Capacitated Arc Routing AI Based on Genetic Algorithm with Heuristic Approach

Luming Yang
Department of Computer Science and Engineering
Southern University of Science and Technology
yanglm2020@mail.sustech.edu.cn

***Abstract*—This paper introduces an innovative approach to address Capacitated Arc Routing Problems (CARP) through the integration of a genetic algorithm with heuristic methods. CARP finds practical application in various domains, including urban garbage collection, snow removal, and road surveys.**

Our algorithm, featuring a unique crossover operator designed to reduce similarity and a shuffle process to counter local minima, demonstrates promising performance. Additionally, the algorithm incorporates several optimization techniques to enhance its results.

We conduct an in-depth analysis of the algorithm's time complexity, and an experiments across five diverse datasets, which have produced positive outcomes, highlighting the algorithm's effectiveness.

While our algorithm exhibits promise, further research and fine-tuning are warranted to address stability concerns. Nevertheless, our work contributes to the ongoing endeavor to provide robust solutions for CARP, offering valuable insights and a basis for future advancements in this field.

***Index Terms*—CARP, Heuristic Approach, Genetic Algorithm, Improvement, Optimization Techniques**

I. Introduction

In logistics and transportation, it is often necessary to complete the task of finding the most effective way to transfer goods or resources from the source to the destination. This is the routing problem. It involves making decisions about the optimal path, sequence, and allocation of resources to minimize costs or maximize efficiency while adhering to various constraints and objectives. These problems are crucial in industries such as delivery services, supply chain management, and transportation planning. Solving routing problems often involves mathematical optimization techniques and the use of algorithms to find the best routes and schedules.

The field of Routing Problems can be categorized into two primary directions: the Vehicle Routing Problem [1] (VRP), where individual points are served as objects, and the Arc Routing Problem [2] (ARP), which focuses on arcs as the service objects. Unlike VRP, ARP's defining characteristic is that vehicles initiate their routes at depots and traverse the edges that require servicing, rather than serving at specific vertices.

The Capacitated Arc Routing Problem [3] [4] (CARP) falls within the realm of ARP. ARP encompasses a range of problems, including single-vehicle ARP variants like the well-known Chinese Postman Problem. CARP, on the other hand, represents a multi-vehicle ARP problem and is one of the most widely studied among them. There are many variations of this problem, including stochastic CARP and uncertain CARP. [5] CARP involves scenarios where vehicles, initially stationed at one or more depots, provide services across a network of roads.

These services can encompass various tasks, such as urban garbage collection [6], snow removal, salt spreading on icy roads, and road surveys. The combination of evolutionary algorithms and weather information systems can play a very good role in optimizing salting out routes. [7] Consequently, CARP holds significant practical relevance and demands the development of highly efficient algorithms to provide cost-effective solutions.

In a nutshell, CARP can be described as follows: It entails an undirected graph in which certain edge sets require servicing, each service possesses a specific demand, and other edges do not necessitate servicing. Each edge has an associated cost, whether it requires servicing or not. Multiple vehicles, each with limited capacity, are tasked with minimizing the overall cost to complete all required services. The detailed problem formulation will be expounded upon later in this report.

In most practical applications, some unexpected events will always occur when the vehicle is driving. Changes in conditions make the carp problem dynamic. However, the traditional CARP algorithm cannot be used to solve dynamic carp problems. [8]

This project delves into the development of a novel AI solution, combining heuristic search methods and genetic algorithms, to address CARP efficiently within specified time constraints. Historically, many heuristic algorithms have been proposed to solve the carp problem. [9] [10] [11] With the development of algorithms, more meta-heuristic algorithms have been proposed. [12] [13] [14], algorithms such as tabu search have achieved very good results. [15] For the uncertainty CARP problem, there are also policies that take into account interpretability and have achieved impressive results. [16] This algorithm combines operators from some excellent heuristic algorithms that have been proposed, and adds its own operators at the same time, achieving good results. In the following sections of this article, we will discuss the problem formulation, the methodology for developing the AI, strategies for evaluating AI performance, and experimentation techniques to enhance the AI's capabilities.

II. Preliminary

The brief description in Introduction has roughly explained the basic content of CARP. In the Preliminary, the input and output, and the standard problem formula will be described.

