Knowledge-Based Systems 220 (2021) 106957

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Memetic algorithm with non-smooth penalty for capacitated arc routing problem[☆]

Rui Li [a], Xinchao Zhao [a,b,*], Xingquan Zuo [c], Jianmei Yuan [b,d], Xin Yao [e]

[a] *School of Science, Beijing University of Posts and Telecommunications, Beijing 100876, China*
[b] *Hunan Key Laboratory for Computation and Simulation in Science and Engineering, Xiangtan University, Xiangtan 411105, China*
[c] *School of Computer Science, Beijing University of Posts and Telecommunications, Beijing 100876, China*
[d] *School of Mathematics and Computational Science, Xiangtan University, Xiangtan 411105, China*
[e] *Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen 518055, China*

ARTICLE INFO

Article history:
Received 9 July 2020
Received in revised form 10 February 2021
Accepted 13 March 2021
Available online 16 March 2021

Keywords:
Capacitated arc routing problem (CARP)
Memetic algorithm (MA)
Smooth degree
Non-smooth penalty
Reinsertion method
Smooth condition

ABSTRACT

The capacitated arc routing problem (CARP) has attracted much attention during last decades due to its wide applications. However, the existing research methods still have a little potential to make full use of the characteristics of CARP. This paper aims to mine the essential characteristics of arc routing problem that node routing problem does not have. Based on the observation on characteristics of arc routing instances, smooth condition is proposed and constructed as a rule to divide the link between two tasks into smooth link and non-smooth link. Then smooth degree is defined to measure the influence of non-smooth links on solution and a small smooth degree means the better quality for a solution. The effect of smooth degree is verified through simulation comparison on several instance sets, which indicates that there is a positive correlation between smooth degree and the total cost of a solution. Non-smooth penalty is used to drive the non-smooth solution to smooth and to improve its total cost. Then non-smooth penalty is inserted into path-scanning variants and new construction algorithms are obtained. A partial reconstruction method (PRM) is designed using these construction algorithms as an alternative kernel method. In order to further reduce the routes number, a reinsert method (RiM) is proposed. Combining these two methods with traditional local search algorithms, a memetic algorithm with non-smooth penalty (MANSP) is proposed which is originated from the initial observation on the essential characteristics of arc routing problem. Extensive experiments on smooth degree, penalty factor, and comparison with state-of-the-art algorithms show that the proposed strategies are effective and the proposed algorithm MANSP performs very competitive.

1. Introduction

Capacitated Arc Routing Problem (CARP) [1] is a well-known arc routing problem, which is defined on an undirected networks $G = (V, E)$. Set V contains n nodes and set E contains m edges. Each edge models a two-way street whose both sides are served in parallel and in any direction, like common narrow streets in rural areas. A fleet of identical vehicles with limited capacity Q is waited at a depot node. A subset E_R of edges require service by a vehicle. All edges can be traversed any times but is served only once. Each edge (i, j) has a known traversal cost $c_{ij} \geq 0$ and a demand $r_{ij} \geq 0$. CARP aims to determine a set of vehicle routes with minimum total cost, such that each route starts and ends at the depot, each required edge is served by one single route, and the total demand served by a route does not exceed the vehicle capacity Q. A survey on CARP and other arc routing problems can be found in [2].

CARP has been widely applied in road networks including gritting [3], urban waste collection [4–6], snow removal [7], field path planning [8], etc. Due to the wide applications of CARP, many variations have been studied. Periodic CARP [9,10] considers various applications in periodic operations on street networks, like waste collection and sweeping. Multi-trip CARP [11,12] is proposed for urban waste-collection problem that depots and disposal facilities are located in different places. Multi-objective CARP [13,14] not only minimizes the total-cost but also balances these trips. Since CARP is NP-hard [15], how to deal with large-scale CARP [9,16–19] remains an open question.

☆ This work is supported by National Natural Science Foundation of China (61973042, 61873040) and Beijing Natural Science Foundation, China (1202020).
The code (and data) in this article has been certified as Reproducible by Code Ocean:https://help.codeocean.com/en/articles/1120151-code-ocean-s-verification-process-for-computational-reproducibility. More information on the Reproducibility Badge Initiative is available at www.elsevier.com/locate/knosys.

* Corresponding author at: School of Science, Beijing University of Posts and Telecommunications, Beijing 100876, China.
E-mail address: zhaoxc@bupt.edu.cn (X. Zhao).

https://doi.org/10.1016/j.knosys.2021.106957
0950-7051/© 2021 Elsevier B.V. All rights reserved.

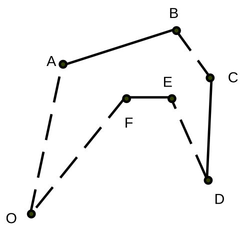

Fig. 1. An example of a non-smooth link.

There have been many successful algorithms that have been reported in the literatures. The NP-hard [15] character of CARP leads exact methods like [20] only applicable to small-scale instances. Then heuristic methods [15,19,21] are proposed to obtain an approximate solution in a relatively low computing time, including Augment-Merge (AM) [15], Path-Scanning (PS) [21], Ulusoy's algorithm [22]. Both AM and PS aim to construct a good feasible solution, so they are usually used to generate the initial solution for other algorithms. A certain giant tour can be splitted into a feasible solution by Ulusoy's tour splitting algorithm [22], which is very useful for route-first cluster-second method. The meta-heuristic algorithm based on local search is expected to improve the existing solutions, like tabu search for CARP (CARPET) [23], variable neighborhood descent algorithm (VND) [24], guided local search algorithm (GLS) [25]. Many excellent results are derived from meta-heuristics, for example, Memetic Algorithms [14,26–29] and ant colony optimization algorithm [30,31] etc.

Memetic Algorithm was firstly applied to CARP in [26,28]. MA has great potential and often leads to better performance. So, many improved MA algorithms are proposed. Memetic algorithm with extended neighborhood search (MAENS) [27] is proposed in which Merge-Split (MS) operator is capable of searching using large step sizes. Quantum-Inspired Immune Clonal Algorithm (QICA) [32] combines the feature of artificial immune system and quantum superposition ground on the qubit to solve the large scale CARP. Rank-based Memetic algorithm [29] designs the Rank-based Neighborhood Search (RENS) operator and constructs two rules to explain its working mechanism. Decomposition-based Memetic algorithm [13] applies the decomposition technique to solve the multi-objective CARP. An improved decomposition-based Memetic algorithm [14] enhances the performance of algorithm by replacing the solutions immediately and an archive for elite individuals. Memetic algorithm is used for periodic capacitated arc routing problem in [9] and large-scale capacitated arc routing problems in [17]. Excellent adaptability makes MA possible to be used for other problems and also indicates competitive performance, like optimal control problems of Zika virus epidemic with equilibriums [33], flow-shop scheduling problem [34] and nonlinear equation system [35].

Different from the node routing problem, the distance between two tasks maybe not the minimum distance in arc routing problem. This can be seen in Fig. 1, where the minimum connection of *EF* and *CD* is *CE*, but *DE* is used, where the solid line is the required edge and the dashed line is the shortest connection. It makes sense to discuss this difference between arc and node routing problem, so smooth conditions are defined in this paper to determine whether a link is smooth or not. Smooth degree is then used to measure the overall characteristics of an arc solution. As shown in Fig. 2 the convergence of smooth degree indicates that there is a positive correlation between the smooth degree and the total cost of a solution.

Since the total cost of a solution can be approximated by smooth degree, the non-smooth penalty is used to punish those

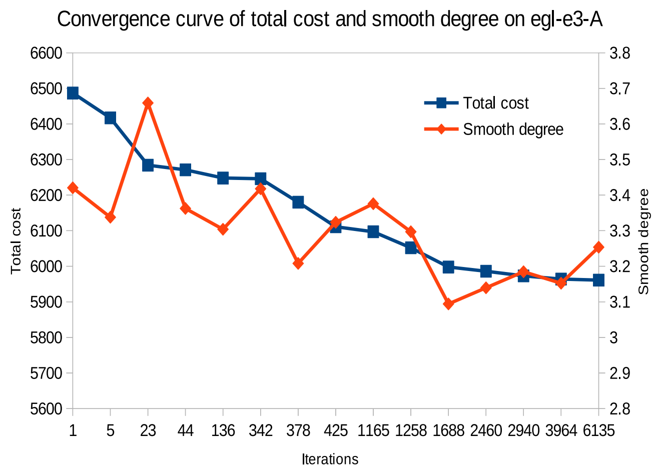

Fig. 2. Convergence curve of total cost and smooth degree.

non-smooth links and non-smooth solutions. It is natural to insert non-smooth penalty into path-scanning operation to make those non-smooth solution as smooth as possible. It means that if one task u is added to the route $(0, \ldots, e)$, it should be the nearest task to the end task e of the route and the distance between e and u must be the minimum distance among all four possible distances between them. Then two new variants of path-scanning methods are proposed and compared with four alternative construction algorithms: Augment-Merge (AM) [15], Path-Scanning (PS) [21], Path-Scanning with random selection (PSE) [36] and Path-Scanning with ellipse rule (PSE) [37]. Experiments show that better solutions can be produced by new construction methods.

Because local search sometimes cannot provide enough improvement, partial reconstruction method (PRM) is proposed using those constructive algorithms as kernel method based on the framework of Merge-Split method. A simple reinsertion method (RiM) is proposed to further reduce the total smooth degree with a trial of reducing the route number. Then an improved memetic algorithm with non-smooth penalty (MANSP) based on the observed essential characteristics of arc routing problem is proposed in this paper.

Fig. 3 illustrates the logic diagram of research motivation of this paper. The motivation of this paper is to find out a special characteristic of arc routing problem which is different from the node routing problem. The observed smooth link is such a characteristic that node routing problem does not have. The concept of smooth degree is defined to measure the importance of the smooth link and to describe the overall characteristics of an arc solution. Based on the experimental observation it is found that there is a positive correlation between the total cost and the smooth degree of a solution. New MA variant with the essential characteristic of arc routing problem is proposed, which embeds non-smooth degree and non-smooth penalty of a solution with non-smooth link into the existing algorithms as a penalty.

The main contributions of this paper are: (1) the essential characteristics of arc routing problem, smooth link and smooth degree are firstly observed and analyzed. (2) non-smooth penalty of solution is embedded into the existing algorithms and two improved path-scanning methods are obtained. (3) taking four existing and two new path-scanning methods as an alternative kernel algorithms, a partial reconstruction method framework is proposed. (4) combining the observed characteristic of arc routing problem and the following defined non-smooth penalty, a novel memetic algorithm (MANSP) based on the reinsertion method is proposed for CARP.

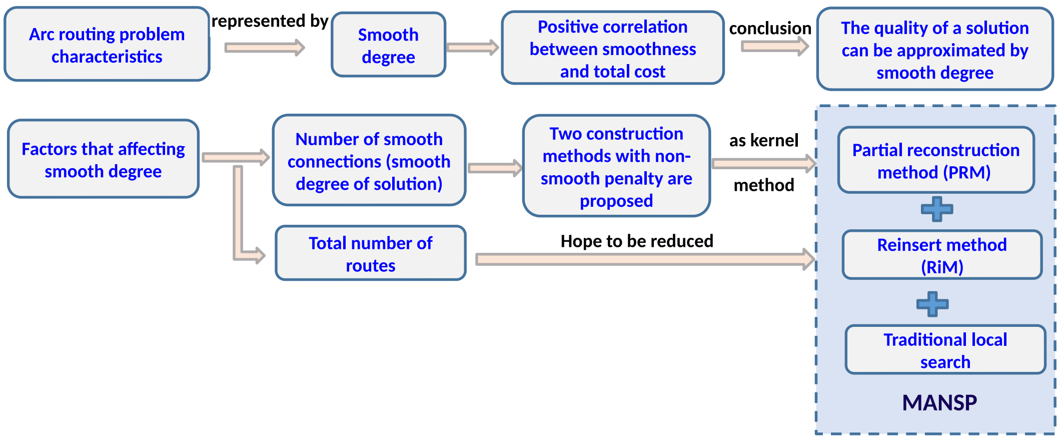
Arc routing problem characteristics
represented by
Smooth degree
Positive correlation between smoothness and total cost
conclusion
The quality of a solution can be approximated by smooth degree
Factors that affecting smooth degree
Number of smooth connections (smooth degree of solution)
Two construction methods with non-smooth penalty are proposed
as kernel method
Partial reconstruction method (PRM)
Reinsert method (RiM)
Traditional local search
MANSP
Total number of routes
Hope to be reduced

Fig. 3. The logic diagram of research motivation of this paper.

The rest of this paper is organized as follows. The CARP model and the basic operator of MA are described in Section 2. In Section 3, the definition of smooth degree and the non-smooth are penalty presented, then two non-smooth penalty path-scanning methods are proposed. Partial reconstruction method, reinsertion method, and a Memetic algorithm with non-smooth penalty (MANSP) are proposed in Section 4. Section 5 is devoted to detailed analyze the smooth degree of construction algorithms, penalty factor parameters, kernel algorithm, performance comparison with several state-of-the-art algorithms on multiple instance sets and the effect of reinsertion method. Section 6 concludes this paper.

