Towards Novel Meta-heuristic Algorithms for Dynamic Capacitated Arc Routing Problems

Hao Tong[1], Leandro L. Minku[1], Stefan Menzel[2], Bernhard Sendhoff[2], and Xin Yao[1]

[1] School of Computer Science, University of Birmingham
Edgbaston, Birmingham, B15 2TT, UK
{hxt922, L.L.Minku, x.yao}@cs.bham.ac.uk
[2] Honda Research Institute Europe GmbH, 63073 Offenbach, Germany
{stefan.menzel,bs}@honda-ri.de

Abstract. The Capacitated Arc Routing Problem (CARP) is an abstraction for typical real world applications, like waste collection, winter gritting and mail delivery, to allow the development of efficient optimization algorithms. Most research work focuses on the static CARP, where all information in the problem remains unchanged over time. However, in the real world, dynamic changes may happen when the vehicles are in service, requiring routes to be rescheduled. In this paper, we mainly focus on this kind of Dynamic CARP (DCARP). Some meta-heuristics solve (D)CARP by generating individuals that are sequences of tasks to be served as the individual representation. The split of this sequence into sub-sequences to be served by different vehicles needs to be decided to generate an executable solution, which is necessary for calculating individual's fitness. However, the existing split schemes for static CARP and DCARP are not capable of getting high quality feasible solutions for DCARP. Therefore, we propose two different split schemes in this paper – an optimal and a greedy split scheme. The optimal split scheme, assisted by A-star algorithm, can obtain the best vehicle routes from an ordered list. The greedy split scheme is not guaranteed to obtain optimal splits, but it is much more efficient. More importantly, it can keep the rank information between different individuals. Our experiments show that the greedy split scheme has good relative accuracy with respect to the optimal split scheme and that the two proposed split schemes are better than the existing DCARP split scheme in terms of the obtained solutions' quality.

Keywords: Dynamic CARP · Split scheme · A-star algorithm · Greedy search.

1 Introduction

The Capacitated Arc Routing Problem (CARP) is a classical and important combinatorial optimization problem, dealing with a set of edges in a graph served by a number of vehicles with limited capacity [4]. Consider a pre-defined graph

containing a set of vertices and edges, where every edge has a travel cost and some edges, called tasks, have demands required to be served once by vehicles. CARP's aim is to find an optimal routing schedule that assigns vehicles to serve all tasks once and only once, minimizing the total travel cost. There is a wide range of real world applications related to CARP, such as waste collection [7], winter gritting [6], mail delivery [2] and others.

Plenty of approaches have been proposed to handle the various CARP applications, including constructive heuristic methods [3, 18], efficient algorithms for different kinds of CARP [1, 7, 16], algorithms for large scale CARPs [11, 17], and algorithms for uncertain CARPs [16, 12]. Most existing work focuses on static scenarios, in which the condition of a CARP does not change once it is given. However, in the real world, CARP is likely to dynamically change during the service process of vehicles. For example, new tasks may be added or some edges may not be available any more. This is referred to as Dynamic CARP (DCARP), and was, to the best of our knowledge, firstly investigated by [15, 5]. Liu et al. [10] proposed a memetic algorithm with a new split scheme to solve DCARP, which included six factors, i.e. vehicle availability, road accessibility, added/cancelled tasks, road congestion and change in tasks' demands. After Liu et al., to the best of our knowledge, only one work related to DCARP was published [13], in which failure of vehicles is the only dynamics considered.

Many algorithms for CARP usually produce a sequence of tasks, i.e. the individual represented in the meta-heuristic and evolutionary algorithms is a sequence of tasks [7, 16]. A split scheme is applied to obtain the executable sub-routes (i.e., sub-sequences of tasks) to be served by different vehicles. Such executable sub-routes are necessary for evaluating the fitness of the individuals. Fitness evaluation, and in particular the split scheme used during fitness evaluation, is thus a key point when using sequence of tasks as the individual representation in meta-heuristics to solve DCARP. However, split schemes for static CARP are unsuitable for DCARP, because DCARP has to assign outside vehicles to serve the remaining tasks and return to the depot, instead of using only vehicles that are in the depot. The only existing split scheme proposed for DCARP uses a random-based operator. Among other problems, it (1) makes the fitness evaluation noisy and (2) leads to fitness values that are unlikely to correspond to the actual fitness of the solution to be adopted in practice.

