Supplementary File of "Decomposition-based Evolutionary Deep Reinforcement Learning for Practical Vehicle Routing Problems"

Junchuang Cai, Haojie Chen, Qingling Zhu, Qiuzhen Lin, *Member, IEEE*, Zhong Ming, and Kay Chen Tan, *Fellow, IEEE*

S.A Problem Formulation

The model of the vehicle routing problem with simultaneous pickup and delivery and time windows (VRPSPDT) is illustrated in Fig. S1. Specifically, VRPSPDT can be modeled by a complete graph $G = (V, E)$, where $V = \{0, 1, 2, \ldots, M\}$ includes the depot 0 and M customer nodes, and $E = \{< i, j > | i, j \in V, i \neq j\}$ represents the set of arcs between each pair of nodes. Each node i within V is characterized by five attributes: delivery demand d_i, indicating the quantity of goods transported from the depot to customer i; pickup demand p_i, detailing the quantity of goods retrieved from customer i and returned to the depot; start time a_i of the time window, denoting the earliest time at which customer i can receive pickup and delivery services, with vehicles required to wait if arriving before a_i; end time b_i of the time window, specifying the latest allowable time for customer i to receive pickup and delivery services, beyond which vehicles cannot arrive; and service time s_i, representing the duration needed for vehicles to load and unload goods at customer i. The edges linking nodes correspond to two matrices: a distance matrix illustrating distances between nodes and a time matrix depicting travel times between nodes.

Moreover, VRPSPDTs involve various constraints, as outlined below:

$$arr(h_{i,j}) = dep(h_{i,j-1}) + time(h_{i,j-1}, h_{i,j}), 1 \leq i \leq k, j > 1 \tag{1}$$

$$dep(h_{i,j}) = max\{arr(h_{i,j}), a_{h_{i,j}}\} + s_{h_{i,j}}, 1 \leq i \leq k, j > 1 \tag{2}$$

$$load(h_{i,1}) = \sum_{j=1}^{L_i} d_{h_{i,j}}, 1 \leq i \leq k \tag{3}$$

$$load(h_{i,j}) = load(h_{i,j-1}) - d_{h_{i,j}} + p_{h_{i,j}}, 1 \leq i \leq k, j > 1 \tag{4}$$

$$h_{i,1} = h_{i,L_i} = 0, 1 \leq i \leq K \tag{5}$$

$$\sum_{i=1}^{K} \sum_{j=2}^{L_i-1} I[h_{i,j} == x] = 1, 1 \leq x \leq M \tag{6}$$

$$load(h_{i,j}) \leq Q, 1 \leq i \leq K, 1 \leq j \leq L_i \tag{7}$$

$$a_{h_{i,j}} \leq arr(h_{i,j}) \leq b_{h_{i,j}}, 1 \leq i \leq K, 2 \leq j \leq L_i \tag{8}$$

$$dep(h_{i,1}) \geq a_0 \; and \; arr(h_{i,L_i}) \leq b_0, 1 \leq i \leq K. \tag{9}$$

Specifically, the arrival and departure times at node $h_{i,j}$ are represented by $arr(h_{i,j})$ and $dep(h_{i,j})$ in Eqs. (1)–(2). The vehicle's capacity upon arrival at $h_{i,j}$ is detailed in Eqs. (3)–(4). Constraint (5) ensures that each route begins and ends at the depot. Constraint (6) guarantees that each customer is serviced exactly once. Constraint (7) ensures that vehicles do not exceed their capacity during transportation. Constraint (8) specifies that customers are served within their designated time windows. Finally, constraint (9) mandates that vehicles can depart from the depot no earlier than the start time a_0 and must return before the end time b_0.

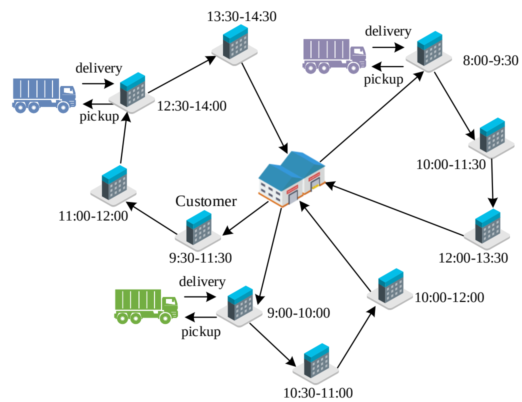

Fig. S1. Model of the VRPSPDT.

S.B Evolutionary Operation

The pseudocode of the initialization is outlined in Algorithm B1. Figure S2 illustrates the process of splitting a permutation sequence into a routing solution. Assume that u_1 and u_2 in Eqs. (2)–(3) are set to 30 and 1, respectively. For example, when considering the route [0, 1, 0], the travel cost is 40, while the vehicle cost is 30, resulting in a total cost of 70. The *split* method computes the total cost for each possible route. Ultimately, the least-cost routing solution obtained is [0, 1, 0, 2, 3, 0, 4, 5, 0], which incurs a total cost of 345.