A. Problem Input

Given an undirected connected graph $G = (V, E)$, a set of Service $S \subseteq E$. Each vehicle has capacity Q. All vehicles have a common starting point depot. Each edge $e(u, v) \in E$ incurs a cost C_e whenever a vehicle travels over it or serves it. Each required service $s \in S$ has a demand $d(s) > 0$ associated with it.

B. Problem Formulation

The ultimate goal of CARP is to minimize the cost of the total distance traveled by all vehicles, so there are the following objective functions:

$$f(R) = \min(\sum_{i=1}^{m} cost(r_i)) \tag{1}$$

R is a solution to CARP, including m vehicles, routes r for one vehicle. Each vehicle shall not have a maximum capacity exceeding Q during driving. The cost of a route includes the cost of all sides of the route. Every service S should only be serve once. Therefore, there are the following constraints:

$$\sum_{i=1}^{|S_r \subseteq S|} d(s_{r,i}) \leq Q \quad \forall r \in R \tag{2}$$

$$s_{r,i} \notin r_j \quad \forall s \in r_i, i \neq j \tag{3}$$

The cost calculation of each route is described by the sum of all road sections that the vehicle passes through. For the service section, the cost of the section itself is calculated; otherwise, the cost of the shortest path between two points is calculated:

$$cost(r_i) = \sum_{i=1}^{|r_i|} minpath(u_{r_i} v_{r_i}) + \sum_{i=1}^{m} C_{S_i} \tag{4}$$

$$minpath(u, v) = \min(\sum_{e} C_{e(u,v)} \quad e \in P(u, v)) \tag{5}$$

Where $P \subseteq E$ is any feasible path from vertex u to v.

Finally, the answer of the problem which needs to figure up is the minimum cost $f(R)$ and the order of all services and routes R served by vehicles.

III. Methodology

The AI presented in this paper leverages the power of a genetic algorithm. Within the framework of this genetic algorithm, several key modules are employed, one of which is the initialization module. In instances where there's a potential risk of encountering a local minimum, the algorithm triggers a shuffle operation. Following the shuffle operation and the generation of a new population, the primary optimization operation is executed (as illustrated in Figure 1).

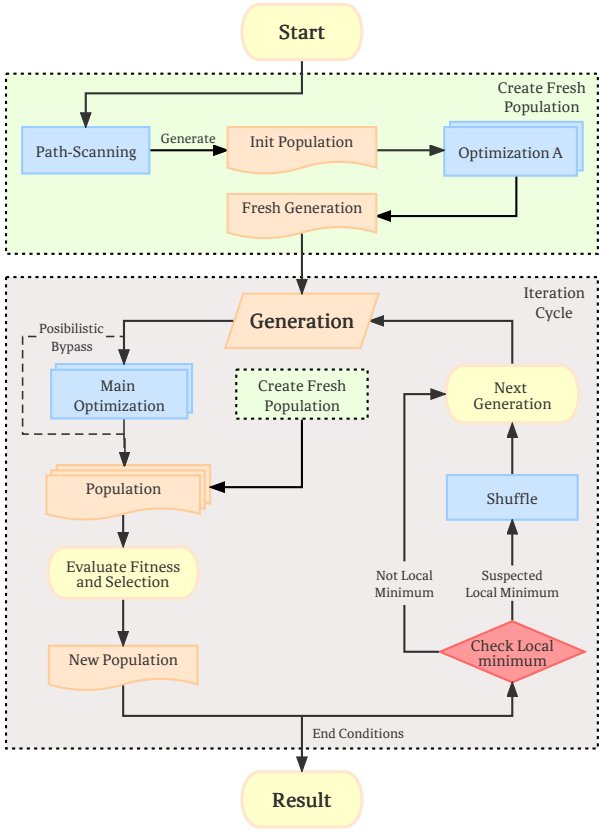


Fig. 1. A demonstration of the algorithm, which shows the workflow of general program and generic algorithm. Green part represents procedure of creating fresh population, which is used again in iteration cycle.

In the algorithm, each population size is $size(p)$, including $size(p)$ individuals. The population size may vary in different parts of the process. During the evaluate fitness and cut process, redundant individuals are eliminated.

A. Create Population

The first part of the algorithm is to create a new population, namely init population. This part uses the improved path scanning [17] method to construct a feasible solution with basic advantages.