In this paper, we propose two new split schemes toward meta-heuristic algorithms for DCARP: (1) an optimal split scheme whose time and space complexity are high, and (2) a greedy split scheme which is not optimal, but has lower time and space complexity, being more suitable for real world applications. We perform experiments showing that the greedy split scheme maintains the rank of individuals of the population much better than the exiting split scheme, and also showing that the greedy split scheme is much more efficient than the proposed optimal split scheme.

The remainder of this paper is organized as follows. Section 2 discusses related work on split schemes for (D)CARP. Section 3 presents the main procedures of

our two proposed split schemes. Section 4 presents experiments to evaluate the proposed and existing DCARP split schemes. Section 5 concludes the paper.

2 Related Work

As explained in Section 1, the representation of individuals for solving CARP by meta-heuristic algorithms is a sequence of tasks. To form a feasible CARP solution whose fitness can be computed, this sequence needs to be split into different sections, each of them to be served by a different vehicle. Given this work's focus on DCARP split schemes, this section concentrates on split schemes.

2.1 Split Scheme for Static CARP

For static CARP, the Ulusoy's scheme [18] is used, which can obtain an optimal split from an individual by building an auxiliary graph. Assume an individual $\{t_1, t_2, ..., t_{N_t}\}$ for CARP, Ulusoy's split builds an auxiliary graph, G^*, to find the optimal split. G^* contains $N_t + 1$ nodes, in which the first node represents the depot. Each edge (i, j) in the auxiliary graph represents a feasible sub-route, r_{sub}, of the CARP, and its cost is the weight, $w_{i,j}$, of the corresponding edge. For example, consider that edge (i, j) represents the route $r_{sub} = \{depot \rightarrow t_{i+1} \rightarrow t_{i+2} \rightarrow ... \rightarrow t_j \rightarrow depot\}$, and $w_{ij} = cost_{r_{sub}}$. Then, Ulusoy's split uses Dijkstra's algorithm to find the shortest path from first node to the last node in G^*, which indicates the optimal split for the assigned individual. Ulusoy's split cannot be used for DCARP, because different vehicles in DCARP start from different stop points in the intermediate states, whilst all targets are the depot.

Some studies extended Ulusoy's split to multi-depot CARP [19]. However, each edge in the auxiliary graph represents a route that must start and end at the same depot. Therefore, these split schemes are also not suitable for DCARP, where vehicles start from different stop points in the intermediate states.

2.2 Split Schemes for DCARP

Liu et al. [10] proposed the first split scheme for DCARP, called Distance Based Split Scheme (DSS). Consider a given individual $\{t_1, t_2, ..., t_{N_t}\}$. DSS randomly splits it into N_K sub-routes, where N_K is the number of vehicles used in service and is defined by the solution of the initial CARP instance. For each sub-route S_k, DSS calculates the sum of cost C_{pvj} of each vehicle j from its current position to each task's start node and end node:

$$C_{pvj} = \sum_{t_i}^{S_k} cost(v_j, start_{t_i}) + cost(v_j, end_{t_i}) \tag{1}$$

where v_j denotes the location of vehicle j. The vehicle with minimal C_{pvj} will be assigned to serve the S_k. After all vehicles are assigned to all tasks, the total

cost can be calculated for the whole schedule. Finally, DSS repeats the above process three times and selects the schedule with minimal total cost.

DSS has some significant weaknesses. Firstly, it makes the fitness evaluation noisy, as the fitness is not deterministic anymore and highly depend on the quality of the best among three random splits. Secondly, the fitness value obtained based on DSS is unlikely to correspond to the actual fitness of the solution to be adopted in practice. Thirdly, if the demand of a sub-route exceeds the remaining capacity of an assigned vehicle, DSS uses a path repair operator, where the vehicle returns back to the depot to get refilled and then goes back to continue serving the next task. This is inefficient to some extent, because other vehicles may still have big remaining capacities and could potentially be assigned to serve the remaining tasks that exceeded the original vehicle's capacity. Furthermore, DSS never considers new sub-routes starting from the depot, resulting in an inflexible schedule. In some cases, vehicles may not have enough capacity to serve the distant tasks after serving the near tasks so that they have to apply the repair operator to serve the distant tasks separately with a high cost. However, if it was allowed to create a new route for near tasks, vehicles would have enough capacity to serve all distant tasks simultaneously, avoiding to serve the distant tasks independently with a very high cost, and thus the total cost is reduced. Finally, DSS cannot guarantee an optimal split.

