

## What makes the dynamic capacitated arc routing problem hard to solve

Tong, Hao; Minku, Leandro; Menzel, Stefan; Senhoff, Bernhard; Yao, Xin

DOI:

[10.1145/3512290.3528756](https://doi.org/10.1145/3512290.3528756)

*Document Version*

Peer reviewed version

*Citation for published version (Harvard):*

Tong, H, Minku, L, Menzel, S, Senhoff, B & Yao, X 2022, What makes the dynamic capacitated arc routing problem hard to solve: insights from fitness landscape analysis. in JE Fieldsend (ed.), *GECCO '22: Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO: Genetic and Evolutionary Computation Conference, Association for Computing Machinery (ACM), New York, pp. 305-313, GECCO '22: Genetic and Evolutionary Computation Conference, Boston, Massachusetts, United States, 9/07/22.  
<https://doi.org/10.1145/3512290.3528756>

[Link to publication on Research at Birmingham portal](#)

### General rights

Unless a licence is specified above, all rights (including copyright and moral rights) in this document are retained by the authors and/or the copyright holders. The express permission of the copyright holder must be obtained for any use of this material other than for purposes permitted by law.

- Users may freely distribute the URL that is used to identify this publication.
- Users may download and/or print one copy of the publication from the University of Birmingham research portal for the purpose of private study or non-commercial research.
- User may use extracts from the document in line with the concept of 'fair dealing' under the Copyright, Designs and Patents Act 1988 (?)
- Users may not further distribute the material nor use it for the purposes of commercial gain.

Where a licence is displayed above, please note the terms and conditions of the licence govern your use of this document.

When citing, please reference the published version.

### Take down policy

While the University of Birmingham exercises care and attention in making items available there are rare occasions when an item has been uploaded in error or has been deemed to be commercially or otherwise sensitive.

If you believe that this is the case for this document, please contact [UBIRA@lists.bham.ac.uk](mailto:UBIRA@lists.bham.ac.uk) providing details and we will remove access to the work immediately and investigate.

---

**Algorithm 1: VND-CARP**

---

```
1 Generate tour by Frederickson heuristic.
2 Apply SHORTEN and CUT to obtain an initial solution.
3 while True do
4   Set  $i = D/Q$ ,  $s_{best} = s$ ;
5   while True do
6     Set number of neighbours:  $c = 1$ .
7     Set best value of a neighbour:  $f_{best} = f(s)$ .
8     while  $c \leq M_i$  do
9       Select  $i$  routes in  $s$ , merge them into a giant tour.
10      Apply SWITCH and then CUT on this tour.
11      Apply SHORTEN on each new tour.
12      New resulting solution:  $s'$ ;
13      if  $f(s') < f_{best}$  then
14        Set  $best_s = s'$ ;
15        Set  $f_{best} = f(s')$ ;
16       $c = c + 1$ ;
17      if  $f_{best} < f(s)$  then
18        Set  $s = best_s$ ;
19      else
20         $i = i - 1$ 
21        if  $i \geq 1$  then
22          break;
23    if  $f(s) \geq f(s_{best})$  then
24      break;
```

---

---

**Algorithm 2: ILMA**

---

```
1 Initialization:  $nc - 3$  chromosomes;  
2 Add one chromosome by Path_Scanning;  
3 Add one chromosome by Augmnt-Merge;  
4 Add one chromosome by Ulusoy's split;  
5 while Stop criterion is not met do  
6   Select two chromosomes  $P_1$  and  $P_2$  by binary tournament selection;  
7   Apply ordered crossover operator to  $P_1$  and  $P_2$  to generate  $O_x$ ;  
8   Set  $O = O_x$ ;  
9   if  $rand() < P_{ls}$  then  
10    apply local search to  $O_x$  to generate  $O_m$ ;  
11    if  $O_m$  is not existed in pop then  
12    |  $O = O_m$ ;  
13   Evaluate  $O$  to get  $f(O)$ ;  
14   if  $f(O) == f(P_1)$  then  
15   | Replace  $P_1$  by  $O$ ;  
16   else if  $f(O) == f(P_2)$  then  
17   | Replace  $P_2$  by  $O$ ;  
18   else if  $f(O) == f(P)$  &&  $P! = P_1$  &&  $P! = P_2$  then  
19   | Discard  $O$ ;  
20   else if  $f(O)$  is not used in current pop then  
21   | Randomly choose a  $P$  from  $[nc/2, nc]$ ;  
22   | Replace  $P$  with  $O$  ;  
23   Resort Population;  
24   if Replacement criteria are met then  
25   | replace  $nrep$  chromosomes with randomly generated chromosomes;
```