The path scanning method [18] is to complete the solution construction by adding a service node each time and constructing the route with the most demands one by one. In the process of selecting a new node each time, path scanning will select the scheme that the current road can still bear and has the lowest cost before reaching the target. If there is a solution that meets both of the above conditions, path scanning will use comparison rules (Better Rule) to decide which path to choose for service.

Generally speaking, there are five kinds of Better Rules. Select one rule each time to generate the basic population. But this algorithm uses random rules to determine rules. The program has σ_s probability generation paths using the same comparison strategy, and $1 - \sigma_s$ probability generation paths

using completely random comparison strategies in five rules. σ_s is very small, as long as there are several individuals in the initial population. A small σ_s helps the program find better results.

B. Optimization Operators

This procedure uses a variety of operators to optimize the answers. These operators include flip operators, single insertion operators, 2-opt operators, swap operators, etc. The algorithm uses different combinations of operators to obtain better answers.

1) Flip operator [19]: The operation process of the flip operator is to traverse all routes of a solution, judge all edges on the route, and determine whether the best effect can be achieved if the edge is flipped.

2) Single Insertion operator [20]: The operation process of this operator is to insert a service edge to another legal location, and the insertion to this location does not affect the overall demand less than capacity. In this case, if lower cost can be achieved, the operator is completed.

3) 2-opt operator [21]: The operation process of this operator is to select any route segment in a solution and rotate the whole segment. This process can be understood as winding a rope in a circle.

It should be noted that this operation is equivalent to the flip operator when the fragment length is equal to 1. Therefore, 2-opt operator does not need to be used with flip operator. However, 2-opt operator must be used multiple times, because after a flip operation, there may be new operations that can optimize the solution.

C. Mutation Operators

1) Swap operator: Swap operator selects a content in a route and exchanges it with a content in another route. This exchange process must be carried out under the condition of meeting the requirements of capacity.

It should be noted that in this algorithm, the swap operator, as the mutation operator, does not need to limit the optimization of cost. That is, even if the swap operation makes the final result worse, the algorithm will accept this operation.

2) Crossover operator: The function of the crossover operator in this algorithm is to regard route as a chromosome and exchange the order of services in two different feasible solutions to achieve the crossover function of exchanging chromosomes.

First, select a random exchange object, and find the closest chromosome in another solution set according to the exchange object. For random segmentation of this chromosome, the contents after segmentation shall be rearranged according to the order of new chromosomes, and finally a new chromosome will be obtained.

In pseudo code, R_1 and R_2 represents two solutions which passed into crossover operator. S_{new} represents a solution with cost which is the result of crossover operator. In the process, if a service appears in r_{11} but also in r_{22}, it will be deleted from r_{22}; If a service appears in r_{12} but does not appear in r_{22}, the service will be reinserted into rnew at the best location, that is, the location with the lowest cost.

Algorithm 1 Crossover Operator

Require: R_1, R_2
  $r_1 \leftarrow$ Random select route in R_1
  $cut_1 \leftarrow$ Rand(length(r_1))
  $r11 \leftarrow$segment of r_1 before cut_1
  $r12 \leftarrow$segment of r_1 after cut_1
  for route r_2 in R_2 **do**
    if has more same Service to r_{12} **then**
      $Choose \leftarrow r_2$
    end if
  end for
  for edge u in r_2 **do**
    if u not in r_1 OR u in r_{11} **then**
      Remove u from r_2
    end if
  end for
  $r_{new} = r_{11} + r_{22}$
  for edge u in r_{12} **do**
    if u not in r_{22} **then**
      $bestPos \leftarrow MinimumCostPosition(r_{new}, u)$
      Insert u into r_{new} at $bestPos$
    end if
  end for
  $C_{new} \leftarrow CalculateRouteCost(r_{new})$
  Update R_{new} with R_{new}
  Update S_{new} with R_{new}, C_{new}
  return S_{new}

D. Shuffle Operation

Shuffle operation is a remedial measure taken when the population of program operation may fall into the local minimum. The operation is divided into two parts. The first part is to execute the Mutation Operator once for all the individuals in the population. The second part is to insert a batch of new individuals into the population, which are generated by the create population part.

