

# Policy Evolution for Routing and Scheduling in the Dynamic Parallel Drone-Vehicle Routing Problem with Time Windows

First Author<sup>1</sup>[0000–1111–2222–3333], Second Author<sup>2,3</sup>[1111–2222–3333–4444], and Third Author<sup>3</sup>[2222–3333–4444–5555]

<sup>1</sup> Princeton University, Princeton NJ 08544, USA

<sup>2</sup> Springer Heidelberg, Tiergartenstr. 17, 69121 Heidelberg, Germany  
[lncs@springer.com](mailto:lncs@springer.com)

<http://www.springer.com/gp/computer-science/lncs>

<sup>3</sup> ABC Institute, Rupert-Karls-University Heidelberg, Heidelberg, Germany  
[{abc,lncs}@uni-heidelberg.de](mailto:{abc,lncs}@uni-heidelberg.de)

**Abstract.** **Keywords:** genetic programming · Dynamic Parallel Drone-Vehicle Routing Problem with Time Windows.

## 1 Methodologies

### 1.1 Dispatching policies

In this paper, the DPDTRPTW is decomposed into two interrelated sub-problems. The first sub-problem focuses on customer selection and vehicle assignment, which decides whether to accept each incoming request and, if accepted, allocates it to the most suitable resource. The second sub-problem concerns service sequencing, which determines the order in which the accepted requests should be handled within their time windows. This decomposition allows the system to dynamically react to incoming requests while optimizing both service quality and system efficiency in a hybrid delivery setting. Specifically, we cast the two sub-problems into two corresponding decision rules: a **routing rule** and a **sequencing rule**. Each routing rule  $R$  and sequencing rule  $S$  is a time-dependent function of vehicle  $k \in \mathcal{D} \cup \mathcal{T}$  and customer  $i \in \mathcal{C}$ .

**Routing rule** When the order of customer  $i$  arrives, the system calculates the priority value  $R(k, i)$  for every vehicle  $k \in \mathcal{D} \cup \mathcal{T}$ , and assigns  $i$  to the vehicle  $k'$  that maximizes this value:

$$k' = \arg \max_{k \in \mathcal{D} \cup \mathcal{T}} R(k, i).$$

After  $k'$  is determined, customer  $i$  is added to the vehicle queue  $Q_{k'}$  of vehicle  $k'$ .

**Sequencing rule** When a vehicle  $k$  needs to decide which node to visit next, it evaluates the priority value  $S(k, i)$  for all customers  $i$  currently in its queue. The next node to visit  $i'$  is selected as the one that maximizes this value, while still satisfying its time window constraint:

$$i' = \arg \max_{i \in \mathcal{V} \setminus \{0\}} S(k, i).$$

If the capacity constraint is violated, then the vehicle  $k$  will return to the depot before serving the selected customer.

**Evaluation Strategy** For the given dynamic problem  $P$ , we transform it into a static problem  $s_T(P)$  by only considering customers  $i$  with creation time  $t_i < T$  for some  $T$ .

Given a dispatching policy  $\pi = (R, S)$ , we run a simulation of  $\pi$  on  $s_T(P)$  and extract its relevant objectives  $f_1$  and  $f_2$ . These values are normalized into  $\bar{f}_1$  and  $\bar{f}_2$ . Then, we combine the two objectives using a weighted sum:

$$f = \lambda \bar{f}_1 + (1 - \lambda) \bar{f}_2$$

$f$  is then used as an estimation of how well  $\pi$  would perform on  $P$ .

**Problem augmentation** Since  $s_T(P)$  is only a small part of the dynamic problem  $P$ , we might perform augmentation as follows:

- For every customer  $i$ , we reduce the time window of  $i$  by a factor of  $sf \in (0, 1)$  (stress factor). The original time window  $W = [s_i, e_i]$  will be transformed into:

$$W' = [s_i, s_i + (1 - sf)(e_i - s_i)].$$

- Additional customers will be generated based on the available customers in  $s_T(P)$ .

After a policy  $\pi^*$  that performs well on  $s_T(P)$  is found, we will use it as the dispatching strategy on the dynamic problem  $P$ .

## 1.2 Genetic programming hyper-heuristic algorithm approach

To tackle these sub-problems, we introduce a genetic programming hyper-heuristic (GPHH) to handle two distinct sets of rules efficiently. The proposed algorithm is divided into two steps: the training phase and the testing phase. During the training phase, the algorithm generates a set of heuristics rules rather than a direct solution for the DPDRPTW problem. These rules would then serve as the input for the testing phase, where their effectiveness is evaluated.