Algorithm B2 outlines the pseudocode for the population update process.

S.C Deep Reinforcement Learning Operation

In DQNLS, the optimization of VRPSPDT is formulated as a Markov decision process (MDP), with the following specifications:

Algorithm B1 Initialization()

Input: population size: N; The total number of customers: D;
$B \leftarrow$ initialize weight vector;
/* Initialize population $P = \{x^1, ..., x^N\}$ */
for i := 1 to N **do**
  $seq \leftarrow$ randomly generate a permutation sequence of length D;
  $x^i \leftarrow$ split the seq into a routing solution;
end for
$z \leftarrow$ initialize reference point of P;
Output: The initial population $P = \{x^1, ..., x^N\}$.

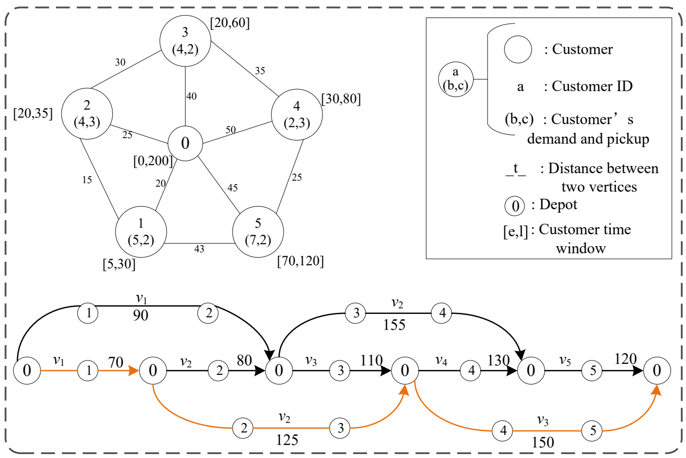

Fig. S2. The splitting process of a solution.

Algorithm B2 Population_Update(P, x_{child}, B)

Input: population: P; child solution: x_{child}; the T nearest weight vectors for each subproblem: B;
Update reference point z;
/* Update population */
$P = rand < \delta?B(i) : 1, \ldots, N$;
Set $c = 0$;
while $c < n_r$ and $P \neq \emptyset$ **do**
  Randomly pickup an index j from P;
  if $g(x_{child}|\lambda^j, z) < g(x^j|\lambda^j, z)$ and x^j is not the best solution **then**
    $x^j \leftarrow x_{child}$, and $c \leftarrow c + 1$;
  end if
  Delete index j from P;
end while

- ***State:*** The state is used to represent the environmental conditions, which can be encoded to capture the relevant information among nodes. Here, the state is encoded by concatenating sequences derived from removing the depot delimiters in all routes of the current solution. The delimiter-free sequence encoding simplifies the state representation and improves the learning efficiency to solve VRPSPDTs, which allows DQNLS to generalize across varying problem sizes while capturing essential patterns in node relationships.
- ***Action:*** The actions encompass a range of local search strategies for the current solution, including exchange, relocate, and cross-exchange, or-opt, and 2-opt*. In DQNLS, the action design integrates the learning capability of DQN with the efficiency of local search strategies to rapidly improve overall performance. Reward feedback guides DQNLS in identifying the most effective strategy at each step while balancing exploration and exploitation to avoid local optima and progressively improve the solutions.
- ***Transition:*** An action is selected based on the current state and applied to the current solution, resulting in a transition to a deterministic next state that enables DQNLS to navigate toward an optimal solution.
- ***Reward:*** Designing an effective reward function is crucial in DRL, as it directly impacts the learning process and the performance of the network. The reward value is defined as the difference between the objective value of the original solution and that of the improved solution after the action is applied. This enables the DQNLS to learn and prioritize actions that yield better solutions.

S.D Benchmark Problems and Performance Metrics

Table A.I presents the specific settings of the large-scale JD dataset. In our DEDRL, both the Q-network and target network share a similar architecture, consisting of two fully connected layers. The input layer, with a size of 1000, is designed to handle the sequence encoding of the maximum number of customers. The output layer contains five units, which correspond to the five predefined actions (that is, five local search strategies). A hidden layer of 128 neurons is positioned between these two layers and uses the rectified linear unit (ReLU) activation function. This architecture is capable of effectively processing the input state information and predicting the associated Q-values for each possible action. The fully connected layers help capture intricate relationships between inputs and outputs. During training, the network learns the appropriate weights and biases, thus approximating the mapping from an input solution to the corresponding action. Additionally, the trained model weights can be stored and reloaded for subsequent tasks, eliminating the need to retrain each time. The detailed parameters of DEDRL are shown in Table A.II.