E. Algorithm Analysis

Given that our algorithm operates within the Python architecture, it's worth noting that the time complexity of fundamental operations can be relatively high. Therefore, the time complexity analysis provided here serves as a general reference, with the understanding that specific time complexity assessments may necessitate tailored investigation for precise evaluation.

It's important to underscore that this algorithm, rooted in genetic algorithms and heuristic techniques, is primarily designed for NP-complete problems. Consequently, its primary objective is to deliver feasible solutions rather than aiming for a globally optimal solution. In practice, for smaller-scale instances, the algorithm often does yield global optimal solutions. However, in scenarios involving larger problem sets,

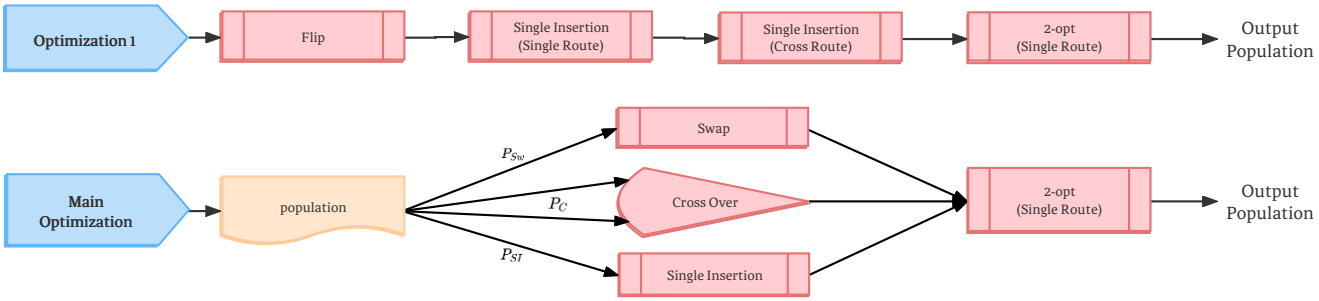


Fig. 2. Workflow of Optimization A(1) and main optimization progress. Red rectangle in the flowchart represents operators.

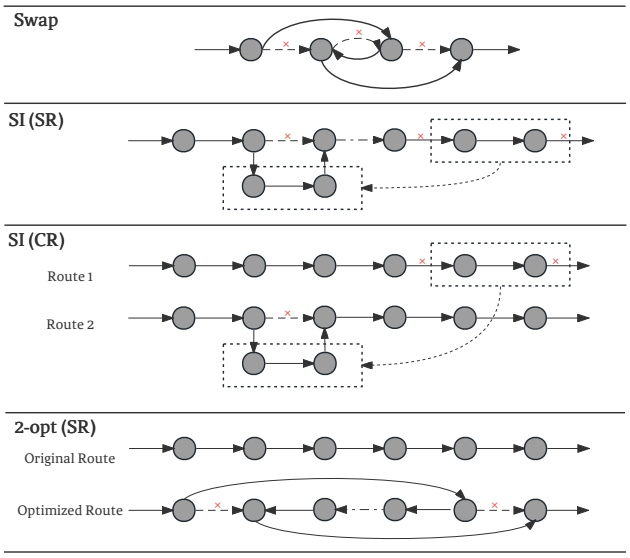


Fig. 3. Workflow of Optimization 1 and main optimization progress. Red rectangle in the flowchart represents operators.

the algorithm typically converges towards local optima or solutions that are relatively close to optimal.

TABLE I
TIME COMPLEXITY OF ALGORITHM FUNCTIONS

Function	Complexity	Per Generation
Floyd	$O(n^3)$	-
Path Scanning	$O(m^2)$	$O(m^2 p_{new})$
Flip operator	$O(m)$	$O(m)$
2-opt operator	$O(mk^2)$	$O(mk^2 p)$
duplication-check	$O(pm)$	$O(p^2 m)$
duplication-check(hash)	$O(p)$	$O(p^2)$
crossover operator	$O(m+k^2)$	$O(P_c p(m+k^2))$
Swap operator	$O(1)$	$O(P_{sw} p)$
SI/SSI operator	$O(k)$	$O(P_{si} pk)$

This algorithm includes several modules, and we will analyze the complexity of the algorithm by modules.

Table 1 lists the worst-case complexity of all algorithms. In the table, n represents the number of points in Vertex set V. The number of services required is $m = |s|$. k represents the number of services performed by a vehicle, and r represents the number of vehicles in the solution set.