---

---

**Algorithm 3:** The pseudo code of Simulation System

---

**Input:** Executable solution  $s$ , Time of change:  $t$ , Previous graph  $G$

```
1 Set Events probability:  $\{p_1, p_2, p_3, p_4, p_5\}$ ;  
2 Set probability for broke down roads recovering:  $p_{bdr}$ ;  
3 Set probability for congest roads recovering and becoming better:  $p_{crr}, p_{crbb}$ ;  
4 Determine the stopping point for each vehicles according to  $s, t, G$ ;  
5 Update graph, and remove all served tasks.  
6 Randomly select  $p_1 \times 100\%$  vehicles to break down (Event 1).  
7 Update the graph.  
8 **** Cost Impact ****/  
9 for each edge  $e_i$  do  
10   if  $e_i.change == 0$  then  
11      $r_2 = rand(), r_3 = rand()$   
12     if  $r_2 < p_2$  and  $r_3 < p_3$  then  
13       Event 2 happens:  $e_i.cost = Inf, e_i.change == 2$  ;  
14     if  $r_2 < p_2$  and  $r_3 > p_3$  then  
15       Event 3 happens: Increase cost of  $e_i, e_i.change == 3$  ;  
16   else if  $e_i.change == 2$  and  $rand() < p_{recover}$  then  
17     Recover edge  $e_i, e_i.change == 0$  ;  
18   else if  $e_i.change == 3$  then  
19     if  $rand() < p_{congestion.recover}$  then  
20       Recover edge  $e_i, e_i.change == 0$  ;  
21     else if  $rand() < p_{congestion.better}$  then  
22       Decrease cost of  $e_i$   
23     else  
24       Increase cost of  $e_i$   
25   else  
26     continue;  
27 **** Demand Impact ****/  
28 for each edge  $e_i$  do  
29   if  $e_i.change = 1$  then  
30     continue;  
31   else  
32      $r_4 = rand(), r_5 = rand()$ ;  
33     if  $Edge.demand > 0$  and  $r_4 < p_4$  then  
34       Event 4 happens:  $e_i.change = 4$ ;  
35     if  $Edge.demand == 0$  and  $r_5 < p_5$  then  
36       Event 5 happens:  $e_i.change = 5$ ;
```

**Output:** The new graph  $G_1$

---

---

**Algorithm 4:** Build auxiliary graph for DCARP

---

**Input:** Individual :  $I = \{t_1, t_2, \dots, t_N\}$

- 1 Stop points for outside vehicles:  $V = \{v_1, v_2, \dots, v_K\}$
- 2 Remain capacity for outside vehicles:  $CP = \{cp_1, cp_2, \dots, cp_K\}$
- 3 Generate  $N + 1$  Nodes (Index from 0 to  $N$ ) for the auxiliary graph  $G^*$ .
- 4 **for** each outside vehicle  $k$  **do**
- 5     **for** each node pair:  $Node_i$  and  $Node_j$  **do**
- 6         Use vehicle  $k$  to serve  $\{t_{i+1}, t_{i+2}, \dots, t_j\}$ ;
- 7         Sub-route:  $r_{ijk} = \{v_k \rightarrow t_{i+1} \rightarrow t_{i+2} \rightarrow \dots \rightarrow t_j \rightarrow depot\}$ ;
- 8         Calculate the total demand  $d_{ijk}$  of  $r_{ijk}$ ;
- 9         **if**  $d_{ijk} > cp_k$  **then**
- 10              $\hookrightarrow$  *continue*;
- 11         **else**
- 12             Calculate the cost of  $r_{ijk}$ :  $c_{ijk}$ ;
- 13             Assign an edge  $e_{ijk}$  between  $Node_i$  and  $Node_j$  with weight equal to  $c_{ijk}$ ;

**Output:** An auxiliary graph  $G^*$

---

---

**Algorithm 5:** A\* based optimal split scheme

---

**Input:** Individual :  $I = \{t_1, t_2, \dots, t_N\}$

- 1 Build an auxiliary graph  $G^*$  for DCARP;
- 2  $expandNode = Node_0$ ;  $openNodeSet = \{\}$ ;  $pathSet = \{\}$ ;
- 3 **while** *True* **do**
- 4     **if**  $expandNode == target$  **then**
- 5         Shortest path  $P$ : path correspond to  $expandNode$ ;
- 6         Minimal cost  $C$ :  $f_{expandNode}$  correspond to  $expandNode$ ;
- 7         **break**;
- 8     Select  $rootPath$  (i.e. path from  $Node_0$  to  $expandNode$ ) from  $pathSet$ ;
- 9     **for each** *successor* of  $expandNode$  **do**
- 10          $newPath = rootPath + expandNode \rightarrow successor$ ;
- 11         Remove all edges correspond to vehicles being used in  $newPath$  for  $successor$ ;
- 12         Calculate the  $h_{succ}$  and  $g_{succ}$ ;
- 13         Set  $f_{succ} = h_{succ} + g_{succ}$ ;
- 14         **if**  $successor == target$  **then**
- 15             Repair  $f_{succ}$ ;
- 16         Add the  $successor$  into  $openNodeSet$ ;
- 17         Add the path correspond to  $successor$  into  $pathSet$ ;
- 18     Remove the  $expandNode$  from  $openNodeSet$ , and the  $rootPath$  from  $pathSet$ ;
- 19     Select the node in  $openNodeSet$  with minimal  $f$  as  $expandNode$ ;
- 20 The shortest path from  $Node_0$  to  $target$  in  $G^*$ :  $P = \{p_1, p_2, \dots, p_M\}$ ;
- 21 Each  $p_m$  represents an edge  $e_{ijk}$ , which denotes a sub-route  $r_{ijk}$ ;
- 22 Obtain the solution  $S$  by splitting the  $I$  by  $P$ .

