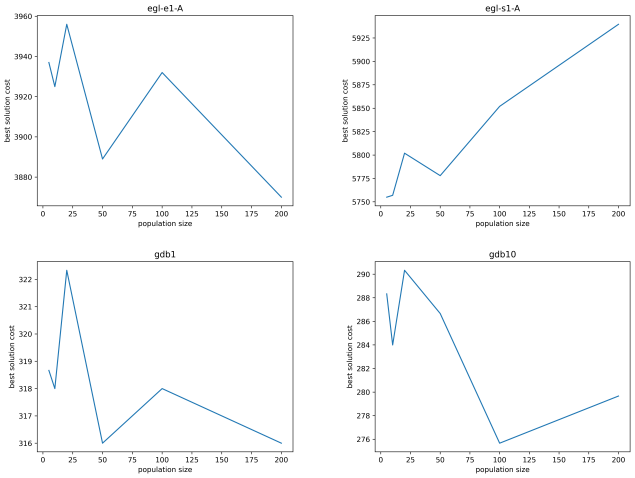

Fig. 6. Solution cost in different population size part1

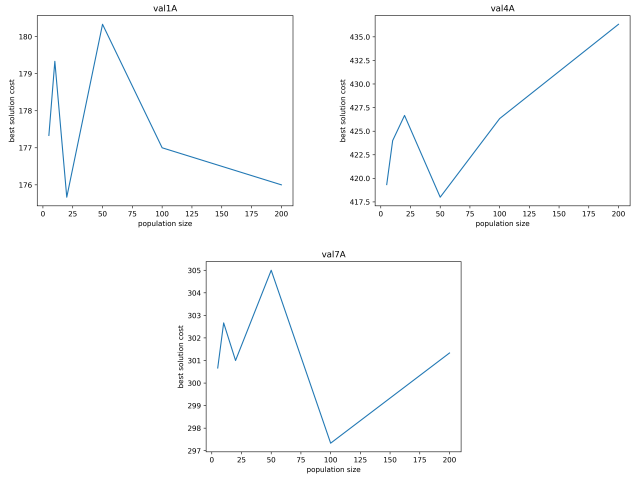

Fig. 7. Solution cost in different population size part2

Experimental results on other maps are highly similar to image 8 and are not shown here.

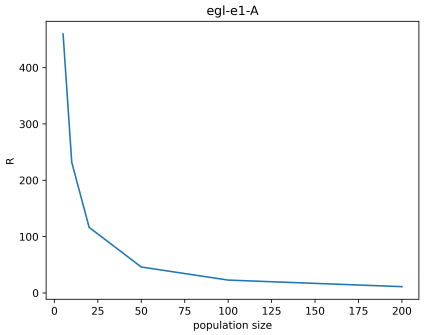

Fig. 8. R with different population size

TABLE I
ALGORITHM EFFICIENCY WITH POPULATION SIZE 5 AND TIME 10S

Map	egl-e1-A	egl-s1-A	gdb1	gdb10	val1A	val4A	val7A
R	447	323	853	785	479	322	342

Using crossover to produce the offspring will result in an 80% reduction in R in general.

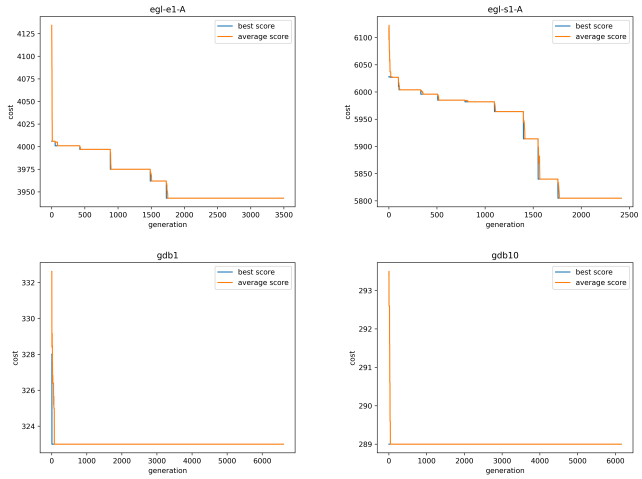

Fig. 9. Evolutionary process part1

C. Analysis

From the image 6 and 7 we can see there do not exist a best population size for all maps. Considering that the quality of the initial population has a great influence on the results of the genetic algorithm, **the result on smaller map is less stable**. I mainly referred to the results for the largest graph in the experiment(egl-s1-A). Besides, the image 8 shows that R **decreases rapidly with the increase of population**. All things considered, I decide to choose 5 as the final population size.

From the image 9 and 10 we can see cost decreases in a stepwise manner. The average score of the population lags a bit behind the minimum score in population. **The algorithm is more likely to fall into local optimum in the case of small data size** like gdb1 and gdb10. In these case, the algorithm converge in the first few generations. Larger local search algorithms might be helpful in this case.

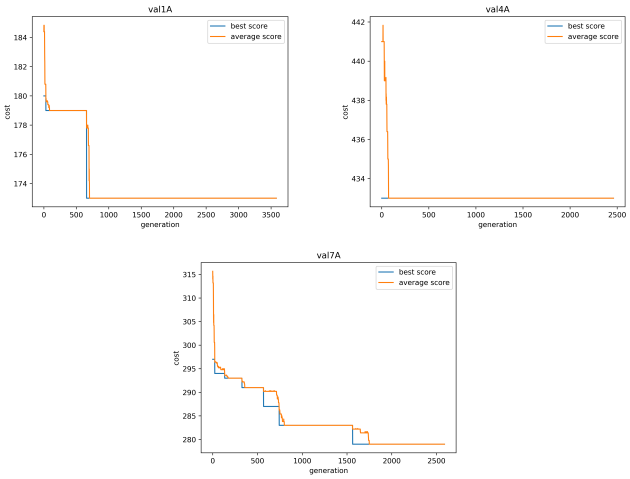

Fig. 10. Evolutionary process part2

V. CONCLUSION

In this project, I implement a genetic algorithm to solve the Capacitated Arc Routing problems. Through a series of experiments, I explore the effect of each argument on the algorithm and chose the best one based on the experimental data.

One possible extension to the approach is to incorporate additional constraints into the optimization problem. For example, restrictions on the number of vehicles that can be used on a given route. In order to handle these constraints, we could modify the genetic algorithm to incorporate penalty functions for infeasible solutions. This would allow us to find solutions that satisfy the additional constraints while still minimizing the total cost of the routes.

Another potential direction for future work is to investigate the use of other optimization techniques, such as reinforcement learning, for solving Capacitated Arc Routing problems. Reinforcement learning algorithms, which learn from experience by trial and error, have been shown to be effective for solving complex optimization problems. By incorporating reinforcement learning into our approach, we may be able to improve the efficiency and robustness of our solution.

In summary, my genetic algorithm-based approach for solving Capacitated Arc Routing problems has been shown to be effective and computationally efficient. By incorporating additional constraints and exploring alternative optimization techniques, I believe that our approach can be further improved and applied to a wider range of real-world scenarios.

REFERENCES

[1] Tang, Ke, Yi Mei, and Xin Yao. "Memetic algorithm with extended neighborhood search for capacitated arc routing problems." IEEE Transactions on Evolutionary Computation 13.5 (2009): 1151-1166.