Considering that the demand is usually not too close to the demand in more complex cases, the average time complexity in complex cases can be calculated by substituting the following equation:

$$m = rk, \quad r = k = \sqrt{m} \tag{6}$$

When the input data degenerates into a line, the equation becomes $m = k$.

Therefore, it can be analyzed that in addition to the fixed data size change such as the size of edge set and point set, the complexity of the algorithm in the program focuses on the number of service edges served by each vehicle. Under the same level of edge set data, if the vehicle capacity becomes unusually large or the demand becomes very small, the algorithm will degenerate and the operation time will be slow. In addition, the preset population size, new population size and other parameters will also affect the running time.

IV. EXPERIMENT

A. Setup

1) Data Set: In the experiment, a total of 5 data sets were used. It includes two data sets from gdb [22], two data sets from egl, and one data set from val [23]. The egl benchmark instance is the famous CARP benchmark. It was generated by Eglese based on data from Lancashire. [24] [25] These datasets have their own characteristics, such as gdb1 and gdb10. Although their edge sets and point sets are not significantly different in number, the undirected graph content is quite different. The characteristics of various data sets and the reasons for their use in testing will be described later. In these datasets, egl-e1-A is used most frequently.

2) Testing Environment:

1) Operation System: Windows 10 21H2
2) CPU: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
3) RAM: 16.0 GB
4) Python version: 3.9

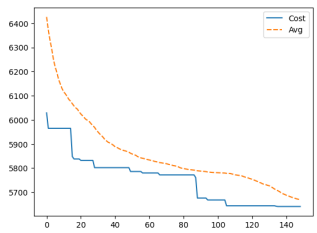
TABLE II
STATISTICS OF DATA-SETS USED

Data Set Name	Vertices	Required Edges
gdb1	12	22
gdb10	12	25
val7A	40	66
egl-e1-A	77	51
egl-s1-A	140	75

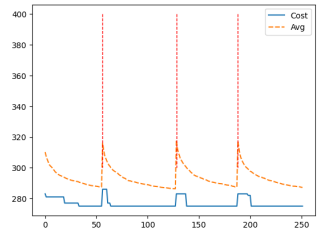
B. Experiment content

The experimental phase consists of two primary components.

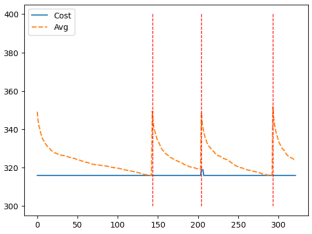
In the first part, we conducted tests using five distinct datasets and meticulously recorded the evolution of relevant data over generations. Our graphical representation of the results featured three distinct lines: a red line, a blue line, and an orange line. The blue line, indicative of the cost, represents the value associated with the individual having the lowest cost within the current population. Meanwhile, the orange line signifies the average cost, reflecting the mean cost across all individuals in the current population. The red line, as mentioned earlier, corresponds to the shuffle process – each instance of a red line signifies an automated shuffling event. These graphical representations enable a comprehensive understanding of the evolution of each curve.



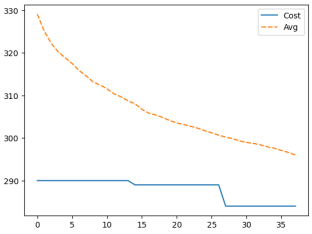
(a) egl-s1-A (60s)



(b) gdb10 (20s)



(c) gdb1 (20s)



(d) val7A (20s)

In the second part of our experimentation, we introduced different random seeds into the program, generating seven distinct runtime cost curves for the same input. These costs represent the minimum cost among all individuals in the population, essentially identifying the optimal solution. By overlaying and comparing the performance of various random seeds on the same input, this part of the experiment serves as a robust evaluation of the algorithm's stability and resilience.