2. Background knowledge

In this section, the mathematical model and solution representation of CARP are firstly introduced. Then the general framework of MA and traditional local search operations for CARP are briefly described.

2.1. Representation of solution and CARP model

A general CARP model is described in this section. CARP is searching for a minimum cost route for vehicles to serve all the required edges $E_R \subseteq E$ and required arcs $A_R \subseteq A$ for a given graph G = (V, E, A), and does not exceed vehicles' capacity constraints Q. Each arc task $\langle i,j\rangle$ is assigned a unique tag, marked as t, $t \in Z_+$ which is the positive integer set. Each edge task (i,j) is considered as a pair of arcs $\langle i,j\rangle$ and $\langle j,i\rangle$, one tag for each direction. Thus, each edge task is assigned with two tags. Each tag t is associated with five features, namely, $tail(t)$, $head(t)$, $sc(t)$, $dc(t)$, and $dem(t)$, which stand for the tail and head vertices, serving cost, deadheading cost, and the demand of the corresponding task respectively. If t belongs to an edge task, $inv(t)$ denotes the inversion of task t. The serving cost, deadheading cost, and demand of task $inv(t)$ are the same as $sc(t)$, $dc(t)$, and $dem(t)$ respectively. But note that each edge task should be served only once, in either directions (i.e., either task t or $inv(t)$ is served). Specially, depot is looked as an arc task $\langle depot, depot\rangle$, and its tag is set to 0. A solution of CARP is represented as an ordered list of tasks, denoted as $S = (t_1, t_2, \dots, t_N)$, where t_i is the ith task of S, N is the number of tasks (both edge task and arc task). The 0 task can be used to split the giant road to routes, and a solution of CARP can be marked as $S = (S_1, S_2, \dots, S_{N(S)})$, where S_i is the ith route of solution S, $N(S)$ is the number of routes.

Fig. 4 illustrates such a solution representation, in which S is a complete solution with several routes, and S_i is the ith route of the solution. Given a solution S, the corresponding routing plan can be obtained by connecting every two subsequent tasks with the shortest path between them (i.e., finding a shortest path from the head vertex of the former task to the tail vertex of the subsequent task), which can be easily found by *Dijkstra*'s algorithm [38]. The distance between task t_j and task t_{j+1} is presented as $sp(t_j, t_{j+1})$.

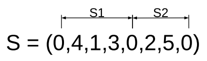

Fig. 4. The representation of a solution.

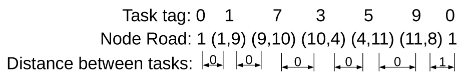
Task tag: 0 1 7 3 5 9 0
Node Road: 1 (1,9) (9,10) (10,4) (4,11) (11,8) 1
Distance between tasks: 0 0 0 0 0 1

Fig. 5. A part of solution.

Fig. 5 represents a route with different encoding schemes, which further shows the corresponding transformations among them. The first line is the task-based encoding, and the second line is the corresponding vertex-based encoding. For example, task 1 in the first row matches the arc task (1, 9) of the second row. It is similar for the rest tasks of the first row and arc tasks of the second row. Specially, task 0 represents the virtual task (1, 1) in which node 1 is the depot of the second row. Route starts from the depot and ends at the depot. Then the start and the end component values of both rows are marked as 0 in the task tag and 1 in the node path. The third line represents the distance between the adjacent vertices in the second row. If the adjacent nodes are the same, the distance is set as 0.

According to the solution representation and the above mentioned constraints, CARP model is given as follows [27].

Definition 1 (*Model of Capacitated arc Routing Problem*)**.**

$$\min_{S} \quad z = \sum_{j=1}^{length(S)-1} [sc(t_j) + sp(t_j, t_{j+1})] \tag{1a}$$

$$s.t. \sum_{j=1}^{|S_i|} dem(t_j) \le Q, \ \ \forall i \in \{1, 2, \dots, \ N(S)\} \tag{1b}$$

$$app(t_j) = 1, \ \ \forall t_j \in A_R \tag{1c}$$

$$app(t_j) + app(rev(t_j)) = 1, \ \ \forall t_j \in E_R$$

$$t_j \in \{1, 2, \dots, (2|E_R| + |A_R|)\} \tag{1d}$$

where $length(S)$ is the total encoding length of S and $length(S) = |E_R| + |A_R| + N(S) + 1$, $sc(t_j)$ is the serving cost of task

t_j, $N(S)$ is the number of routes in solution S, $|S_i|$ is the number of tasks in the ith route of solution S, $app(t_j)$ counts the times that task t_j appears in the solution S.

2.2. Framework of Memetic algorithms (MAs)

Memetic algorithm was invented by Moscato [39], which is inspired by both Darwinian principles of natural evolution and Dawkins' notion of memes. From the evolutionary computation perspective, MA can be roughly viewed as a general framework of population-based EA hybridizing with individual learning procedure that is capable of performing local refinements [26]. The general framework of MAs is summarized as Alg. 1.

Algorithm 1 General MAs

```
1: Initialization: Generate an initial population
2: while stopping criteria are not met do
3:    Evaluate all individuals in the population
4:    Evolve population using evolutionary operators
5:    for each individual do
6:       if rand(0,1) < $P_{ls}$ then
7:          improve the individual by local search
8:       end if
9:    end for
10: end while
```

From Alg. 1, the main difference between MAs and EAs is that the mutation operator of EAs is replaced by local search in MAs. Hence, the success of MAs [26,28] is mainly attributed to the performance of local search operation. Many important enhancing works in the incremental development of MAs [40,41] are centered around the local search procedure.

The procedure of MAs for CARP is shown in Alg. 2, which improves individuals one by one.

Algorithm 2 MA for CARP

```
1: Initialization: Generate an initial population
2: while stopping criteria are not met do
3:    Evaluate all individuals in the population
4:    Evolve a new solution $S_x$ using evolutionary operators
5:    if rand(0,1)< $P_{ls}$ then
6:       improve the individual by local search
7:       $S_l = LS(S_x)$
8:       if $S_l$ is better than $S_x$ then
9:          $S_x = S_l$
10:      end if
11:   end if
12:   sort population with fitness increasing
13:   randomly select one $S_y, y \in [popsize/2, popsize]$
14:   if $S_x$ is better than $S_y$ then
15:      $S_y = S_x$
16:   end if
17: end while
```

2.3. Initial population

Many construction algorithms [15,19,21,37] are used to solve CARP, which can be used to initialize the population of MA. Several classic construction algorithms used in this paper are introduced in this section.

In the initialization process for CARP, one individual is usually generated by one construction algorithm. These individuals generated by the adopted construction algorithms, for example, AM, PS, PSR and PSE, aim to obtain several good initial solutions and to accelerate the convergence speed. Other individuals are generated randomly, then Ulusoy's split algorithm [22] is used to split these individuals to feasible solutions.

2.3.1. Augment Merge (AM)

Augment-merge (AM) [15] algorithm is composed of two phases, namely, Augment phase and Merge phase. Initially, AM builds one route from each task and then sorts these $T = |E_R| + |A_R|$ routes in decreasing cost order. In the Augment phase each route S_i $(i = 1, 2, \ldots, T - 1)$ is compared with each subsequent route S_j with smaller cost $(j = i+1, i+2, \ldots, T)$. If S_i traverses the unique task u of S_j and it does not exceed the capacity constraints, S_i will collect task u and S_j discards it. In the Merge phase, each pair of routes is sorted in descending order according to the savings if both routes can be merged. This process will not stop until no further positive saving remains.

2.3.2. Path Scanning (PS)

Firstly, the heuristic path-scanning (PS) [21] is introduced. PS is a greedy-adding heuristic algorithm that first initializes the route using the task tag of depot. At each iteration, PS tires to find out the nearest tasks which do not violate the capacity constraints. If no task satisfies the constraints, PS will connect the end of the current route to depot with the shortest path between them to form a route. Then it initializes a new empty path and adds depot to it. If only one task satisfies the constraints, PS adds that task to the end of the current route. If there are multiple feasible tasks which are called as the tied edges, five rules are used to determine which one should be chosen.

Rule 1: maximize the distance from the head of task to the depot;
Rule 2: minimize the distance from the head of task to the depot;
Rule 3: maximize the term $dem(t)/sc(t)$, where $dem(t)$ and $sc(t)$ are the demand and serving cost of task t;
Rule 4: minimize the term $dem(t)/sc(t)$;
Rule 5: use Rule 1 if the current capacity of vehicle is less than half-full. Otherwise use Rule 2.

If multiple tasks still remain, ties are broken arbitrarily. PS terminates when all the tasks have been selected. It scans the unordered list of tasks for five times and only one rule is used in each scan. Hence, PS will generate five ordered lists of tasks in total. The best one is selected as the result of PS.

2.3.3. Path Scanning with random select (PSR)

Path Scanning with random selection (PSR) [36] is a random-adding heuristic which is different from PS. PSR selects new task randomly in the feasible nearest list. To eliminate the negative effects of randomness, PSR is possible to be rerun multiple times to get a better solution. Performance [36] indicates that a random selection of tied edges performs as well as the multi-criterion selection procedures in [21], and suggests that greedy-adding heuristics should be compared to random-adding heuristics during the testing and evaluation for CARP. The main difference between PSR and the original PS is the selection rule, i.e., PSR selects next edge randomly and PS selects next edge based on the above mentioned five rules. Therefore, PSR requires multiple runs to get the best solution.

2.3.4. Path Scanning with ellipse rule (PSE)

Path Scanning with ellipse rule (PSE) [37] is based on PSR. The only difference is the time when to select the final task. PSE uses the "ellipse rule" to determine which one is chosen. When the remain load space $rsv \leq \alpha * td/ned$, the "ellipse rule" will be used, where rsv is the remain space of vehicle, td is the total demand of instance, ned is the task number of the instance. α can be artificially set, and PSE algorithm advises $\alpha = 1.5$. The "ellipse rule" selects the nearest task whose serving cost is no less than the expected value of cost. The expected value is defined as $tc/ned + sp(end\_task, depot)$, where tc is the total cost of the instance, $sp(end\_task, depot)$ is the deadheading distance from the latest task of current route to depot.

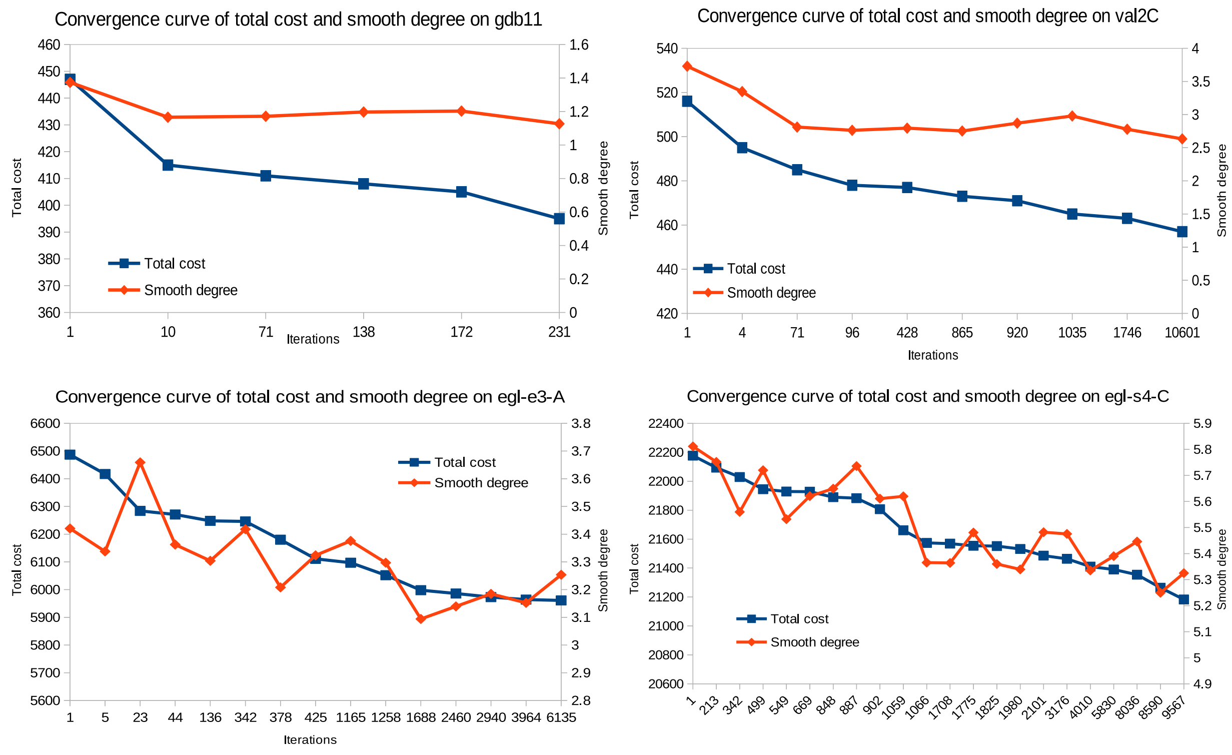

Fig. 6. the total cost and smooth degree convergence of MA.