To overcome DSS' shortcomings, we propose two deterministic split schemes for DCARP. Both of them can handle the situation where vehicles start from different stop points, and they also consider that new vehicles can be assigned to serve tasks. One of them focuses on optimality and the other one on efficiency.

3 Proposed Split Schemes for DCARP Fitness Evaluation

For static CARP, Ulusoy's split finds the optimal split by building an auxiliary graph, in which the shortest path represents the optimal split. Inspired by this idea, our two proposed DCARP split schemes also mainly contain two steps: *auxiliary graph construction* and *path finding*. The two split schemes apply different path-finding strategies based on the same auxiliary graph.

3.1 Auxiliary Graph Construction

In the auxiliary graph for static CARP, the edge between any two nodes represents a sub-route, serving a set of tasks. For instance, edge e_{ij} represents a sub-route serving the task's set $\{t_{i+1}, t_{i+2}, ..., t_j\}$. Similarly, we also use an edge in the auxiliary graph to represent a sub-route for DCARP. Hence, for an ordered list of tasks, i.e., $t_1, t_2, ..., t_{N_t}$, there are $N_t + 1$ nodes in the auxiliary graph. However, different from the static CARP, we already have some vehicles outside of the depot, which can be assigned to serve the remaining tasks. They stopped in different positions when the change happened, and they have to start from these positions to serve the remaining tasks. Therefore, an edge (i, j) between two nodes represents different routes for different vehicles in the auxiliary

graph. As a result, we construct several edges between two nodes in the auxiliary graph to represent sub-routes for all outside vehicles. Besides, considering that new sub-routes starting from the depot can also be created[1], we add an additional edge between two nodes to represent the route starting from the depot. Algorithm 1 presents the procedure for building an auxiliary graph for DCARP.

Algorithm 1: Build auxiliary graph for DCARP

Input: Individual : $I = \{t_1, t_2, ..., t_N\}$
Stop points for outside vehicles: $V = \{v_1, v_2, ..., v_K\}$
Remaining capacity for outside vehicles: $CP = \{cp_1, cp_2, ..., cp_K\}$

```
1  Generate N + 1 Nodes (Index from 0 to N) for the auxiliary graph G*;
2  v_0 = depot, cp_0 = original capacity;
3  V = {v_0, v_1, v_2, ..., v_K}, CP = {cp_0, cp_1, cp_2, ..., cp_K};
4  for each vehicle k in V do
5      for each node pair: (Node_i, Node_j) do
6          Use vehicle k to serve {t_{i+1}, t_{i+2}, ..., t_j};
7          Sub-route: r_ijk = {v_k → t_{i+1} → t_{i+2}, → ..., → t_j → depot};
8          Calculate the total demand d_ijk of r_ijk;
9          if d_ijk > cp_k then
10             continue;
11         else
12             Calculate the cost of r_ijk: c_ijk;
13             Assign an edge e_ijk with weight c_ijk between Node_i and Node_j;
```

Output: An auxiliary graph G^*

Assume that K vehicles are currently outside the depot. The stop points and remaining capacities for vehicles are $V = \{v_1, v_2, ..., v_K\}$ and $CP = \{cp_1, cp_2, ..., cp_K\}$. For the additional edge for new vehicles, we add a stop point $v_0 = depot$ into V and cp_0, equal to the original capacity, into CP, in Line 2-3. For each pair of nodes $(Node_i, Node_j)$, we build an edge e_{ijk} for each vehicle k, which represents the sub-route $r_{ijk} = \{v_k \rightarrow t_{i+1} \rightarrow t_{i+2} \rightarrow ... \rightarrow t_j \rightarrow depot\}$, in Lines 6-7. However, if the total demand of r_{ijk} exceeds the vehicle's remaining capacity, edge e_{ijk} will be removed due to the capacity constrain, in Lines 8-10. Otherwise, the weight of e_{ijk} is assigned with the cost of r_{ijk}, c_{ijk} in Lines 12-13.

3.2 A-Star Based Optimal Split Scheme

Path Finding: For static CARP, the Dijkstra algorithm is used directly to find the shortest path in the auxiliary graph. However, the number of edges in DCARP is much larger, as explained in Section 3.1, because there is a different

[1] New routes could potentially be served by a new vehicle (if we extra vehicles are available), or by an outside vehicle (after it finishes serving its currently assigned tasks and returns to the depot).

edge (i, j, k) for each vehicle k between the pair of nodes (i, j). To increase efficiency, we adopt the A-star algorithm to find the optimal path, instead of Dijkstra.