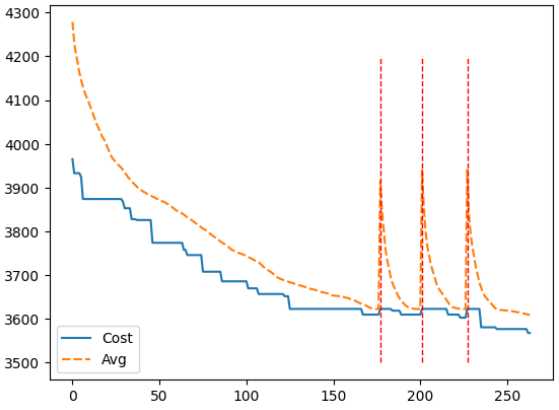


Fig. 4. egl-e1-A

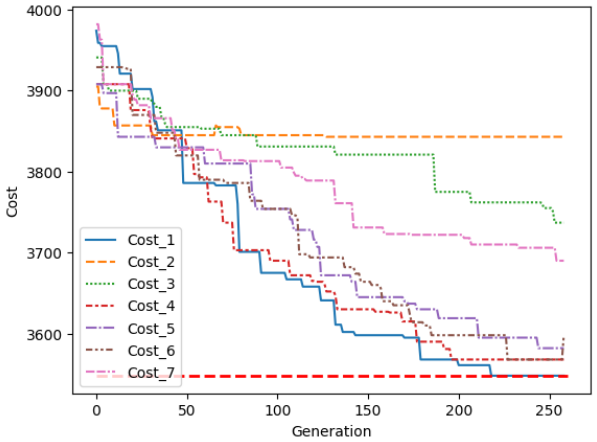


Fig. 5. Run time cost curves for the same input

C. Experiment Analysis

Based on the experiments conducted and the methodology outlined earlier, we can derive the following key conclusions.

Firstly, our experiments have revealed that the shuffle process plays a vital role in assisting the AI algorithm in escaping local minima, as exemplified by the findings in Figure 3. Although the use of shuffle may lead to a deterioration in the average cost performance in most scenarios, it does not significantly impede the overall process. Given the relatively low time complexity associated with the shuffle process, its inclusion in the algorithm remains justified.

Furthermore, upon examining Figure 3(d), we observe that the val7A dataset, under a 20-second time limit, only allows the algorithm to run for approximately 35 generations. This limited number of generations may appear less than ideal. However, this observation can be rationalized when considering the high time complexity associated with the 2-opt operations within the data structure of the val7A dataset. These operations are computationally intensive, resulting in a low number of iterations and, consequently, a less optimal final program execution outcome.

Based on the outcomes of multiple runs using different

random seeds on the same dataset, it becomes evident that while the algorithm can produce favorable results in many instances, it exhibits a degree of instability. While most runs yield satisfactory outcomes, there are isolated cases where the results fall short of expectations. These instances suggest that the algorithm may still be susceptible to local optima, even though it generally performs well.

V. Conclusion

In conclusion, our algorithm, which incorporates heuristic methods and genetic algorithms, has demonstrated promising results in addressing the Capacitated Arc Routing Problem. An innovative crossover operator pursuing reduction of similarity has laid a solid foundation for the algorithm. In the process, the algorithm includes the shuffle process, which can avoid the trap of the algorithm falling into the local minimum to a certain extent.

In the algorithm analysis part, the time complexity of the algorithm is analyzed. This gives the algorithm a theoretical basis in practical implementation.

Our experiments, based on five distinct data sets, have yielded promising results, demonstrating the algorithm's effectiveness to a significant extent. The shuffle process has also emerged as a beneficial addition to the algorithm. However, it's worth noting that our experiments have unveiled some degree of instability, with results prone to substantial fluctuations under random circumstances.

While our algorithm shows considerable promise and innovation, further research and fine-tuning may be necessary to address the algorithm's stability concerns. Nevertheless, our work contributes to the ongoing effort to tackle routing problems effectively, offering valuable insights and a foundation for future enhancements in this domain.

References

[1] G. B. Dantzig and J. H. Ramser, "The truck dispatching problem," *Management science*, vol. 6, no. 1, pp. 80–91, 1959.

[2] M. Dror, *Arc routing: theory, solutions and applications*. Springer Science & Business Media, 2012.

[3] B. L. Golden and R. T. Wong, "Capacitated arc routing problems," *Networks*, vol. 11, no. 3, pp. 305–315, 1981.

[4] S. Wøhlk, "A decade of capacitated arc routing," *The vehicle routing problem: latest advances and new challenges*, pp. 29–48, 2008.

[5] J. Liu, K. Tang, and X. Yao, "Robust optimization in uncertain capacitated arc routing problems: Progresses and perspectives," *IEEE Computational Intelligence Magazine*, vol. 16, no. 1, pp. 63–82, 2021.