Table 1
Comparison between the optimal and the suboptimal results on *gdb9* and *gdb23*.

Solution	Routes number	Total cost	Smooth interval link number	Total interval link number	Cost of smooth interval link	Total cost of non-smooth interval link	Smooth degree (SD)
$gdb9-S1$	10	303	60	61	81	3	1.3972
$gdb9-S2$	10	308	59	61	85	4	1.3991
$gdb9-S3$	10	309	59	61	83	7	1.4429
$gdb23-S1$	10	233	65	65	10	0	1.0448
$gdb23-S2$	11	235	66	66	12	0	1.0538

Definition 2 (*Ellipse Rule*).

$$sp(v_i, v_j) + sc(v_j) + sp(v_j, depot) \leq tc/ned + sp(v_i, depot) \qquad (2)$$

where $sp(v_i, v_p)$ is the deadheading distance from the task v_i to task v_p. PSE also requires multiple runs to get the best solution. There are also several variants of the path-scanning algorithms, see [42].

2.4. Crossover

At each iteration of Memetic algorithm [26,28], order crossover (OX) operator is implemented. Two parents are chosen by binary tournament selection, marked as P1 and P2 with length N. The classical OX crossover draws two random subscripts p and q with $1 \leq p < q \leq N$. To build a child C, the substring $P1[p] - P1[q]$ is copied into $C[p] - C[q]$. Finally, it scans P2 in a circular way from $q + 1(mod\ N)$ to $q(mod\ N)$ and copies each element not yet taken to fill C. OX is required to extend for the CARP and the solution encoding method. Each chromosome contains all t tasks, but an edge can appear as one internal arc u or its inverse $inv(u)$. Therefore, when copying u from a parent, we must check whether u and $inv(u)$ are not yet taken. Then the children individuals are then evaluated after Ulusoy's splitting algorithm [22].

2.5. Local search operators

Unlike OX operator, which is usually general and applicable to various list-based problems, local search operators are usually expected to incorporate some domain specific heuristics. Therefore, MAs can balance well between application generality and problem specificity with a hybridization of population-based EA and individual learning. In this section, some traditional local search operators for CARP will be briefly introduced. Because a solution to CARP is encoded as a sequence of tasks, any local search algorithm that works for sequences can be used. There are four commonly used move operators for CARP [26,28], which are single insertion, double insertion, swap, and 2-opt.

(1) Single Insertion: In the single insertion move, a task t 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 $t \in E$, both of its directions will be considered when re-inserting.

(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 also considered for edge tasks.

(3) Swap: In the swap move, two candidate tasks are selected and their positions are exchanged. Both directions are also 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

Algorithm 3 Path_scanning with random selection and non-smooth penalty (PSRP)

```
Input: candidate list $\{t_1, t_2, \ldots, t_{2*N(S)}\}$, $t_i$ is the $i$th task, $N(S)$ is the required task number, max_ite is the maximum run times.
Output: a feasible solution $S_{best}$
 1: ite := 0, $S_{best} := \emptyset, S := \emptyset$
 2: while ite<max_ite do
 3:    $S := \emptyset$, add the task 0 to the S, set e := 0, reset candidate list F=∅,
 4:    set remain capacity of the current vehicle $rvc := 0$
 5:    while candidate list is not empty do
 6:       if $rvc <= d_a$ // for the end task then
 7:          get the candidate tasks set $U$ with smallest penalty distance tasks from $e$ to $u$ and $u$ to depot
 8:       else
 9:          get the candidate tasks set $U$ with smallest penalty distance tasks from $e$ to $u$
10:       end if
11:       if $U$ is empty then
12:          Add task 0 to $S$, set $e := 0$, $rvc := Q$, go to line 5.
13:       end if
14:       select one task $u$ randomly from $U$.
15:       add $u$ to $S$
16:       Set $e = u$, $rvc := rvc - demand(u)$
17:       delete $u$ and $u'$ from the candidate list, $u'$ is the reverse task of $u$.
18:    end while
19:    if $S$ better than $S_{best}$ then
20:       $S_{best} := S$
21:    end if
22: end while
```

move for a single route, a sub-route (i.e., a part of the route) is selected and its direction is reversed. When applying the 2-opt move to two routes, each route is cut into two routes and new solutions are generated by reconnecting these four routes.

The existing algorithms are general and cannot take advantage of the characteristics of the special problems. Then the characteristic feature exploited from problem is discussed and analyzed as follows.

3. Smooth degree and non-smooth penalty

In this section, the analysis on smooth degree, its influence on the performance of MA on some instances, and the improving methods are presented.

3.1. Smooth degree

Definition 3 (*Smooth Condition of Intermediate Link Between e_a, e_b*).

$$d(e_a, e_b) = min\{d(e_a, e_b), d(e_a, e'_b), d(e'_a, e_b), d(e'_a, e'_b)\} \tag{3}$$

where $e_a, e_b \in S$ are the task edges in solution S, $d(e_a, e_b)$ is the minimum distance between the head of e_a and the tail of e_b, e'_a, e'_b are the reverse of e_a, e_b respectively. If the interval link between e_a and e_b satisfies the smooth condition, it is called a smooth interval link between e_a and e_b, or this link is smooth.

Definition 4 (*Smooth Degree of Solution S*).

$$SD(S) = \frac{tc(S) + tic(S) - tsc(S)}{tc(S) - tic(S)} \tag{4}$$

where S is a solution, $tc(S)$ is the total cost of S, $tic(S)$ is the total cost of interval link of S, and $tsc(S)$ is the total cost of smooth interval link of S. $tc(S)-tic(S)$ is the service cost and $tic(S)-tsc(S)$ is the non-smooth interval link cost. $tc(S) + tic(S) - tsc(S)$ is the penalty to the total cost. Eq. (4) actually stands for non-smooth degree of a solution. So a large smooth degree value of a solution S means that it is less smooth. Otherwise, it is smooth. This phenomenon can be intuitively found in Fig. 6 and Table 1.

Fig. 6 illustrates the convergence curves of MA on four instances. Each subfigure in Fig. 6 has two convergence curves of total cost and smooth degree on instances *gdb11*, *val2C*, *egl-e3-A* and *egl-e4-C*. Observed from these four convergence curves, the changing trends of total cost and smooth degree are roughly consistent with each other, but not exactly the same. It means that, not strictly speaking, smooth degree can be used as a symbol for the quality of solution to some extent. The greater fluctuation of smooth degree for instances *egl-e3-A* and *egl-e4-C* means that the correlation of smooth degree and total cost, relatively speaking, is a little weaker for these two instances.

More detailed analysis on three solutions of *gdb*9 and two solutions of gdb23[43] are listed in Table 1. For the solutions of *gdb*9, the lower cost of non-smooth total interval link can lead to the lower cost of total interval link which means that the total cost is related to the total cost of non-smooth interval link. So the total cost can be decreased by reducing the total cost of non-smooth interval link. The experimental results of $gdb9 - S2$ and $gdb9 - S3$ show that the increase of smooth interval cost may be an offset by reducing the cost of non-smooth interval link. It means that the total cost of smooth interval may be allowed to rise moderately.

When solving *gdb*23 instance [26], Memetic algorithm (MA) is easy to fall into the local optimal solution with total cost 235, but the optimal solution has a total cost 233. Comparing the optimal and the suboptimal solutions, it can be found that the suboptimal solution needs more routes (11 vs. 10) than the optimal solution. The suboptimal solution needs more vehicles to serve the tasks which causes more wasting between the depot and the first or the last task. The experimental results of $gdb23 - S1$ and $gdb23 - S2$ show that the total cost is possible to be further decreased by reducing the total number of routes when there is no enough room to improve.

3.2. Non-smooth penalty path-scanning method

3.2.1. Non-smooth penalty

Although smoothness cannot completely represent the quality of the solution, it can reflect the quality of the solution to a certain extent. Therefore, the fitness value of a solution can be constructed by adding non-smoothness to the original cost as an additional penalty as Eq. (5).

Definition 5 (*Penalty Fitness of Solution S*).

$$fitness(S) = tc(S) + \lambda \cdot ns(S) \tag{5}$$

where $tc(S)$ is the total cost of S, $\lambda > 0$, $ns(S)$ is the non-smooth travel cost of S. Instead of taking the total cost only as the fitness value, non-smooth penalty is used as a partial of the fitness value when evaluating a solution. Therefore, the modified fitness value $fitness(S)$ of a solution is calculated with the sum of the total cost and its non-smooth penalty.

3.2.2. Non-smooth penalty path-scanning method

The non-smooth penalty method can improve the smoothness by penalizing the non-smooth cost and adding it to the original cost. This means that the next added task is the closest and has the minimal non-smooth travel cost. Non-smooth penalty can be easily hybridized into the existing constructive method as Eq. (6).

Definition 6 (*Penalty Distance From Task u to Task v*)**.**

$$d_p(u, v) = d(u, v) + \lambda(d(u, v) - d_{min}(u, v)) \quad (6)$$

where $d(u, v)$ represents the distance from head of task u to tail of task v, $d_p(u, v)$ means the penalty distance from head of task u to tail of task v, and $\lambda > 0$, $d_{min}(u, v)$ is the minimum distance among all the possible links between task u and task v. Then the next task is selected from those tasks with the minimum penalty distance.

The performance of PSE is much better than PSR because "ellipse rule" is considered in PSE. The success of "ellipse rule" in Eq. (2) shows that the situation of returning to the depot has to be considered. Therefore, when the remaining space of the vehicle is less than the average demand d_a of all tasks, the penalty distance from the current task e to u and u to the depot need to be considered. The average demand $d_a = td/ned$, td is the total demand of all tasks, ned is the number of tasks with a positive demand. In this paper, non-smooth penalty is embedded into path-scanning with random selection (PSR) and path-scanning with ellipse rule (PSE) methods, then two new methods are proposed which are path-scanning with random selection and non-smooth penalty (PSRP Alg. 3) and path-scanning with ellipse rule and non-smooth penalty (PSEP Alg. 4).

Algorithm 4 Path_scanning with ellipse rule and non-smooth penalty (PSEP)

```
Input: candidate list $\{t_1, t_2, \ldots, t_{2*N(S)}\}$, $t_i$ is the $i$th task, $N(S)$ is the required
    task number, max_ite is the maximum run times .
Output: a feasible solution $S_{best}$
1: ite := 0, $S_{best} := \emptyset, S := \emptyset$
2: while ite<max_ite do
3:   $S := \emptyset$, add the task 0 to the S, set e := 0, reset candidate list
4:   while candidate list is not empty do
5:     start from e to find the next nearest tasks set $U$ which satisfies smooth
       condition and capacity constraint
6:     if $U$ is empty then
7:       Add task 0 to $S$, set e := 0, $rvc := Q$, go to line 4.
8:     end if
9:     if $rvc \geq \alpha \times td/ned$ then
10:      get the candidate tasks set $U$ with smallest penalty distance.
11:    else
12:      get the candidate tasks set $U$ with smallest penalty distance from $e$
         to $u$ and $u$ to the depot and satisfied ellipse condition Eq. (2).
13:    end if
14:    select one task $u$ randomly from $U$.
15:    add $u$ to $S$
16:    Set $e = u$, $rvc := rvc - demand(u)$
17:    delete $u$ and $u'$ from the candidate list, $u'$ is the reverse task of $u$.
18:  end while
19:  if $S$ better than $S_{best}$ then
20:    $S_{best} := S$
21:  end if
22: end while
```

The comparison between two new path-scanning methods with PSR and PSE is shown in Table 2. The average deviation to the lower bound cost is listed on three instance sets. The performance comparison considers two different cases with maximum running times, 1000 and 10000. The comparison results show that PSRP and PSEP get better solutions than PSR and PSE in most cases except for PSE on *val* instance set. PSRP is significantly superior to PSR in three instance sets. It means that the non-smooth penalty method significantly improves the performance of path-scanning with random selection. On the *gdb* and *egl* instance sets, PSEP performs better than PSE, but slightly worse than PSEP in *val* instance set. It indicates that the non-smooth penalty cannot improve the performance of path-scanning with ellipse rule in any case. Comparing PSRP and PSEP, PSRP performs better than PSEP in *gdb* instance set, but worse in *val* and *egl* instance sets. It is possible to be caused by the large scale of *val* and *egl* instance sets. There is no doubt that the results of the maximum running 10000 times are better than those of 1000 times, but the improvement is not so significant. To save the computing cost for comparable results, the maximum running time is set as 1000. The non-smooth penalty does not introduce any extra computational complexity to PSRP and PSEP, which is still $O(n^2)$. Considering multiple runs for PSRP and PSEP, the final computational complexity is $O(max\_ite * n^2)$, where max_ite is the maximum running times of PSRP and PSEP. In general, PSRP and PSEP are competitive construction algorithms and non-smooth penalty is effective for path-scanning.

Table 2
Performance comparison among the existing and new path-scanning methods.