**Output:** Solution  $S = \{r_1, r_2, \dots, r_M\}$ , Minimal cost:  $C$

---

---

**Algorithm 6:** Greedy split scheme

---

**Input:** Individual :  $I = \{t_1, t_2, \dots, t_N\}$

1 Build an auxiliary graph  $G^*$  for DCARP;

2 **for** each edge  $e_{ijk}$  in  $G^*$  **do**

3     Calculate the UDC:  $UDC_{ijk}$ ;

4  $expandNode = Node_0$ ;  $newPath = Node_0$

5 **while** *True* **do**

6     **if**  $expandNode == target$  **then**

7         Greedy path:  $newPath, P = \{p_1, p_2, \dots, p_M\}$ ;

8         Calculate the greedy cost of greedy path:  $C$ ;

9         **break**;

10      $rootPath \leftarrow newPath$ ;

11     Select the  $Node_X$  with minimal  $UDC$  from all *successors* for  $expandNode$ ;

12      $newPath = rootPath + expandNode \rightarrow Node_X$ ;

13     Remove all edges correspond to vehicles being used in  $newPath$ ;

14      $expandNode \leftarrow Node_X$ ;

15 Each  $p_m$  represents an edge  $e_{ijk}$ , which denotes a sub-route  $r_{ijk}$ ;

16 Obtain the solution  $S$  by splitting the  $I$  by  $P$ .

**Output:** Solution  $S = \{r_1, r_2, \dots, r_m\}$ , Greedy cost:  $C$

---

---

**Algorithm 7:** The hybrid local search framework

---

**Input:** The update Map (update graph data)

Dynamic State:

- 1). Stop locations of outside vehicles;
- 2). Remaining capacities of outside vehicles;
- 3). Remaining tasks.

```
1 Initialize the solution archive  $SA \leftarrow \emptyset$ ;  
2 Re-construct the solution  $S_0$  with explicit routes;  
3 Add initial solution into archive  $SA = SA \cup S_0$ ;  
4 Set global best solution  $S_{gb} = S_0$  for each solution  $S_i$  in  $SA$  do  
5   Local best solution  $S_{lb} = S_i$ ;  
6   while true do  
7     // The following loop (line7-line10) run in parallel  
8     for each neighborhood move  $Move_j$  do  
9       Solution  $S_{mj} = Move_j(S_{lb})$   
10      if improved AND archive is not full then  
11        Add  $S_{mi}$  into archive:  $SA = SA \cup S_{mi}$ ;  
12      Update best solution  $S_{lb}$  from  $S_{mj}$ ;  
13      if No improved move OR exceed time limitation then  
14        break;  
14  if  $S_{lb}.cost < S_{gb}.cost$  then  
15     $S_{gb} \leftarrow S_{lb}$ ;  
16  if exceed time limitation then  
17    break;
```

**Output:** The global best solution  $S_{gb}$

---



---

**Algorithm 8:** Pseudo code of the instance generator.

---

**Input:** Static instance, initial solution;

The full capacity of vehicles:  $Q$ ;

A configuration of dynamic event:  $C_{Event}$ ;

Configurations of state factors:  $C_{OV}$ ,  $C_{RQ}$ ;

**Output:** A DCARP instance

```
1 if  $C_{Event} == ND$  then
2   if  $ND-N == few$  then
3      $p = 20\%$ 
4   else
5      $p = 80\%$ 
6   Uniformly random select  $p$  of tasks in the remaining tasks and save them into a set
      $Set_{ND}$ .
7   if  $ND-V == small$  then
8      $dm = \frac{Q}{|Set_{ND}|}$ 
9   else
10     $dm = \frac{4Q}{|Set_{ND}|}$ 
11   Add demand  $dm$  to each task in  $Set_{ND}$ .
12 if  $C_{Event} == NT$  then
13   Save all available edges (not task) in a list  $List_{NT}$ ;
14   if  $NT-P == close$  then
15     Sort  $List_{NT}$  in ascending order according to the max distance of two nodes to the
       depot.
16   else
17     Sort  $List_{NT}$  in descending order according to the max distance of two nodes to the
       depot.
18   if  $NT-N == few$  then
19      $p = 20\%$ 
20   else
21      $p = 80\%$ 
22   Select the front  $p$  of edges from  $List_{NT}$  as the new tasks and save into  $Set_{NT}$ .
23   if  $NT-V == small$  then
24      $dm = \frac{Q}{|Set_{ND}|}$ 
25   else
26      $dm = \frac{4Q}{|Set_{ND}|}$ 
27   Add demand  $dm$  to each task in  $Set_{NT}$ .
```

---