[6] M. Fadzli, N. Najwa, and M. Luis, "Capacitated arc routing problem and its extensions in waste collection," in *AIP Conference Proceedings*, vol. 1660, no. 1. AIP Publishing, 2015.

[7] H. Handa, L. Chapman, and X. Yao, "Robust route optimization for gritting/salting trucks: a cercia experience," *IEEE Computational Intelligence Magazine*, vol. 1, no. 1, pp. 6–9, 2006.

[8] M. Liu, H. K. Singh, and T. Ray, "A memetic algorithm with a new split scheme for solving dynamic capacitated arc routing problems," in *2014 IEEE Congress on Evolutionary Computation (CEC)*, 2014, pp. 595–602.

[9] A. Hertz and M. Mittaz, "A variable neighborhood descent algorithm for the undirected capacitated arc routing problem," *Transportation science*, vol. 35, no. 4, pp. 425–434, 2001.

[10] P. Beullens, L. Muyldermans, D. Cattrysse, and D. Van Oudheusden, "A guided local search heuristic for the capacitated arc routing problem," *European Journal of Operational Research*, vol. 147, no. 3, pp. 629–643, 2003.

[11] P. Lacomme, C. Prins, and W. Ramdane-Cherif, "Competitive memetic algorithms for arc routing problems," *Annals of Operations Research*, vol. 131, pp. 159–185, 2004.

[12] M. Monroy-Licht, C. A. Amaya, A. Langevin, and L.-M. Rousseau, "The rescheduling arc routing problem," *International Transactions in Operational Research*, vol. 24, no. 6, pp. 1325–1346, 2017.

[13] H. Tong, L. Minku, S. Menzel, B. Sendhoff, and X. Yao, "A novel generalized metaheuristic framework for dynamic capacitated arc routing problems," in *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, 2023, pp. 45–46.

[14] A. Kansou, B. Kanso, and A. Yassine, "Hybridised ant colony optimisation for the multi-depot multi-compartment capacitated arc routing problem," *International Journal of Operational Research*, vol. 48, no. 1, pp. 18–46, 2023.

[15] A. Hertz, G. Laporte, and M. Mittaz, "A tabu search heuristic for the capacitated arc routing problem," *Operations research*, vol. 48, no. 1, pp. 129–135, 2000.

[16] S. Wang, Y. Mei, and M. Zhang, "Explaining genetic programming-evolved routing policies for uncertain capacitated arc routing problems," *IEEE Transactions on Evolutionary Computation*, 2023.

[17] G. Ulusoy, "The fleet size and mix problem for capacitated arc routing," *European Journal of Operational Research*, vol. 22, no. 3, pp. 329–337, 1985.

[18] B. L. Golden, J. S. DeArmon, and E. K. Baker, "Computational experiments with algorithms for a class of routing problems," *Computers & Operations Research*, vol. 10, no. 1, pp. 47–59, 1983.

[19] W. L. Pearn, "Approximate solutions for the capacitated arc routing problem," *Computers & Operations Research*, vol. 16, no. 6, pp. 589–600, 1989.

[20] ——, "Augment-insert algorithms for the capacitated arc routing problem," *Computers & Operations Research*, vol. 18, no. 2, pp. 189–198, 1991.

[21] Y. Mei, K. Tang, and X. Yao, "Improved memetic algorithm for capacitated arc routing problem," in *2009 IEEE Congress on Evolutionary Computation*. IEEE, 2009, pp. 1699–1706.

[22] J. S. DeArmon, "A comparison of heuristics for the capacitated chinese postman problem," Ph.D. dissertation, University of Maryland, 1981.

[23] E. Benavent, V. Campos, A. Corberán, and E. Mota, "The capacitated arc routing problem: lower bounds," *Networks*, vol. 22, no. 7, pp. 669–690, 1992.

[24] R. W. Eglese, "Routeing winter gritting vehicles," *Discrete applied mathematics*, vol. 48, no. 3, pp. 231–244, 1994.

[25] L. Y. Li and R. W. Eglese, "An interactive algorithm for vehicle routeing for winter—gritting," *Journal of the Operational Research Society*, vol. 47, no. 2, pp. 217–228, 1996.