There are also two important constraints which have to be considered in DCARP and which did not exist in static CARP. First, any two edges in the whole path cannot belong to the same outside vehicle. Otherwise, the outside vehicle would have to serve two sub-routes, starting from the same outside stop point. However, when the vehicle finishes one sub-route, it stops at the depot. Therefore, it is impossible for this vehicle to serve another sub-route starting from an outside stop point. Secondly, all outside vehicles have to return to the depot even if they are not assigned to serve tasks in the new schedule. This means that, if no edge is assigned to a given outside vehicle in the whole path, the cost of this vehicle returning to the depot still needs to be considered in the total final split cost.

To use A-star, we need to determine a suitable admissible heuristic function $h(n)$. A heuristic is admissible if the cost it retrieves is smaller than or equal to the actual minimal cost to reach the target node in the tree from n [14]. In our problem, we use the minimal cost from the current node to the final node in the auxiliary graph without considering the constraints as the heuristic function. The cost function $g(n)$ is calculated according to the actual path to reach n. In our split scheme, when different paths arrive at the same node in the auxiliary graph, the path with better cost will not replace the worse one because the choice of vehicles before influences the cost from the current node to the target. Therefore, the tree-based A-star search [14] is used in our split scheme. The procedure of the A-star-based optimal split scheme is presented in Algorithm 2.

The split scheme builds an auxiliary graph in Line 1 for shortest path finding. From the first node in the auxiliary graph, A-star expands each selected node with minimal $f(n)$ and finds its successors, in Lines 8-9 and Line 20. The estimated costs $f(n)$ for all successors are calculated in Lines 13-14. During the procedure of expanding the current node, the A-star-based split scheme applies two strategies to handle the two constraints for DCARP. Firstly, in order to avoid the repetition of edges belonging to the same vehicle, it removes all edges belonging to the vehicles which have already been selected in the current part of the path and then explores the rest of path, in Line 12. For the second constraint, when A-star reaches the final node, we repair the cost of the final path if some outside vehicles are never selected, adding the returning cost for these non-selected outside vehicles in Lines 15-16.

Complexity Analysis: A-star guarantees the optimality of the path found. However, it has a high space and time complexity since there are N tasks in total. For static CARP, the number of edges is $\#E_S = \frac{N(N-1)}{2}$, and the number of all possible paths is $\#P_S = 2^{N+1}$ in the auxiliary graph. In the auxiliary graph for DCARP, if there are K outside vehicles, the number of edges is

$$\#E_D = \frac{(K+1) \cdot N(N-1)}{2}$$

Algorithm 2: A* based optimal split scheme

Input: Individual : $I = \{t_1, t_2, ..., t_N\}$
Build an auxiliary graph G^* for DCARP;
$expandNode = Node_0$; openNodeSet = {}; pathSet = {};
while *True* **do**
  if *expandNode == target* **then**
    Shortest path P: path correspond to $expandNode$;
    Minimal cost C: $f_{expandNode}$ correspond to $expandNode$;
    break;
  Select *rootPath* (i.e. path from $Node_0$ to $expandNode$) from pathSet;
  Find all feasible successors for $expandNode$ in the graph;
  for *each successor of expandNode* **do**
    $newPath = rootPath + expandNode \rightarrow successor$;
    Remove all edges belong to vehicles used in *newPath* for *successor*;
    Calculate the h_{succ} and g_{succ};
    Set $f_{succ} = h_{succ} + g_{succ}$;
    if *successor == target* **then**
      Repair f_{succ};
    Add the *successor* into openNodeSet;
    Add the *newPath* into pathSet;
  Remove *expandNode*, *rootPath* from openNodeSet and pathSet;
  Select the node in openNodeSet with minimal f as $expandNode$;
The shortest path from $Node_0$ to *target* in G^*: $P = \{p_1, p_2, ..., p_M\}$;
Each p_m represents an edge e_{ijk}, which denotes a sub-route r_{ijk};
Obtain the solution S by splitting the I by P.
Output: Solution $S = \{r_1, r_2, ..., r_M\}$, Minimal cost: C

and the number of all possible paths, i.e. from the first node to the target in the auxiliary graph, is

$$\#P_D = \sum_{n=1}^{N} \mathrm{C}_{N-1}^{n-1} \sum_{i=0}^{\mathbf{min}(n,K)} \mathrm{C}_{n}^{n-i} \cdot \mathrm{P}_{K}^{i}$$

Therefore, the number of routes in the auxiliary graph for DCARP is much larger than in the static case. The A-star algorithm has to save all expanded nodes during the search process. In the worst case, it will visit and save all $\#P_D$ possible paths. Even though heuristics can frequently avoid the worst case scenario, the computational time still depends on $\#P_D$ and is often still unacceptably high, as will be demonstrated by our experiments in Section 4.3.