	Av. deviation to LB(%)		
	gdb	val	egl
PSG	10.62	16.34	26.05
Total run = 1000			
PSR(1000)	3.34	7.65	17.68
PSRP(1000)	1.32	5.35	10.12
PSE(1000)	1.53	4.84	10.05
PSEP(1000)	1.45	4.95	9.78
Total run = 10 000			
PSR(10 000)	2.35	5.31	16.28
PSRP(10 000)	0.80	3.89	9.04
PSE(10 000)	1.07	3.44	8.92
PSEP(10 000)	0.93	3.49	8.89

4. Memetic algorithm with non-smooth penalty (MANSP)

4.1. Partial reconstruction method for individual

The main idea of partial reconstruction method (PRM) is to randomly select two routes from one solution, reconstruct them with kernel method and insert them into the original solution. This is possible that the construction algorithm can approach to the optimal solution only on small scale problems. The experiments in Section 5.1 have shown that none algorithm can generally deal with different problems properly. The same method is firstly used in Merge-Split [27] algorithm, which selects 2 or 3 routes as sub-problem and reconstructs two routes by Path-Scanning method. Considering that the construction algorithms are only effective on small scale problems, two routes are chosen from a solution to construct a new solution in this paper. Referring to the practice of Merge-Split approach, a partial reconstruction method for solutions is proposed as Alg. 5. It has several alternative kernel algorithms that are different from Merge-Split.

Four heuristic algorithms mentioned in Section 2.3 and two new methods mentioned in Section 3.2.2 are the candidate kernel algorithms in the PRM framework. The experiment result shows that none algorithm can get the lowest cost for all the instance sets. It means that it is impossible for one algorithm to work well for all the problems. So it is necessary to select the matching kernel algorithm for different specific problem. In order to choose the suitable kernel algorithm, a hypothesis is presented as follows.

HYPOTHESIS: if a kernel algorithm works well on the whole problem, it works well on a partial problem with a high probability.

Therefore, we only need to check whether the kernel algorithm can solve the whole problem well. The most matching algorithm for a certain problem is selected as the kernel algorithm for this problem. So, this assumption will be verified empirically in Section 5.3. The computational complexity of PRM depends on the complexity of the kernel algorithm, and the computational complexity is $O(n^2)$.

Algorithm 5 Partial Reconstruction Method (PRM)

```
Input: a solution $S$, $ite = 0$
Output: an improved solution $S$
1: while ite<100 do
2:     select two different routes $S_i, S_j \in S$ randomly,
3:     remove $S_i, S_j$ from S and let $S_p^{old} = \{S_i, S_j\}$
4:     construct a sub-CARP by $S_p^{old}$
5:     generate $S_p^{new}$ by solving sub-CARP with kernel algorithm
6:     calculate the total cost of $S_p^{new}$
7:     if $S_p^{new}$ is better than $S_p^{old}$ then
8:         $S = \{S - S_p^{old}\} \cup S_p^{new}$
9:         break;
10:    end if
11:    ite = ite+1
12: end while
```

4.2. Initialization and the kernel method selection

During the population initialization, six initial solutions are firstly generated with six construction methods. Then the smooth degrees of six solutions are computed and compared. The corresponding construction method with the lowest smooth degree is selected as the kernel method for the following PRM method in the whole algorithm when solving one problem. Other solutions are randomly initialized.

4.3. Reinsertion method

As shown in Table 1 the total cost is possible to be further decreased by reducing the total number of routes when there is no enough room to improve it. So the reinsertion method is proposed to decrease the total number of routes. If all the tasks of any route can be inserted into other routes, the total number of routes is reduced accordingly. This method is described as Alg. 6. The average computational complexity is $O(n^2)$.

Algorithm 6 Reinsert Method (RiM)

```
Input: a feasible solution $S = \{S_1, S_2, \dots, S_{N(S)}\}$, $S_i$ is the $i$th route, $k$ is randomly
    selected or specified in $\{1, 2, \dots, N(S)\}$
Output: a reinserted solution $S^I$
1: for each task $t$ in $S_k$ do
2:     find the best feasible insert position in all the other routes $\{S_i, i \in \{1, 2, \dots, N(S)\} \wedge i \neq k\}$ for $t$ or $rev(t)$
3:     move $t$ from $S_k$ to the feasible insert position
4: end for
```

However, it is possible that the number of routes in some solutions cannot always be reduced. That is to say, it is not all the routes of one solution that can be splitted and inserted into other routes. For example, a solution has demands {{1, 2}, {1, 2}, {2, 3}, {1, 3}}, and capacity is 5. The first route is inserted into other routes which maybe {{1, 2, 2}, {2, 3}, {1, 1, 3}}. But neither the third nor the fourth route can be inserted into other routes without violating the constraint due to the demand 3 of a task.

Table 3
Comparison on average deviation to LB and Average smooth degree among algorithms.

	Av. deviation to LB(%)			Av. smooth degree		
	gdb	val	egl	gdb	val	egl
AM	12.72	17.60	10.10	1.40	1.72	4.73
PS	11.47	15.11	26.13	1.46	1.82	6.79
PSR	3.37	7.60	17.73	1.29	1.57	6.15
PSE	1.60	5.02	9.82	1.24	1.50	5.30
PSRP	1.24	5.57	10.25	1.23	1.52	5.40
PSEP	1.42	4.97	9.86	1.23	1.51	5.30

Table 4
Comparison on different penalty factor λ.

	Av. deviation to the LB(%)						
Value of λ	0	0.2	0.4	0.6	0.8	1	1.2
gdb	0.66	0.57	1.09	0.67	1.07	0.43	0.78
val	2.14	2.07	2.14	2.14	2.08	0.73	1.64
egl	5.70	5.62	5.56	5.63	5.69	3.34	5.36

Table 5
Parameters of MANSP.

Popsize	Population size	30
I	Max number of main loop	20 000
p_l	Local search rate in main phase	0.2
max_ite	The maximum rerun times of kernel methods	100
λ	The penalty factor in Eqs. (5) and (6)	1

This phenomenon raises a question that if there is always a route that can be inserted into other routes for any solution. At the same time, it keeps the capacity constraints satisfied when the total number of routes does not reach the minimum. More details are given as below.

Assume that:

(1) A solution of CARP is denoted as $S = \{S_1, S_2, \dots, S_{N(S)}\}$, $S_i = \{t_{i1}, t_{i2}, \dots\}$ is the ith route of S, t_{ij} is the jth task of S_i.
(2) $\exists S', s.t.\ N(S') = N(S) - 1, N(S) > 1$, which means that there is a solution with fewer routes.

The question is that if there always exists a route in S which can be inserted into other routes. A mathematical expression is that, $\forall S$, if $\exists S_m \in S, s.t.\ S^I = RiM(S_m)$, and $N(S^I) = (N(S) - 1) \wedge (\sum_{j=1}^{|S_i^I|} dem(t_j)) \le Q, \forall\ S_i^I \in S^I$. where $RiM(S_m)$ represents that all the tasks in S_m are inserted into other routes by Alg. 6 aiming at reducing the route number, and S_i^I is the ith route of S^I.

It is a pity that the answer is not true. Here is a counter example. Supposing tasks have demands {1, 2, 3, 4, 5, 6, 7, 8}, and capacity is 9. The demands of solution S are supposed to be {{1,7}, {2,6}, {3,5}, {4}, {8}} and the demands of solution S' are {{1,8}, {2,7}, {3,6}, {4,5}}. So none route of S can be inserted into other routes of either S or S'. It means that splitting and reinserting one route into other routes does not always lead to a fewer routes number. However, it is still a useful heuristic to reduce the route number. In the reinsertion method of this paper, one route is randomly selected and tried to insert into the others. The experiment in Section 5.5 shows that the reinsertion method can improve the probability of obtaining the global optimal solution of 233 for gdb23 instance. So this method is a useful and beneficial strategy and has a certain universality.

4.4. Memetic algorithm with non-smooth penalty (MANSP)

In MA [28] for CARP, a chromosome S is simply a sequence of N tasks, without trip delimiters and with implicit shortest paths between consecutive tasks. S can be viewed as a giant

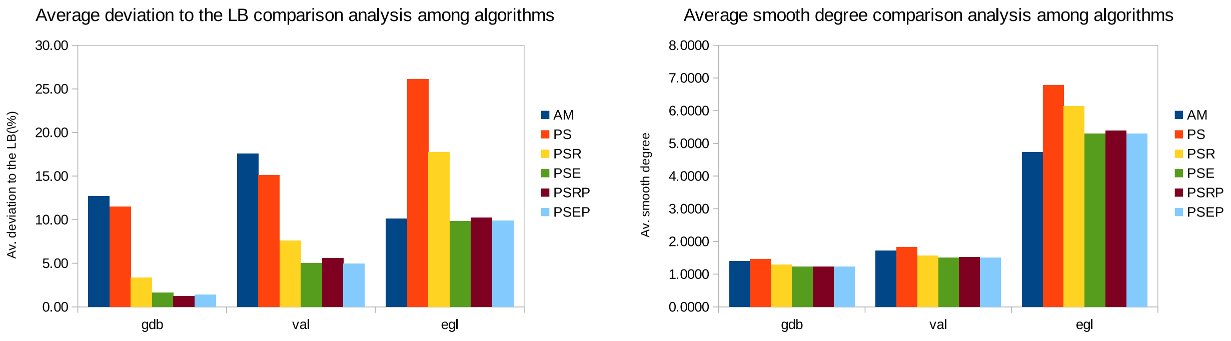

Fig. 7. Comparison on construction algorithms on three instance sets.

trip ignoring capacity Q. Then the Split procedure of Ulusoy's heuristic [22] is applied to S and a solution is obtained. Ulusoy's Split method is very useful to split the routes for CARP. The fitness $F(S)$ of S is the sum of the total cost and the non-smooth penalty of this solution. Compared to traditional local search, an MA works on a population of solutions whose crossover is based on two solutions with a large neighborhood. This leads to the intrinsic parallelism of MA.

An improved MA with the observed smooth or non-smooth characteristic of arc routing problem is proposed in this paper to reduce the possible number of routes and to enhance the general performance. There are two questions to be considered here,

(1) If the number of routes is large, how to reduce it;
(2) If the number of routes is suitable, how to improve it.

The first question can be partially solved by Alg. 6 of reinsertion method. The second question can be partially solved by Alg. 5 of partial reconstruction method. For MA, the first question can also be partially solved by improving the rate of individuals with small route numbers. Before initializing the population, it is necessary to decide which method will be chosen from AM, PS, PSR, PSE, PSRP and PSEP as the kernel method. So a simple test is used to determine the kernel method, and the best algorithm in the test is used as the kernel method for this run. Then the general framework of Memetic algorithm with non-smooth penalty is presented as Alg. [39]. Due to the computational complexity of crossover is $O(n)$, the complexity of PRM and RiM are $O(n^2)$, and the complexity of local search is $O(n^2)$, the computing complexity of MANSP is $O(I*(n+n^2+n^2+n^2))$ or $O(I*n^2)$, where I is the total iteration number of main loop as Table 5.

Algorithm 7 MA with non-smooth penalty (MANSP)

```
Input: $G = (V, E)$
Output: a solution $S$
1: Initialize population $POP = \{S^i\}, i \in \{1, 2, \dots, popsize\}$
2: select the best kernel method from six candidate algorithms
3: while the termination condition is not met do
4:    select $S^i, S^j \in POP$ randomly
5:    generate $S^x$ by OX($S^i, S^j$)
6:    if rand(0,1)<$p_l$ then
7:       improve $S^l$ by PRM($S^l$) (Alg. 5)
8:       use traditional local search LS to improve $S^l$
9:       if $\lceil \frac{total\_cost}{Q} \rceil$ <|S| then
10:         generate $S^l$ by RiM($S^x$) (Alg.6)
11:         use traditional local search LS to improve $S^l$
12:      else
13:         $S^l \leftarrow S^x$
14:      end if
15:      if fitness($S^l$) < fitness($S^x$) then
16:         $S^x \leftarrow S^l$
17:      end if
18:   end if
19:   sort $POP$ in increasing order
20:   randomly choose $k \in [popsize/2, popsize]$
21:   if $S^x \preceq S^k$ or $N(S^x) \leq N(S^k)$ then
22:      $S^k \leftarrow S^x$
23:   end if
24: end while
```

4.5. Convergence analysis of MANSP

The convergence properties of the canonical genetic algorithm (CGA) with mutation, crossover and proportional reproduction has analyzed by Rudolph [44]. It has proved that a CGA will never converge to the global optimum regardless of the initialization, crossover, operator and objective function by means of homogeneous finite Markov chain analysis. However, the variant of CGA is shown to converge to the global optimum due to the irreducibility property of the underlying original nonconvergent CGA if only it always maintains the best solution in the population, either before or after selection. Alg. [39] shows that the proposed MANSP keeps the best solution found until now in the current population. Therefore, it can be seen that MANSP has convergence property.