TABLE A.I
PROPERTIES OF THE JD DATASET

Problems	D	C	J	u_1	u_2
F201-F204	200	2.5	500	300	0.014
F401-F404	400	2.5	500	300	0.014
F601-F604	600	2.5	500	300	0.014
F801-F804	800	2.5	500	300	0.014
F1001-F1004	1000	2.5	500	300	0.014

TABLE A.II
PARAMETER SETTINGS OF DEDRL.

Parameters	Description	Values
Max_Eval	Total evaluations of algorithms	18000
PM	The predefined mutation probability	0.3
NUM_MU	The predefined number of mutations	30
N	Population size	36
ρ_0	Initial perturbation ratio	1.2
$decayRate$	Decay rate of the perturbation	0.91
α	Learning rate	0.001
γ	The discount factor	0.9
n_{input}	The number of input layer nodes	1000
n_{hidden}	The number of hidden layer nodes	128
n_{output}	The number of output layer nodes	5

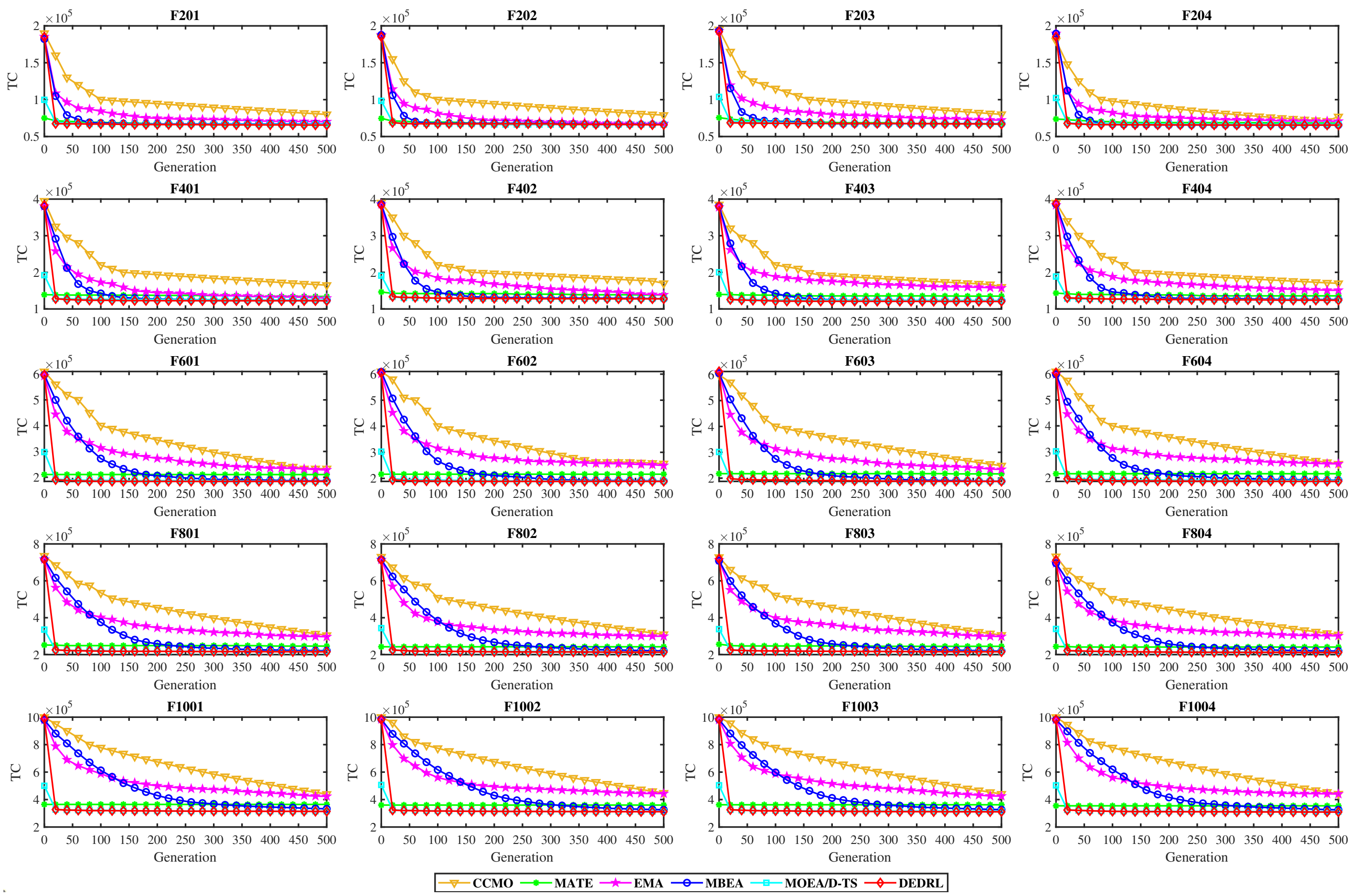

Fig. S3. Averaged search convergence traces of the proposed method and the compared algorithms.