3.3 Greedy split scheme

Path Finding: As discussed above, the A-star based optimal split scheme is computationally expensive when using A-star search to find the optimal path in

the auxiliary graph. Therefore, we propose a greedy strategy to find a path with a good quality in the auxiliary graph.

In the auxiliary graph, each edge e_{ijk} represents a route r_{ijk}, serving a list of tasks and having a cost c_{ijk}. So, we can obtain an efficiency parameter for each route, which is the average cost for each demand of this route. Generally, when there are several possible routes to be selected, we will choose the route with the lowest average cost for each demand, from a greedy perspective. The greedy procedure is presented in Algorithm 3.

Algorithm 3: Greedy split scheme

Input: Individual : $I = \{t_1, t_2, ..., t_N\}$
1 Build an auxiliary graph G^* for DCARP;
2 **for** *each edge* e_{ijk} *in* G^* **do**
3 Calculate the ACD: ACD_{ijk};
4 $expandNode = Node_0$; $newPath = Node_0$
5 **while** *True* **do**
6 **if** $expandNode == target$ **then**
7 Greedy path: $newPath$, $P = \{p_1, p_2, ..., p_M\}$;
8 Calculate the greedy cost of greedy path: C;
9 break;
10 $rootPath \leftarrow newPath$;
11 Find all edges linking to $successors$ for $expandNode$;
12 Select the edge with the minimal ACD;
13 $Node_X \leftarrow successor$ that the selected edge belongs to;
14 $newPath = rootPath + expandNode \rightarrow Node_X$;
15 Remove all edges corresponding to vehicles being used in $newPath$;
16 $expandNode \leftarrow Node_X$;
17 Each p_m represents an edge e_{ijk}, which denotes a sub-route r_{ijk};
18 Obtain the solution S by splitting the I by P.
Output: Solution $S = \{r_1, r_2, ..., r_m\}$, Greedy cost: C

Assuming the ordered tasks $\{t_1, t_2, ..., t_{N_t}\}$, we build an auxiliary graph according to Algorithm 1 (Line 1). Then, we calculate the average cost for each demand (ACD), for each edge e_{ijk}, as ACD_{ijk}, in Lines 2-3. In each step, the greedy split scheme will select the edge with the minimal ACD from all edges linking to the current node, in Lines 11-12. Similarly, in order to satisfy the constraints, when an edge belonging to one outside vehicle is selected in each step, it will remove all edges corresponding to this vehicle in the later path-finding process, in Line 15. Finally, the process terminates when the target is found, and then the actual cost for the greedy path is calculated, considering the return cost of any unused outside vehicles, in Lines 7-8.

Complexity Analysis: The greedy split scheme is much more efficient than A-star-based optimal split scheme. Without considering the auxiliary graph construction, its time complexity is only $\mathcal{O}(N_t)$.

4 Experiments

The greedy split scheme has much lower time complexity than the optimal split scheme, but may lead to splits of lower quality. Given that the split scheme will be used as part of the fitness evaluation of individuals within a meta-heuristic algorithm, it is desirable that better individuals according to the optimal split scheme are still considered as better individuals according to the greedy split scheme. In particular, if the ranking (relative accuracy) of individuals does not change when adopting the greedy instead of the optimal split scheme, the lower quality of the splits obtained by the greedy split scheme will not negatively affect the meta-heuristic algorithm, depending on the selection mechanisms being used. Therefore, in this section, we compare the relative accuracy of the greedy split scheme (GSS) with that of the existing split scheme, the distance-based split scheme (DSS) [9]. In addition, we also compare the fitness and computational time for different split schemes.

4.1 Comparison on relative accuracy

At first, we test three difference split schemes in a series of problem instances to show the relative accuracy of GSS and DSS, relative to optimal split schemes (OSS). Two benchmark sets, referred as *gdb* set [3] and *egl* set [8], for static CARP are used as basis map for DCARP in our experiments. Each benchmark generates one scenario for simulating the situation of changes happening, where the changes include broken down vehicles, closed roads, roads congestion, added tasks and increased demands. After the changes, *gdb* and *egl* instances have on average 174 and 37 tasks, representing a high and a low dimensional set of instances, respectively. For each scenario after the changes, $N_p = 40$ individuals are randomly generated and evaluated by different split schemes.