5. Experimental studies and comparison analysis

To evaluate the efficacy of the proposed algorithm MANSP, extensive experiments are carried out in this section. Firstly, smooth degrees of six construction algorithms (four traditional (PS, AM, PSR, PSE) and two new (PSRP, PSEP)) are analyzed. Secondly, the crucial parameter of non-smooth penalty factor and the possible kernel algorithms are discussed. Thirdly, the competitive performance of MANSP algorithm is compared with several state-of-the-art algorithms. The effect of reinsert method is also verified in Section 5.5. All the experiments were conducted on three benchmark sets of CARP instances: *gdb* [43], *val* [45] and *egl* instance sets [46]. When comparing with the most well-known algorithm MAENS [27], *bmcv* [25] instance set including four subsets and each of which has 25 instances is also adopted. All the experiments were executed on a 3.7 GHz i5-9600K PC under Ubuntu 18.04.4 by C++.

5.1. Smooth degree of construction algorithms

Four traditional (AM, PS, PSR, PSE) and two new (PSRP, PSEP) construction algorithms are studied and the comparison results are shown in Table 3 and Fig. 7. It aims to find an algorithm with lowest smooth degree and the comparison results are shown in Table 3. AM and PS does not need to be run multiple times. The maximum iteration is set as 500 for PSR, PSE, PSRP and PSEP.

The left part of Fig. 7 is the average deviation of these algorithms against LB in each instance set, and the right part indicates the average smoothness comparison. Fig. 7 shows the comparison results more intuitively. Observed from Table 3 and Fig. 7, PSE,

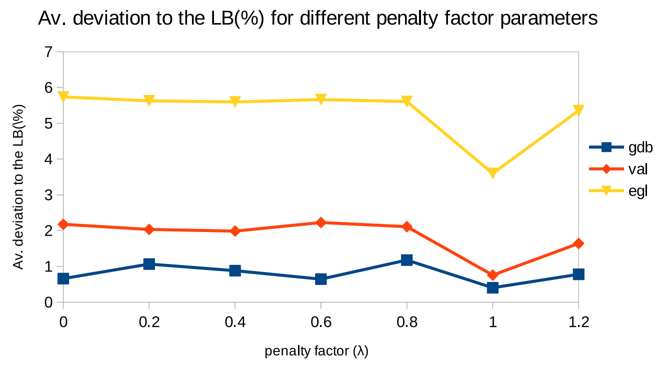

Fig. 8. The trend line of average deviation to LB(%) for different penalty factor.

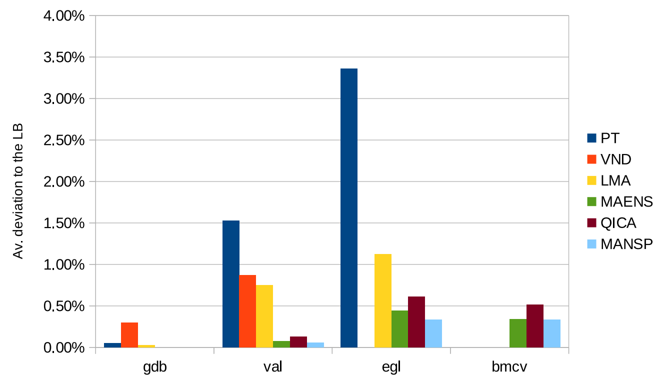

Fig. 9. Comparison on average deviation to the lower bound among six algorithms.

PSEP and PSRP obtains lowest average deviation to the lower bound and smooth degree values on all three instance sets. Especially in the *gdb* instances set, PSRP achieves much lower average deviation to LB than others. So PSRP is more suitable to be used to enhance the individual after reinsert operation for *gdb* instances set. Similarly, algorithm PSEP is suitable for instances set *val*, and algorithm PSE is suitable for *egl*. It needs to be mentioned that AM gets the lowest smooth degree in the *egl* instances set, and obtains rather good solutions. This shows that algorithm AM can produce solutions with good smoothness on *egl* instances set.

In general, the newly proposed construction algorithms have encouraging effects. At the same time, both groups of numerical comparison results also guide which algorithm will be used as the kernel method for the following PRM. At the same time, the results show that different construction algorithms have different performance on different problems. Therefore, the choice of kernel method depends on the specific problems.

5.2. Discussion on penalty factor parameters

This section analyzes the influence of the crucial penalty factor parameter in Eq. (5) and (6) of the proposed MANSP algorithm. There are seven choices for parameter λ from 0 to 1.2 with a step of 0.2. The numerical comparison and the trend lines of average deviation to the LB(%) for different penalty factor parameters on three instance sets are shown in Table 4 and Fig. 8. Other parameters of MANSP are shown in Table 5 which were suggested in [28].

Observed from Fig. 8 it can be seen that smaller parameters have little effects on the algorithmic performance and the best results are obtained when $\lambda = 1$. Therefore, the default setting of λ is 1 without special explanation in the later discussion.

Table 6
Comparison among MANSP_AM, MANSP_PS and MANSP_PSEP on three benchmark sets, 'min', 'mean' and 'std' are the minimum, mean and standard deviation in 30 runs.

Instance	MANSP_AM		MANSP_PS		MANSP_PSEP	
	Min	Mean	Min	Mean	Min	Mean
gdb13	536	536.0	536	539.1	538	542.7
gdb23	233	233.0	233	233.0	235	235.0
val5D	583	594.4	583	590.4	582	591.2
val8C	523	534.0	527	529.3	526	534.5
val9C	332	332.0	332	332.0	333	335.4
val9D	393	395.5	391	392.1	391	396.1
val10D	530	535.5	530	534.7	533	537.5
egl-e1-B	4524	4536.1	4501	4527.3	4498	4507.6
egl-e2-B	6323	6361.0	6317	6344.4	6317	6335.0
egl-e3-B	7789	7850.1	7787	7805.0	7777	7795.9
egl-e3-C	10 292	10402.2	10 303	10333.2	10 305	10341.5
egl-e4-A	6478	6503.1	6461	6482.6	6464	6464.7
egl-e4-B	9009	9107.2	9005	9052.7	9000	9049.3
egl-e4-C	11 724	11817.3	11 629	11713.4	11 643	11731.3
egl-s2-A	10 053	10150.4	9918	9989.2	9907	9952.4
egl-s2-B	13 352	13456.8	13 236	13327.3	13 242	13314.4
egl-s2-C	16 607	16807.0	16 535	16626.1	16 437	16603.1
egl-s3-A	10 325	10417.2	10 290	10366.4	10 262	10325.8
egl-s3-B	13 904	14072.6	13 909	13966.4	13 786	13911.4
egl-s3-C	17 413	17591.1	17 309	17426.7	17 260	17345.6
egl-s4-A	12 467	12606.1	12 432	12481.6	12 409	12469.5
egl-s4-B	16 496	16624.5	16 487	16544.5	16 416	16482.0
egl-s4-C	20 806	20966.7	20 679	20874.9	20 706	20826.9

5.3. Discussion on the kernel algorithm

In Section 4.1, the hypothesis indicates that if a kernel algorithm works well on the whole problem, it is able to work well on a sub-problem. Due to the sub-problem space is too large to check, the final result of the whole problem is used for checking the kernel algorithm instead of the sub-problem. Therefore, the hypothesis is transformed into: if algorithm *A* performs better than algorithm *B* on the whole problem, it will get better result than *B* in the final solution on the sub-problem. Because the results of four algorithms (PSE, PER, PSRP, PSEP) are similar, PSE is chosen as the representative for simplicity. The comparison results on three instance sets are shown in Table 6. Similarly, the instances comparison are not provided here with the same results for four algorithms.

Observed from Table 6, MANSP_AM and MANSP_PS perform a bit better than MANSP_PSE on the *gdb* instances set, but the best construction method is PSEP in Table 3. The comprehensive comparison results in Tables 3 and 6 show a little inconsistency with the assumption.

MANSP_AM, MANSP_PS and MANSP_PSEP obtain 2, 3 and 3 best results, respectively, in Table 6. The best construction method is PSE in Table 3. It indicates that the most matching method gets the best solution with high probability. However, the rest two methods cannot perfectly suit the proposed and empirically verified hypothesis and three algorithms have no significant superiority among each other.

On the *egl* set, MANSP_PSEP performs much better than MANSP_AM and MANSP_PS. It is a strong support to the hypothesis together with the results of Table 3. At the same time, MANSP_PS outperforms MANSP_AM on the *egl* instances set which further confirms the assumption. All in all, the simulation results are basically in line with the proposed hypothesis indicated in Section 4.1. So we can draw a conclusion that a kernel

Table 7
Comparison results on the *gdb* benchmark set, 'min', 'mean', 'std' are the minimum, mean and standard deviation values of 30 results, 'NS' is the success time of finding the lower bound LB [47–49].

Instance	(|N|, |E|)	LB	PT	VND	LMA	MAENS	QICA	MANSP			
								Min	Mean	Std	NS
gdb1	(12,22)	316	316	316	316	316∼	316∼	316	316.0	0.0	30
gdb2	(12,26)	339	339	339	339	339∼	339∼	339	339.0	0.0	30
gdb3	(12,22)	275	275	275	275	275∼	275∼	275	275.0	0.0	30
gdb4	(11,19)	287	287	287	287	287∼	287∼	287	287.0	0.0	30
gdb5	(13,26)	377	377	377	377	377∼	377∼	377	377.0	0.0	30
gdb6	(12,22)	298	298	298	298	298∼	298∼	298	298.0	0.0	30
gdb7	(12,22)	325	325	325	325	325∼	325∼	325	325.0	0.0	30
gdb8	(27,46)	348	348	350	350	348∼	348∼	348	350.5	1.0	6
gdb9	(27,51)	303	303	315	303	303∼	303∼	303	306.2	3.0	7
gdb10	(12,25)	275	275	275	275	275∼	275∼	275	275.0	0.0	30
gdb11	(22,45)	395	395	395	395	395∼	395∼	395	395.0	0.0	30
gdb12	(13,23)	458	458	458	458	458∼	458∼	458	458.0	1.0	27
gdb13	(10,28)	536	538	544	536	536∼	536∼	536	536.0	0.0	30
gdb14	(7,21)	100	100	100	100	100∼	100∼	100	100.0	0.0	30
gdb15	(7,21)	58	58	58	58	58∼	58∼	58	58.0	0	30
gdb16	(8,28)	127	127	127	127	127∼	127∼	127	127.0	0.0	30
gdb17	(8,28)	91	91	91	91	91∼	91∼	91	91.0	0	30
gdb18	(9,36)	164	164	164	164	164∼	164∼	164	164.0	0.0	30
gdb19	(11,11)	55	55	55	55	55∼	55∼	55	55.0	0	30
gdb20	(11,22)	121	121	121	121	121∼	121∼	121	121.0	0.0	30
gdb21	(11,33)	156	156	156	156	156∼	156∼	156	156.0	0.0	30
gdb22	(11,44)	200	200	200	200	200∼	200∼	200	200.0	0.0	30
gdb23	(11,55)	233	235	235	233	233∼	233∼	233	233.0	0.0	30
#			21	19	22	23	23	23			
%			0.05	0.03	0.02	0	0	0			

"#" represents the times of algorithms that find the lower bound.
"%" represents the average deviation to the LB(%).
"∼" indicates that there is no significant difference between MAENS/QICA and MANSP.
"↓" indicates that MAENS/QICA is significantly better than MANSP ($p < 0.05$).
"↑" indicates that MAENS/QICA is significantly worse than MANSP ($p < 0.05$).

algorithm works well on the whole problem and then it will work well on a sub-problem with a high probability. An even suitable kernel algorithm is possible to get an even bigger performance improvement.

5.4. *Performance comparison with state-of-the-art algorithms*

Extensive experiments were conducted to illustrate the performance of MANSP algorithm. Firstly, MANSP is comprehensively compared with several state-of-the-art algorithms. Then it proves that the proposed RiM method is necessary for the non-smooth problem and the final performance of algorithm.

These experiments are firstly conducted on two benchmark sets of undirected instances *gdb* and *val* in which only edge tasks need to be served. Another experimental comparison is done on a mix instance set *egl* with edge tasks and arc tasks. All these instances can be obtained at http://www.uv.es/~belengue/carp.html. The proposed MANSP is also comparatively verified with *bmcv* benchmark [25], which is based on the intercity road network in Flanders and can be found in https://logistik.bwl.uni-mainz.de/forschung/benchmarks. MANSP is compared with state-of-the-art algorithms of CARPET [23] (abbreviated as PT), VND [24], LMA [28] and the well-known MAENS [27], QICA [32]. The *bmcv* [25] benchmark instances set, which contains four subsets and each of which has 25 instances is adopted to compare with MAENS and QICA. *Wilcoxon* rank sum statistical test is used to show whether there are significant difference among algorithms.

The average gap to the lower bound [50] of these algorithms is summarily illustrated in Fig. 9. This figure shows that MANSP achieves the best lower average deviation to the lower bound in four instance sets. It shows the general even better performance of MANSP when comparing with the state-of-the-art competitors. There are no VND results for *egl* instance and no PT, VND and LMA results for *bmcv*. The performance of the algorithm on these instance sets is then discussed specifically in more detail.

5.4.1. gdb

The *gdb* benchmark set contains 23 instances from DeArmon [43] with 7 to 27 nodes and 11 to 55 edges. Experimental comparison results are presented in Table 7, in which (|N|,|E|) represent the total numbers of nodes and edges respectively. The "LB" item is the lower bound [47–49]. The proposed MANSP algorithm runs 30 times on each instance. The "min", "mean", "std" are the minimum total cost, the mean total cost, and the standard deviation of the 30 final results. "NS" stands for the number of success.