In order to compare the relative accuracy, we use the Kendall rank correlation coefficient (τ) as the measurement. Assuming that the fitnesses of individual i are f_{OSS}, f_{GSS} and f_{DSS}, τ can be calculated as

$$\tau = \frac{2}{N_p(N_p-1)} \sum_{i<j} \text{sgn}(f_{OSS_i} - f_{OSS_j})\text{sgn}(f_{SS_i} - f_{SS_j}) \tag{2}$$

where f_{SS_i} denotes f_{GSS_i} or f_{DSS_i}, and N_p is the size of population. $\tau \in [-1, 1]$ and a large τ indicates the relative accuracy of one split schemes is highly agreed with the optimal split scheme.

The results on relative accuracy are presented in Figure 1. The left part belongs to the *egl* set of benchmarks, whose dimension, i.e. the number of tasks, is larger than that of the *gdb* set in the right part. From the result, GSS is better than DSS with respect to the relative accuracy, especially in high dimension. It is mainly because DSS splits the individuals by random selection. The fitness obtained highly depends on the random seed, which influences the relative accuracy deeply. For example, for two individuals, the OSS can provide the relative fitness, but the better individual may be splitted into a low quality solution

due to a ill-chosen random seed. By contrast, the GSS split an individual on a deterministic way and there is no randomness in the split procedure, so that its relative accuracy is higher than DSS.

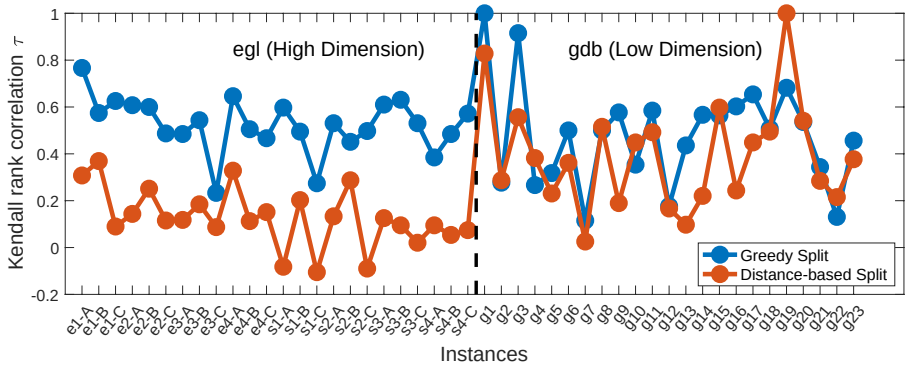

Fig. 1. Relative accuracy achieved by GSS and DSS.

However, when the problem is of low dimension, the difference in relative accuracy is small, because the search space for split becomes small. The result of one random split is close to another random split, so that the influence of randomness decreases.

4.2 Comparison on obtained fitness

For the same individual, GSS and DSS obtain different solutions, with different fitness values, i.e total cost. Therefore, in this subsection, we compare the performance of GSS and DSS, with respect to the fitness values. For each population with N_p individuals, as shown in the previous subsection, we compare every fitness obtained by GDD and DSS. Wilcoxon Sign Rank tests with a 0.05 significance level were carried out to compare the fitness values of the individuals and the ratio of individuals where GSS was better than DSS in the population is shown in Figure 2, where the blue(red) solid stems represent that GSS(DSS) performs significantly better than DSS(GSS) and the dash stems represents that both split schemes have no significant difference. In conclusion, GSS performs better than DSS significantly in most DCARP cases.

We can see that in all high dimensional cases, the fitness values obtained by GSS are almost all better than those of DSS in each population. However, GSS's performance decreases compared with that of DSS in low dimensional cases. There are two main reasons to explain the results. First, as discussed in the previous subsection, the solution obtained by DSS highly depends on the random seed. When the problem dimension is very low, the random split is likely to obtain a very good solution thanks to its diversity and the small search space.

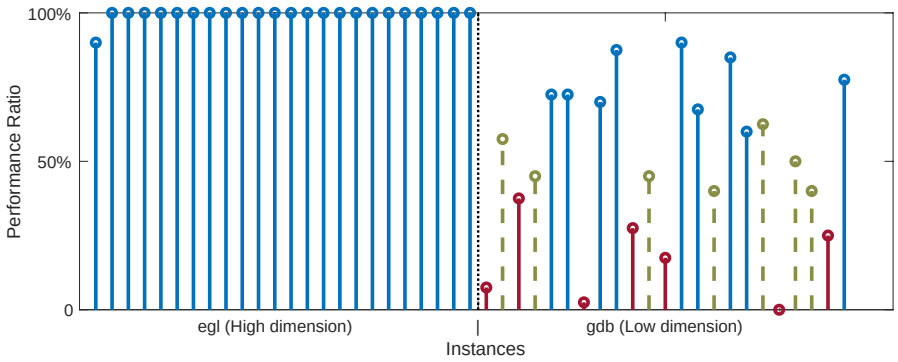

Fig. 2. Ratio of individuals where GSS is better than DSS in the population. Solid stems represent DSS and GSS have significant difference and dash stems represent they have no significant difference.

However, it cannot help a lot in the high dimensional cases. Second, when a given vehicle's capacity is exceeded due to a change, DSS uses a repair operator to make the vehicle return to the depot first and then continue to serve the remaining tasks from the depot. This typically results in higher costs than if new sub-routes starting from the depot could be created during the split process as done by our approaches, given that new sub-routes not only directly help to satisfy the capacity constraints, but also are optimized together with the sub-routes starting from outside, which makes our split more flexible and efficient.

4.3 Comparison on computational time

Finally, in this subsection, we will compare the computational time for three split schemes. We select 8 maps for each dataset as test scenario. As discussed in Section 3, OSS might require much time to obtain the solution. Therefore, in the previous experiments, the scenarios we generated were suitable for OSS to obtain the result within 300 seconds, to have enough results for the comparisons. In this section, we randomly generate the test scenarios. If OSS is unable to find the optimal solution within 300 seconds, the experiment is considered as a failure and the computational time is not considered. The computational results are presented in Table 1 , in which the time for GSS and OSS contains the time for auxiliary graph construction.

GSS is the most efficient method among the three split schemes, and OSS is the most computationally expensive in all test scenarios. GSS starts from the first node in the auxiliary graph, and selects the edge with minimal ACD. DSS splits the individuals randomly. Although the random split is very efficient, the repair operator is required to obtain the total cost, which is a relatively computationally expensive process. OSS has a very large search space, and the A-star algorithm will save all explored nodes, which causes its computational time to be very large in some cases. From the results, we can also observe that

Table 1. The computational time (in *seconds*) for OSS, GSS, DSS. For OSS, the success rate is shown in brackets. N_t is the number of tasks (dimension) and K is the number of outside vehicles.

Instances	N_t	K	t_{GSS}	t_{DSS}	t_{OSS}
egl-e1-A	35	2	0.03	0.70	0.22 (40/40)
egl-e2-A	39	3	0.03	0.66	0.77 (40/40)
egl-e3-A	29	4	0.03	0.42	9.91 (2/40)
egl-e4-A	38	6	0.04	0.64	27.3 (8/40)
egl-s1-A	70	7	0.07	1.38	N/A (0/40)
egl-s2-A	86	13	0.11	1.49	N/A (0/40)
egl-s3-A	66	9	0.08	1.08	N/A (0/40)
egl-s4-A	64	4	0.04	0.88	6.78 (1/40)

Instances	N_t	K	t_{GSS}	t_{DSS}	t_{OSS}
gdb2	3	3	0.04	0.08	0.13 (6/6)
gdb5	2	3	0.02	0.06	0.03 (2/2)
gdb8	20	7	0.03	0.29	11.4 (6/40)
gdb9	16	5	0.02	0.29	0.86 (39/40)
gdb16	7	1	0.02	0.15	0.05 (40/40)
gdb18	14	3	0.02	0.27	0.07 (40/40)
gdb22	15	5	0.02	0.28	4.58 (38/40)
gdb23	21	4	0.02	0.36	26.5 (36/40)

the number of tasks and outside vehicles have a big impact on the computational time for OSS. When there are many outside vehicles, the search space becomes very large, so that it is very hard to find the optimal split results for OSS. This is because the tasks and outside vehicles directly determine the search space as shown in Eq. (23).