Comparing MANSP with VND and LMA, it finds a better result on instance gdb8, which is the known optimal result. Comparing MANSP to PT and VND on instances gdb13 and gdb23, it gets the optimal results and both competitors fall into the local trap. These simulation results show that the proposed MANSP algorithm can solve the non-smooth problem perfectly. MANSP gets all the best known results for all the instances. The computing results on instances gdb8 and gdb9 show that there is still improving space for MANSP. But it shows great improvement when comparing with *PT*, *VND* and *LMA*, especially for *LMA* on instances gdb11,

Table 8
Comparison results on the *val* benchmark set, 'min', 'mean', 'std' are the minimum, mean and standard deviation values of 30 results, 'NS' is the success time of finding the lower bound LB [47–49].

Instance	(|N|, |E|)	LB	PT	VND	LMA	MAENS	QICA		MANSP			
							Min	Std	Min	Mean	Std	NS
val1A	(24,39)	173	173	173	173	173∼	173∼	0.0	173	173.0	0.0	30
val1B	(24,39)	173	173	178	173	173∼	173∼	4.2	173	173.0	0.0	30
val1C	(24,39)	245	245	248	245	245∼	245∼	2.1	245	245.0	0.0	29
val2A	(24,34)	227	227	227	227	227∼	227∼	6.8	227	227.0	0.0	30
val2B	(24,34)	259	260	259	259	259∼	259∼	2.7	259	259.0	0.0	30
val2C	(24,34)	457	494	457	457	457∼	457∼	3.6	457	460.7	2.1	11
val3A	(24,35)	81	81	81	81	81∼	81∼	1.7	81	81.0	0.0	30
val3B	(24,35)	87	87	87	87	87∼	87∼	2.0	87	87.0	0.0	30
val3C	(24,35)	138	138	140	140	138∼	138∼	0.9	138	138.0	0.0	28
val4A	(41,69)	400	400	400	400	400∼	400∼	4.7	400	400.0	0.0	30
val4B	(41,69)	412	416	414	414	412∼	412∼	4.4	412	412.0	0.0	30
val4C	(41,69)	428	453	428	428	428∼	428∼	5.8	428	428.1	2.0	6
val4D	(41,69)	526	556	544	544	530∼	530∼	3.4	530	537.9	3.1	1
val5A	(34,65)	423	423	423	423	423∼	423∼	5.1	423	423.0	0.0	30
val5B	(34,65)	446	448	449	449	446∼	446∼	1.2	446	446.0	0.0	30
val5C	(34,65)	473	476	474	474	474∼	474∼	1.1	474	474.0	0.0	30
val5D	(34,65)	573	607	599	599	577↓	577↓	3.7	583	590.4	3.0	1
val6A	(31,50)	223	223	223	223	223∼	223∼	2.5	223	223.0	0.0	30
val6B	(31,50)	233	241	233	233	233∼	233∼	2.1	233	233.0	0.0	30
val6C	(31,50)	317	329	325	325	317∼	317∼	1.9	317	318.0	2.0	22
val7A	(40,66)	279	279	279	279	279∼	279∼	4.4	279	279.0	0.0	30
val7B	(40,66)	283	283	283	283	283∼	283∼	5.0	283	283.0	0.0	30
val7C	(40,66)	334	343	335	335	334∼	334∼	3.0	334	334.2	0.0	26
val8A	(30,63)	386	386	386	386	386∼	386∼	6.0	386	386.0	0.0	30
val8B	(30,63)	395	401	403	403	395∼	395∼	3.1	395	395.0	0.0	30
val8C	(30,63)	518	533	543	543	521∼	521∼	4.2	523	534.0	4.3	1
val9A	(50,92)	323	323	323	323	323∼	323∼	2.6	323	323.0	0.0	30
val9B	(50,92)	326	329	326	326	326∼	326∼	2.4	326	326.0	0.0	30
val9C	(50,92)	332	332	336	336	332∼	332∼	2.5	332	332.0	0.0	29
val9D	(50,92)	385	409	399	399	391∼	391∼	1.8	393	395.5	2.7	7
val10A	(50,97)	428	428	428	428	428∼	428∼	2.8	428	428.0	0.0	29
val10B	(50,97)	436	436	436	436	436∼	436∼	1.1	436	436.0	0.0	23
val10C	(50,97)	446	451	446	446	446∼	446∼	1.3	446	446.0	0.0	22
val10D	(50,97)	525	544	538	538	531↑	527↓	2.4	530	535.5	2.4	1
#			17	19	21	28	28	29*	28			
%			1.52	0.86	0.74	0.08	0.13		0.06			

"#" represents the times of algorithms that find the lower bound.
"%" represents the average deviation to the LB(%).
"∼" indicates that there is no significant difference between MAENS/QICA and MANSP.
"↓" indicates that MAENS/QICA is significantly better than MANSP ($p < 0.05$).
"↑" indicates that MAENS/QICA is significantly worse than MANSP ($p < 0.05$).
"*" indicates the number of instances that std value of QICA is larger than that of MANSP.

gdb13 and gdb23, in which *LMA* needs to restart many times. The *NS* items show the numbers that MANSP achieves the low bound LB directly at each run on each instance. The second line from the bottom of Table 7 represents the success times of getting the optimal solutions which indicates that MANSP performs as well as the well-known *MAENS* and better than other competitors. Furthermore, as the results of MANSP are comparable with those of MAENS, *Wilcoxon* rank sum test is used to show whether there are significant difference among them. "↑" indicates that the results of MANSP are significantly better than those of MAENS, "↓" represents the results of MANSP significantly worse than those of MAENS, "∼" means that there is no significant difference between them. Statistical test shows that there is no statistic difference between MAENS, QICA and MANSP in *gdb* instance set. However, the much larger "std" items of QICA means that the average results of QICA are much worse than those of MANSP. The last line show the average deviation to LB. MANSP can find the best solutions for all the instances in *gdb* instance set.

5.4.2. val *test set*

The *val* benchmark set [45] contains 34 instances with 24 to 50 nodes and 39 to 97 edges. Experimental results are given in Table 8. As the smooth degree analysis, this instance set also needs the proposed MANSP being hybridized with PSE method.

Table 8 shows that MANSP can find the best solutions on the majority *val* instances. Generally speaking, MANSP gets all the optimal solutions on 28 instances indicating as the last row of Table 8. MANSP achieves as good performance as the well-known MAENS and significantly outperforms other competitors. When comparing with MAENS detailedly, MAENS outperforms MANSP on instances val5D, val8C and val9D and is outperformed by MANSP on instance val10D. It should be pointed out that these four instances are the most difficult in the *val* instance set

Table 9
Comparison results on the *egl* benchmark set, 'min', 'mean', 'std' are the minimum, mean and standard deviation values of 30 results, 'NS' is the success time of finding the lower bound LB [47–49].

Instance	(|N|, |E|, |T|)	LB	PT	LMA	MAENS	QICA		MANSP			
						Min	Std	Min	Mean	Std	NS
e1-A	(77,98,51)	3548	3625	3548	3548∼	3548∼	19.1	3548	3548.0	0.0	30
e1-B	(77,98,51)	4498	4532	4498	4498∼	4498∼	15.9	4498	4507.6	11.2	16
e1-C	(77,98,51)	5566	5663	5595	5595∼	5595∼	25.5	5595	5616.4	14.1	4
e2-A	(77,98,72)	5018	5233	5018	5018∼	5018∼	1.6	5018	5019.8	3.0	26
e2-B	(77,98,72)	6305	6422	6340	6317∼	6314∼	20.6	6317	6335.0	17.8	11
e2-C	(77,98,72)	8243	8603	8415	8335∼	8335∼	28.8	8335	8346.6	7.0	5
e3-A	(77,98,87)	5898	5907	5898	5898∼	5898∼	34.8	5898	5898.4	2.2	26
e3-B	(77,98,87)	7704	7921	7822	7775∼	7775∼	20.8	7777	7795.9	8.2	1
e3-C	(77,98,87)	10163	10805	10433	10292∼	10292∼	21.5	10292	10341.5	43.1	8
e4-A	(77,98,98)	6408	6489	6461	6456∼	6456∼	25.6	6446	6464.7	0.0	29
e4-B	(77,98,98)	8884	9216	9021	8998↑	8975↓	25.8	8988	9049.3	21.3	1
e4-C	(77,98,98)	11427	11824	11779	11561∼	11567∼	37.6	11643	11731.3	50.0	3
s1-A	(140,190,75)	5018	5149	5018	5018∼	5018∼	40.6	5018	5018.0	0.0	30
s1-B	(140,190,75)	6384	6641	6435	6388∼	6388∼	21.1	6388	6412.9	18.8	10
s1-C	(140,190,75)	8493	8687	8518	8518∼	8518∼	22.4	8518	8518.7	0.0	29
s2-A	(140,190,147)	9824	10373	9995	9895∼	9909∼	43.3	9907	9952.4	25.2	1
s2-B	(140,190,147)	12968	13495	13174	13147∼	13110↓	68.0	13242	13314.4	38.3	1
s2-C	(140,190,147)	16353	17121	16795	16430∼	16425∼	39.2	16437	16603.1	78.2	1
s3-A	(140,190,159)	10143	10541	10296	10257∼	10221↓	44.4	10262	10325.8	27.1	1
s3-B	(140,190,159)	13616	14291	14053	13749∼	13692∼	63.3	13786	13911.4	68.7	1
s3-C	(140,190,159)	17100	17789	17297	17207∼	17214∼	36.4	17260	17345.6	42.6	1
s4-A	(140,190,190)	12143	13036	12442	12341↓	12265↓	56.8	12409	12469.5	26.3	1
s4-B	(140,190,190)	16093	16924	16531	16337↓	16262↓	57.9	16416	16482.0	35.2	1
s4-C	(140,190,190)	20375	21486	20832	20538↓	20505↓	87.5	20706	20826.9	52.4	1
#			0	5	5	5	17*	5			
%			3.36	1.12	0.44	0.33		0.61			

"#" represents the times of algorithms that find the lower bound.
"%" represents the average deviation to the LB(%).
"∼" indicates that there is no significant difference between MAENS/QICA and MANSP.
"↓" indicates that MAENS/QICA is significantly better than MANSP ($p < 0.05$).
"↑" indicates that MAENS/QICA is significantly worse than MANSP ($p < 0.05$).
"*" indicates the number of instances that std value of QICA is larger than that of MANSP.

and none existing algorithm can get their lower bounds. When comparing with other competitors, MANSP gets even better results on 16/12/12 instances than PT, VND and LMA and is not outperformed by any of them on any instance. Statistical test shows that there are no significant difference between MANSP and MAENS/QICA on *val* instance set. But the results show that the standard deviations of QICA are larger than those of MANSP on 27 instances, which means that MANSP performs not only as better as QICA but also more stable.

Table 10
Summary comparison of average deviation to LB on the *bmcv* instances set.

	MAENS	QICA	MANSP
bmcv-C	0.33	0.30	0.31
bmcv-D	0.37	0.99	0.36
bmcv-E	0.31	0.23	0.31
bmcv-F	0.16	0.20	0.10
mean	0.34	0.51	0.33

5.4.3. egl *test set*

Table 9 shows the performance comparison among algorithms on the *egl* instance set, which was generated by Eglese [46] based on data from a winter gritting application in Lancashire. It consists of 24 instances based on two graphs, each of which has a distinct set of required edges and capacity constraints. The first graph includes instances from e1-A to e4-C and the second graph includes instances from S1-A to S4-C. Each graph has 12 instances. VND [24] did not provide the simulation results on *egl* set and it is not provided here. Different from the *gdb* and *val* instance sets, not all the edges are the tasks in *egl* instance set. The (|N|, |E|, |T|) presents the node number, the edge number and the task number respectively.

First of all, MANSP outperforms PT on all the instances. Secondly, MANSP outperforms LMA on 15 of 24 instances and is outperformed by LMA on only two instances. Thirdly, MANSP has comparable performance with MAENS on all 12 instances of the first graph. However, it is outperformed by MAENS on the second graph. Even for MAENS, it also has the obvious difference with "LB", which indicates that the instances based on the second graph are very difficult and they are challenging the future optimization solvers. Statistical test shows that MANSP is slightly better than MAENS on the first graph and MANSP is slightly worse than MAENS on the second graph for the *egl* instance set.

QICA is a little better than MAENS on 6 instances. But the standard deviations of QICA are also larger than those of MANSP on 17 instances. It means that MANSP is once again more stable than QICA in *egl* instance set.

5.4.4. bmcv *test set*

The *bmcv* [25] benchmark set is used to further compare the similarities and differences between the proposed MANSP and the state-of-the-art algorithm MAENS [27] and QICA [32]. It is based

Table 11
Comparison between MANSP and MANSP_without_RIM on three benchmark sets, 'min', 'mean', 'std' are the minimum, mean and standard deviation values of 30 results.