5 Conclusion

The split scheme is essential for fitness evaluation in DCARP. However, the existing split scheme for static CARP is unsuitable for DCARP, and the existing DCARP splitting scheme highly depends on the random seed, being unable to provide stable results. Therefore, in this paper, we propose two new split schemes. The first split scheme is an optimal split scheme based on the A-star search. It is capable to provide an optimal solution for an ordered list of tasks. However, it is computationally expensive in many scenarios due to the huge search space. The second is a greedy split scheme, which is much more efficient than the optimal split scheme and even than the existing random split scheme. Our experiments show that the greedy split scheme is capable of leading to similar individual rankings to the optimal split scheme, and its fitness is much better than that of the existing random split scheme for DCARP, especially in high dimensional test cases.

Future work includes the testing of proposed split schemes in more real instances and the design of meta-heuristic methods to solve DCARP using the proposed split schemes.

Acknowledgements

Hao Tong gratefully acknowledges the financial support from Honda Research Institute Europe (HRI-EU).

References

1. Brandão, J., Eglese, R.: A deterministic tabu search algorithm for the capacitated arc routing problem. Computers & Operations Research **35**(4), 1112–1126 (2008)
2. Eiselt, H.A., Gendreau, M., Laporte, G.: Arc routing problems, part ii: The rural postman problem. Operations research **43**(3), 399–414 (1995)
3. Golden, B.L., DeArmon, J.S., Baker, E.K.: Computational experiments with algorithms for a class of routing problems. Computers & Operations Research **10**(1), 47–59 (1983)
4. Golden, B.L., Wong, R.T.: Capacitated arc routing problems. Networks **11**(3), 305–315 (1981)
5. Handa, H., Chapman, L., Yao, X.: Dynamic salting route optimisation using evolutionary computation. In: 2005 IEEE Congress on Evolutionary Computation. vol. 1, pp. 158–165. IEEE (2005)
6. Handa, H., Chapman, L., Yao, X.: Robust route optimization for gritting/salting trucks: A cercia experience. IEEE Computational Intelligence Magazine **1**(1), 6–9 (2006)
7. Lacomme, P., Prins, C., Ramdane-Chérif, W.: Evolutionary algorithms for periodic arc routing problems. European Journal of Operational Research **165**(2), 535–553 (2005)
8. Li, L.Y., Eglese, R.W.: An interactive algorithm for vehicle routeing for winter—gritting. Journal of the Operational Research Society **47**(2), 217–228 (1996)
9. Liu, M., Singh, H.K., Ray, T.: A benchmark generator for dynamic capacitated arc routing problems. In: 2014 IEEE Congress on Evolutionary Computation (CEC). pp. 579–586. IEEE (2014)
10. Liu, M., Singh, H.K., Ray, T.: A memetic algorithm with a new split scheme for solving dynamic capacitated arc routing problems. In: 2014 IEEE Congress on Evolutionary Computation (CEC). pp. 595–602. IEEE (2014)
11. Mei, Y., Li, X., Yao, X.: Cooperative coevolution with route distance grouping for large-scale capacitated arc routing problems. IEEE Transactions on Evolutionary Computation **18**(3), 435–449 (2013)
12. Mei, Y., Tang, K., Yao, X.: A global repair operator for capacitated arc routing problem. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) **39**(3), 723–734 (2009)
13. Monroy-Licht, M., Amaya, C.A., Langevin, A., Rousseau, L.M.: The rescheduling arc routing problem. International Transactions in Operational Research **24**(6), 1325–1346 (2017)
14. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach (3rd Edition). Prentice Hall, third edn. (2010)
15. Tagmouti, M., Gendreau, M., Potvin, J.Y.: A dynamic capacitated arc routing problem with time-dependent service costs. Transportation Research Part C: Emerging Technologies **19**(1), 20–28 (2011)
16. Tang, K., Mei, Y., Yao, X.: Memetic algorithm with extended neighborhood search for capacitated arc routing problems. IEEE Transactions on Evolutionary Computation **13**(5), 1151–1166 (2009)
17. Tang, K., Wang, J., Li, X., Yao, X.: A scalable approach to capacitated arc routing problems based on hierarchical decomposition. IEEE Transactions on Cybernetics **47**(11), 3928–3940 (2016)
18. Ulusoy, G., et al.: The fleet size and mix problem for capacitated arc routing. European Journal of Operational Research **22**(3), 329–337 (1985)

19. Xing, L., Rohlfshagen, P., Chen, Y., Yao, X.: An evolutionary approach to the multidepot capacitated arc routing problem. IEEE Transactions on Evolutionary Computation **14**(3), 356–374 (2009)