Instance	MANSP			MANSP_without_RiM		
	Min	Mean	Std	Min	Mean	Std
gdb8	348	350.5	1	348	351.0	2
gdb9	303	306.2	3	303	308.1	3
val4D	530	537.9	3	536	540.4	3
val5D	583	590.4	3	587	595.7	4
val8C	523	534.0	4	528	536.1	3
egl-e1-B	4498	4507.6	11	4524	4539.4	10
egl-e1-C	5595	5616.4	14	5609	5648.3	28
egl-e2-B	6317	6335.0	17	6344	6368.1	20
egl-e2-C	8335	8346.6	7	8343	8445.7	64
egl-e3-B	7777	7795.9	8	7789	7870.4	54
egl-e4-A	6464	6464.7	0	6475	6500.6	20
egl-e4-B	9000	9049.3	21	9063	9134.4	55
egl-s1-C	8518	8518.7	0	8526	8612.3	42
egl-s2-A	9907	9952.4	25	10 072	10162.2	43
egl-s2-B	13 242	13314.4	38	13 310	13443.7	89
egl-s2-C	16 437	16603.1	78	16 598	16832.1	85
egl-s3-A	10 262	10325.8	27	10 334	10428.4	59
egl-s3-B	13 786	13911.4	68	13 818	14067.5	82
egl-s3-C	17 260	17345.6	42	17 391	17603.9	121
egl-s4-A	12 409	12469.5	26	12 483	12638.0	88
egl-s4-B	16 416	16482.0	35	16 433	16647.2	110
egl-s4-C	20 706	20826.9	52	20 784	20951.1	88
$\leq$	22	22		2	0	

Table 12
Comparison results on the C subset in *BMCV* benchmark set, 'min', 'mean', 'std' are the minimum, mean and standard deviation values of 30 results, 'NS' is the succeeding number of finding the lower bound LB [47–49].

Instance	LB	MAENS			QICA	MANSP			
		Repaired	Published	Offset		Min	Mean	Std	NS
C1	4150	4150∼	1660	2490	4150∼	4150	4209.5	27	4
C2	3135	3135∼	1095	2040	3135∼	3135	3158.2	23	0
C3	2575	2575∼	925	1650	2575∼	2575	2582.3	4	0
C4	3478	3510∼	1340	3510	3510∼	3510	3511.3	3	0
C5	5365	5365∼	5365	2895	5365∼	5365	5402	34	0
C6	2535	2535∼	895	1640	2535∼	2535	2549.5	14	5
C7	4075	4075∼	1795	2280	4075∼	4075	4075	0	0
C8	4090	4090∼	1730	2360	4090∼	4090	4091.7	8	0
C9	5233	5270↑	1830	3440	5260∼	5260	5305.2	21	0
C10	4700	4700∼	2270	2430	4700∼	4700	4726.5	16	3
C11	4583	4630∼	1805	2825	4640∼	4640	4705.8	31	0
C12	4209	4240∼	1610	2630	4240∼	4240	4244.3	11	0
C13	2955	2955∼	1110	1845	2955∼	2955	2955.7	1	0
C14	4030	4030∼	1680	2350	4030∼	4030	4041.8	15	0
C15	4912	4940↓	1860	1860	4940↓	4950	4986.5	12	0
C16	1475	1475∼	585	890	1475∼	1475	1478.7	6	0
C17	3555	3555∼	1610	1945	3555∼	3555	3563.8	10	0
C18	5577	5660↑	2425	3235	5620↓	5630	5657.3	15	0
C19	3096	3115∼	1395	1720	3120↑	3115	3119.8	0	0
C20	2120	2120∼	665	1455	2120∼	2120	2122.3	4	0
C21	3960	3970∼	1725	2245	3970∼	3970	3970	0	0
C22	2245	2245∼	1070	1175	2245∼	2245	2245	0	0
C23	4032	4095↑	1700	2395	4085∼	4085	4141	30	0
C24	3384	3400∼	1360	2040	3400∼	3400	3406.8	6	0
C25	2310	2310∼	905	1405	2310∼	2310	2320	14	0
%		0.33			0.30	0.31			

"%" represents the average deviation to the LB(%).

Table 13
Comparison results on the D subset in *BMCV* benchmark set, 'min', 'mean', 'std' are the minimum, mean and standard deviation values of 30 results, 'NS' is the succeeding number of finding the lower bound LB [47–49].

Instance	LB	MAENS			QICA	MANSP			
		Repaired	Published	Offset		Min	Mean	Std	NS
D1	3215	3230∼	740	2490	3215↓	3230	3234.8	0	0
D2	2520	2520∼	480	2040	2520∼	2520	2520	0	0
D3	2065	2065∼	415	1650	2065∼	2065	2065	0	0
D4	2785	2785∼	615	2170	2785∼	2785	2785	0	0
D5	3935	3935∼	1040	2895	3935∼	3935	3935	0	0
D6	2125	2125∼	485	1640	2125∼	2125	2125	0	0
D7	3078	3115↓	835	2280	3125∼	3125	3140	17	0
D8	2995	3045∼	685	2360	3045∼	3045	3064.2	14	3
D9	4120	4120∼	680	3440	4120∼	4120	4120	0	0
D10	3335	3340∼	910	2430	3330↓	3340	3340	0	0
D11	3745	3755	930	2825	3715↓	3760	3760	0	0
D12	3310	3310∼	680	2630	3310∼	3310	3310	0	0
D13	2535	2535∼	690	1845	2535∼	2535	2535	0	0
D14	3272	3280∼	930	2350	3272↓	3280	3280	0	0
D15	3990	4000∼	920	3080	3990↓	3995	4000	1	0
D16	1060	1060∼	170	890	1260↑	1060	1061	4	0
D17	2620	2620∼	675	1945	2620∼	2620	2620	0	0
D18	4165	4185↑	950	4165	4165∼	4165	4170.8	10	0
D19	2393	2400∼	680	1720	2393↓	2400	2400	0	0
D20	1870	1870∼	415	1455	1870∼	1870	1870	0	0
D21	2985	3055∼	810	2245	3045↓	3055	3074.3	19	0
D22	1865	1865∼	690	1175	1865∼	1865	1865	0	0
D23	3114	3130∼	735	2395	3130∼	3130	3147.2	6	0
D24	2676	2710∼	670	2040	2710∼	2710	2710	0	0
D25	1815	1815∼	410	1405	1815∼	1815	1815	0	0
%		0.37			0.99	0.36			

"%" represents the average deviation to the LB(%).

on the intercity road network in Flanders which can be found in https://logistik.bwl.uni-mainz.de/forschung/benchmarks. *Bmcv* instance set contains four subsets, namely, sets C, D, E, and F, each of which contains 25 instances. Instances of sets D and F share the same networks with instances of sets C and E, respectively, but with larger capacities of vehicles. But the results of MAENS [27] need to add the offset. The first column of MAENS provides the repaired results and the second column of MAENS provides the raw results. The better results of MANSP and MAENS are highlighted in bold.

The comparison results among MANSP, MAENS and QICA on *bmcv* benchmark set are presented in Tables 12–15 in the Appendix. The LB represents the lower bound found in [47–49]. Table 10 shows the average deviation values to LB. Table 10 indicates that MANSP gets the lowest mean average deviation values over all four subsets and 100 instances.

Table 12 shows that MANSP is better than MAENS on 3 instances and worse on 2 instances. MANSP is better than QICA on 1 instances and worse on 2 instances. Table 13 shows that MANSP is better than MAENS on 2 instances and worse on 2 instances. MANSP is better than QICA on 1 instances and worse on 7 instances. But for instance D16, QICA got a very poor solution. So QICA got the largest average deviation values. Table 14 shows that MANSP is better than MAENS on 3 instances and worse on 5 instances. MANSP is worse than QICA on 5 instances. Table 15 shows that MANSP is better than MAENS on 2 instances. MANSP is worse than QICA on 2 instances and MANSP gets the lowest average deviation values.

Overall, the proposed MANSP and the state-of-the-art MAENS get the same performance on most of the instances. MANSP is better than MAENS on 10 instances and worse on 9 instances. It shows that MANSP performs very competitive even when it compares with the most state-of-the-art algorithms.

5.5. Effect verification of reinsertion method

Memetic algorithm with non-smooth penalty (MANSP) performs powerful competitiveness. This section will empirically prove the necessity of the proposed reinsert operator which aims to reduce the route number. For all the instances of three instance sets, only those instances that MANSP and MANSP_without_RiM obtain different results are provided in Table 11, where MANSP_without_RiM is the MANSP algorithm without RiM operator and all others remain the same.

Generally speaking, MANSP outperforms MANSP_without_RiM on all these provided instances with different experimental results. Observed from Table 11, all the "std" items of MANSP_without_RiM are larger than those of MANSP, which means that MANSP with reinsert operator is more stable. This group of comparison results show that reinsertion method can not only improve the performance of algorithm but also increase its stability.

6. Conclusion

According to the observed characteristics of arc routing problem, smooth condition is proposed and constructed as a rule which divides the link between two tasks into smooth link and non-smooth link. To extend smooth link to the whole solution, smooth degree is defined to measure the quality of solution.

Table 14
Comparison results on the E subset in *BMCV* benchmark set, 'min', 'mean', 'std' are the minimum, mean and standard deviation values of 30 results, 'NS' is the succeeding number of finding the lower bound LB [47–49].

Instance	LB	MAENS			QICA	MANSP			
		Repaired	Published	Offset		Min	Mean	Std	NS
E1	4885	4910∼	1935	2975	4910↓	4915	4985.8	60	0
E2	3990	3990∼	1610	2380	3990∼	3990	4030.2	37	0
E3	2015	2015∼	750	1265	2015∼	2015	2015	0	0
E4	4155	4155∼	1610	2545	4155∼	4155	4244.2	17	0
E5	4585	4610↓	2185	2425	4585↓	4595	4682	49	0
E6	2055	2055∼	670	1385	2055∼	2055	2055	0	0
E7	4155	4155∼	1900	2255	4155∼	4155	4156	3	0
E8	4710	4710∼	2150	2560	4710∼	4710	4714.2	5	0
E9	5780	5870↓	2285	3585	5810↓	5855	5956.8	45	0
E10	3605	3605∼	1690	1915	3605∼	3605	3621	14	0
E11	4637	4670↓	1850	2820	4670↓	4680	4764.8	42	0
E12	4180	4200↓	1715	2485	4195↓	4215	4254.5	17	0
E13	3345	3345∼	1325	2020	3345∼	3345	3345.3	1	0
E14	4115	4115∼	1810	2305	4115∼	4115	4131.7	12	0
E15	4189	4225↓	1610	2615	4215∼	4215	4235.7	11	0
E16	3755	3775∼	1825	1950	3775∼	3775	3789.2	15	0
E17	2740	2740∼	1290	1450	2740∼	2740	2740	0	0
E18	3825	3835∼	1610	2225	3835∼	3835	3841.2	7	0
E19	3222	3235∼	1435	1800	3235∼	3235	3235	0	0
E20	2802	2825∼	990	1835	2825∼	2825	2825	0	0
E21	3728	3730∼	1705	2025	3730∼	3735	3779.8	22	0
E22	2470	2470∼	1185	1285	2470∼	2470	2470	0	0
E23	3686	3710∼	1430	2280	3710∼	3715	3742	15	0
E24	4001	4020∼	1785	2235	4020∼	4020	4063.5	21	0
E25	1615	1615∼	655	960	1615∼	1615	1615	0	0
%		0.31			0.23	0.31			

"%" represents the average deviation to the LB(%).

Further observation indicates that there is a positive correlation between smooth degree and the total cost of a solution, which means that the smaller smooth degree is, the smaller total cost is. The following experiments show that non-smooth penalty can enhance the construction algorithms. Two new construction algorithms (PSRP, PSEP) and four traditional construction methods are used as the alternative kernel method of PRM, which can find a better solution in a large step. The kernel algorithm of PRM is selected according to the ranks of the candidate construction algorithms according to the smooth degrees of the best solutions they ever find in the initialization process of algorithm. According to the comparison analysis between the optimal and the suboptimal solutions of instances *gdb9* and *gdb23*, a non-smoothed phenomenon is discovered which is described as Eqs. (3),(4). The non-smooth problem of *gdb23* is one example whose a bit larger smooth degree is caused by a larger number of routes. Then reinsert strategy is designed to solve this problem. But RiM method is not always beneficial and cannot improve the solution already with the minimum number of routes. Then an MA framework with non-smooth penalty (MANSP) is proposed. Extensive experiments show that MANSP can solve the non-smooth problem well based on four CARP instances sets and the performance of MANSP is very competitive even comparing with state-of-the-art algorithms.

Based on comparison analysis, a matching kernel algorithm is indeed required for an algorithm. So one of the future work is to seek a more competitive kernel algorithm. In addition, how to find a way to reduce the number of routes and improve the quality are also the possible directions. The code of this paper can be found in https://github.com/zmdsn/MANSP for research.

CRediT authorship contribution statement

Rui Li: Conceptualization, Methodology, Writing, Revising. **Xinchao Zhao:** Conceptualization, Revising, Supervision. **Xingquan Zuo:** Methodology, Suggestion. **Jianmei Yuan:** Suggestion, Reviewing. **Xin Yao:** Conceptualization, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The first two authors will express our awfully thanks to all the members of Swarm Intelligence Research group of BeiYou University. The first author Rui Li will express his awful thanks to Prof. Xin Yao, Ke Tang, Yuhui Shi and the research group of the Southern University of Science and Technology for their help and guidance when he was visiting there for one and a half year as a visiting student. Many thanks are also for Changwu Huang and Dongbin Jiao, for their guidance and help in study and life in Shenzhen.

Appendix. Comparison between MAENS and MANSP on *bmcv* benchmark set

See Tables 12–15.

Table 15
Comparison results on the F subset in *BMCV* benchmark set, 'min', 'mean', 'std' are the minimum, mean and standard deviation values of 30 results, 'NS' is the succeeding number of finding the lower bound LB [47–49].

Instance	LB	MAENS			QICA	MANSP			
		Repaired	Published	Offset		Min	Mean	Std	NS
F1	4040	4040∼	1065	2975	4040∼	4040	4048.2	2	30
F2	3300	3300∼	920	2380	3300∼	3300	3300	0	30
F3	1665	1665∼	400	1265	1665∼	1665	1665	0	30
F4	3476	3495↑	950	2545	3435↓	3485	3509.2	7	21
F5	3605	3605∼	1180	2425	3600∼	3605	3605.2	0	19
F6	1875	1875∼	490	1385	1875∼	1875	1875	0	30
F7	3335	3335∼	1080	2255	3335∼	3335	3335	0	30
F8	3690	3705∼	1145	2560	3705∼	3705	3705	0	30
F9	4730	4730∼	1145	3585	4730∼	4730	4780.8	31	24
F10	2925	2925∼	1010	1915	2925∼	2925	2925	0	30
F11	3835	3835∼	1015	2820	3835∼	3835	3861.7	8	20
F12	3390	3395∼	910	2485	3395∼	3395	3453.5	22	23
F13	2855	2855∼	835	2020	2855∼	2855	2855	0	30
F14	3330	3340↑	1035	2305	3330∼	3330	3350.5	9	15
F15	3560	3560∼	945	2615	3560∼	3560	3562	4	22
F16	2725	2725∼	775	1950	2725∼	2725	2725	0	30
F17	2055	2055∼	605	1450	2055∼	2055	2055	0	30
F18	3063	3075∼	850	2225	3065∼	3075	3075	0	30
F19	2500	2525∼	725	1800	2525∼	2525	2525	0	30
F20	2445	2445∼	610	1835	2445∼	2445	2445.2	0	28
F21	2930	2930∼	905	2025	2930∼	2930	2930	0	30
F22	2075	2075∼	790	1285	2075∼	2075	2075	0	30
F23	2994	3005∼	725	2280	3000∼	3005	3010.7	5	25
F24	3210	3240∼	1005	2235	3210↓	3240	3255.5	7	0
F25	1390	1390∼	430	960	1390∼	1390	1390	0	30
%		0.16			0.20	0.10			

"%" represents the average deviation to the LB(%).

References

[1] M. Dror, Arc Routing: Theory, Solutions and Applications, Springer US, Boston, MA, 2000.

[2] B. Golden, S. Raghavan, E. Wasil, The Vehicle Routing Problem: Latest Advances and New Challenges, Springer, Boston, MA, 2008.

[3] H. Handa, L. Chapman, X. Yao, Applications notes - Robust route optimization for gritting/salting trucks: A CERCIA experience, IEEE Comput. Intell. Mag. 1 (1) (2006) 6–9.

[4] P. Lacomme, C. Prins, W. Ramdane-Cherif, Evolutionary algorithms for periodic arc routing problems, European J. Oper. Res. 165 (2) (2005) 535–553.

[5] E.B. Tirkolaee, I. Mahdavi, M.M.S. Esfahani, A robust periodic capacitatedarc routing problem for urban waste collection considering drivers and crew working time, Waste Manage. 76 (2018) 138–146.

[6] E.B. Tirkolaee, A. Goli, M. Pahlevan, R.M. Kordestanizadeh, A robust bi-objective multi-trip periodic capacitated arc routing problem for urban waste collection using a multi-objective invasive weed optimization, Waste Manage. Res. 37 (11) (2019) 1089–1101.

[7] P. Li, M. Yun, J. Tian, et al., Stacked dense networks for single-image snow removal, Neurocomputing 367 (2019) 152–163.

[8] A. Khajepour, M. Sheikhmohammady, E. Nikbakhsh, Field path planning using capacitated arc routing problem, Comput. Electron. Agric. 173 (2020) 105401.

[9] Y. Zhang, Y. Mei, K. Tang, K. Jiang, Memetic algorithm with route decomposing for periodic capacitated arc routing problem, Appl. Soft Comput. 52 (2017) 1130–1142.

[10] G.V. Batista, C.T. Scarpin, J.E. Percora, A. Ruiz, A new ant colony optimization algorithm to solve the periodic capacitated arc routing problem with continuous moves, Math. Probl. Eng. 2019 (2019) 1–12, Article 3201656.

[11] D. Cattaruzza, N. Absi, D. Feillet, Vehicle routing problems with multiple trips, Ann. Oper. Res. 271 (2018) 127–159.

[12] Y. Sun, D. Wang, Ma. Lang, X. Zhou, Solving the time-dependent multi-trip vehicle routing problem with time windows and an improved travel speed model by a hybrid solution algorithm, Cluster Comput. 22 (2019) 15459–15470.

[13] Y. Mei, K. Tang, X. Yao, Decomposition-based memetic algorithm for multiobjective capacitated arc routing problem, IEEE Trans. Evol. Comput. 15 (2) (2011) 151–165.

[14] L. Delgado-Antequera, R. Caballero, J. Sanchez-Oro, et al., Iterated greedy with variable neighborhood search for a multiobjective waste collection problem, Expert Syst. Appl. 145 (2020) 113101.

[15] B.L. Golden, R.T. Wong, Capacitated arc routing problems, Networks 11 (3) (1981) 305–315.

[16] K. Tang, J. Wang, X. Li, X. Yao, A scalable approach to capacitated arc routing problems based on hierarchical decomposition, IEEE Trans. Cybern. 47 (11) (2017) 3928–3940.

[17] R. Shang, B. Du, K. Dai, L. Jiao, Y. Xue, Memetic algorithm based on extension step and statistical filtering for large-scale capacitated arc routing problems, Nat. Comput. 17 (2) (2018) 375–391.

[18] R. Shang, K. Dai, L. Jiao, R. Stolkin, Improved memetic algorithm based on route distance grouping for multiobjective large scale capacitated arc routing problems, IEEE Trans. Cybern. 46 (4) (2016) 1000–1013.

[19] J. de Armas, P. Keenan, A.A. Juan, S. McGarraghy, Solving large-scale time capacitated arc routing problems: from real-time heuristics to metaheuristics, Ann. Oper. Res. 273 (2019) 135–162.

[20] G. Marques, R. Sadykov, J.-C. Deschamps, R. Dupas, An improved branch-cut-and-price algorithm for the two-echelon capacitated vehicle routing problem, Comput. Oper. Res. 114 (2020) 104833.

[21] B.L. Golden, J.S. Dearmon, E.K. Baker, Computational experiments with algorithms for a class of routing problems, Comput. Oper. Res. 10 (1) (1983) 47–59.

[22] G. Ulusoy, The fleet size and mix problem for capacitated arc routing, European J. Oper. Res. 22 (3) (1985) 329–337.

[23] A. Hertz, G. Laporte, M. Mittaz, A tabu search heuristic for the capacitated arc routing problem, Oper. Res. 48 (1) (2000) 129–135.

[24] A. Hertz, M. Mittaz, A variable neighborhood descent algorithm for the undirected capacitated arc routing problem, Transp. Sci. 35 (4) (2001) 425–434.

[25] P. Beullens, L. Muyldermans, D. Cattrysse, D. Van Oudheusden, A guided local search heuristic for the capacitated arc routing problem, European J. Oper. Res. 147 (3) (2003) 629–643.

[26] P. Lacomme, C. Prins, W. Ramdane-Cherif, A genetic algorithm for the capacitated arc routing problem and its extensions, in: EvoWorkshops 2001: Applications of Evolutionary Computing, LNCS2037, 2001, pp. 473–483.

[27] K. Tang, Y. Mei, X. Yao, Memetic algorithm with extended neighborhood search for capacitated arc routing problems, IEEE Trans. Evol. Comput. 13 (5) (2009) 1151–1166.
[28] P. Lacomme, C. Prins, W. Ramdane-Cherif, Competitive memetic algorithms for arc routing problems, Ann. Oper. Res. 131 (1–4) (2004) 159–185.
[29] B. Peng, Y. Zhang, Z. Lv, T.C.E. Cheng, F. Glover, A learning-based memetic algorithm for the multiple vehicle pickup and delivery problem with LIFO loading, Comput. Ind. Eng. 142 (2020) 106241.
[30] Y. Wang, L. Wang, G. Chen, Z. Cai, Y. Zhou, L. Xing, An improved ant colony optimization algorithm to the periodic vehicle routing problem with time window and service choice, Swarm Evol. Comput. 55 (2020) 100675.
[31] H. Zhang, Q. Zhang, L. Ma, Z. Zhang, Y. Liu, A hybrid ant colony optimization algorithm for a multi-objective vehicle routing problem with flexible time windows, Inform. Sci. 490 (2019) 166–190.
[32] R. Shang, B. Du, K. Dai, L. Jiao, A.M.G. Esfahani, R. Stolkin, Quantum-inspired immune clonal algorithm for solving large-scale capacitated arc routing problems, Memetic Comput. 10 (1) (2018) 81–102.
[33] K. Bi, Y. Chen, C.-H. (John) Wu, D. Ben-Arieh, A memetic algorithm for solving optimal control problems of zika virus epidemic with equilibriums and backward bifurcation analysis, Commun. Nonlinear Sci. Numer. Simul. 84 (2020) 105176.
[34] S. Wang, L. Wang, An estimation of distribution algorithm-based memetic algorithm for the distributed assembly permutation flow-shop scheduling problem, IEEE Trans. Syst. Man Cybern. Syst. 46 (1) (2016) 139–149.
[35] Z. Liao, W. Gong, L. Wang, Memetic niching-based evolutionary algorithms for solving nonlinear equation system, Expert Syst. Appl. 149 (2020) 113261.
[36] J.M. Belenguer, E. Benavent, P. Lacomme, C. Prins, Lower and upper bounds for the mixed capacitated arc routing problem, Comput. Oper. Res. 33 (12) (2006) 3363–3383.
[37] L. Santos, J. Coutinho-Rodrigues, J.R. Current, An improved heuristic for the capacitated arc routing problem, Comput. Oper. Res. 36 (9) (2009) 2632–2637.
[38] L.M.S. Bento, D.R. Boccardo, R.C.S. Machado, et al., Dijkstra graphs, Discrete Appl. Math. 261 (2019) 52–62.
[39] P. Moscato, On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Toward Memetic Algorithms, Rep. 826, in: Caltech Concurrent Comput. Program, CalTech, Pasadena, CA, 1989.
[40] M. Gong, C. Chen, Y. Xie, S. Wang, Community preserving network embedding based on memetic algorithm, IEEE Trans. Emerg. Top. Comput. Intell. 4 (2) (2020) 108–118.
[41] J.C. Molina, J.L. Salmeron, I. Eguia, An ACS-based memetic algorithm for the heterogeneous vehicle routing problem with time windows, Expert Syst. Appl. 157 (2020) 113379.
[42] R. Kendy Arakaki, F. Luiz Usberti, An efficiency-based path-scanning heuristic for the capacitated arc routing problem, Comput. Oper. Res. 103 (2019) 288–295.
[43] J.S. DeArmon, A Comparison of Heuristics for the Capacitated Chinese Postman Problems (M.S. thesis), Univ. Maryland, College Park, MD, 1981.
[44] G. Rudolph, Convergence analysis of canonical genetic algorithms, IEEE Trans. Neural Netw. 5 (1) (1994) 96–101.
[45] E. Benavent, V. Campos, A. Corberan, E. Mota, The capacitated arc routing problem: Lower bounds, Networks 22 (7) (1992) 669–690.
[46] R.W. Eglese, Routeing winter gritting vehicles, Discrete Appl. Math. 48 (3) (1994) 231–244.
[47] J.M. Belenguer, E. Benavent, A cutting plane algorithm for the capacitated arc routing problem, Comput. Oper. Res. 30 (5) (2003) 705–728.
[48] R. Baldacci, V. Maniezzo, Exact methods based on node-routing formulations for undirected arc-routing problems, Networks 47 (1) (2006) 52–60.
[49] H. Longo, M.P. de, E. Uchoa, Solving capacitated arc routing problems using a transformation to the CVRP, Comput. Oper. Res. 33 (6) (2006) 1823–1837.
[50] C. Bode, S. Irnich, Cut-first branch-and-price-second for the capacitated arc-routing problem, Oper. Res. 60 (5) (2012) 1167–1182.