- **Training phase:** In this phase, we work with a population, denoted as  $\mathcal{P} = \{p_1, p_2, \dots, p_{p\_size}\}$ , where each individual in  $\mathcal{P}$  is represented as a multi-tree (see Figure ??). The fitness value of each individual is evaluated

by applying the two rules over a replication  $\mathcal{T}_{train}$ . The population evolves through various genetic programming mechanisms during each generation to enhance diversity and explore potential solutions. Details of our proposed algorithm are described in subsection ??.

- **Testing phase:** The rules derived from the training phase are thoroughly tested on the testing data during this phase. The testing process is similar to a single evaluation in the training phase, but takes place in a larger simulation environment. It is essential to highlight that the datasets used for training and testing are separate, even though they share similar attributes.

GPHH emulates the process of natural evolution to enhance the offspring quality through successive generations. The process involves four main steps: initialization, evaluation, selection, and recombination, as outlined in Algorithm ??.

GPHH begins with a population of randomly initialized individuals. The evaluation step assesses the quality of each individual using DPDTRPTW instance. As long as the stopping criterion is not met (i.e., the current generation number  $t$  is less than the maximum number of generations  $G$ ). In that case, the algorithm proceeds to parent selection, where individuals with higher fitness are chosen to produce new offspring through genetic operators such as reproduction, crossover, and mutation. If the stopping criterion is satisfied, the best scheduling heuristic  $p^*$  found during the process is returned as the solution of the DPDTRPTW problem.

**Algorithm 1** Training phase in GPHH

---

**Input:**

- $pop\_size$ : population size;
- $intermediate\_size$ : offspring population size;
- $r_m$ : mutation rate;
- $r_c$ : crossover rate;
- $G$ : the number of generation;
- $\mathcal{T}_{train}$ : training dataset

**Output:** Best individual  $p^*$

```

1: Initialize a population  $\mathcal{P}^{(0)}$  ;
2: Evaluate every individual in  $\mathcal{P}^{(0)}$ ;
3:  $t \leftarrow 1$ 
4: while  $t \leq G$  do
5:   Offspring population  $\mathcal{O}^{(t)} \leftarrow \emptyset$  ;
6:   while  $|\mathcal{O}^{(t)}| < intermediate\_size$  do
7:      $rand \leftarrow \mathcal{U}(0, 1)$  ;
8:     if  $rand < r_c$  then
9:       Select parents  $pa, pb$  from  $\mathcal{P}^{(t-1)}$ ;
10:       $o \leftarrow \text{crossover}(pa, pb)$  ;
11:      else if  $rand < r_c + r_m$  then
12:        Select parent  $p$  from  $\mathcal{P}^{(t-1)}$ ;
13:         $o \leftarrow \text{mutation}(p)$  ;
14:      else
15:        Select parent  $p$  from  $\mathcal{P}^{(t-1)}$ ;
16:        Reproduce  $o$  from  $p$ ;
17:      end if
18:       $\mathcal{O}^{(t)} \leftarrow \mathcal{O}^{(t)} \cup \{o\}$  ;
19:    end while
20:    Evaluate  $\mathcal{O}$  ;
21:     $\mathcal{P}^{(t)} \leftarrow \mathcal{P}^{(t-1)} \cup \mathcal{O}^{(t)}$  ;
22:     $\mathcal{P}^{(t)} \leftarrow \text{ELITISMSELECTION}(\mathcal{P}^{(t)})$  ;
23:     $t \leftarrow t + 1$  ;
24:  end while
25:   $p^* \leftarrow \text{Best individual in } \mathcal{P}^{(t)}$ 
26: return  $p^*$ 

```

---

**Individual representation** Each individual in GPHH is represented as a tree-based program structure, composed of internal nodes and terminal nodes. These nodes define the structure and behavior of the program, tailored to the specific requirements of the optimization problem. At its core, this representation captures relevant problem attributes at decision points and encodes them into a *priority function* that guides which operation or action should be taken.

By recursively combining attributes through mathematical operators, the tree can represent complex, non-linear decision logic. This allows the model to consider a wide range of information when determining the priority of actions, leading to more informed and effective decision-making. Among the representa-

tions evaluated, the tree-based structure demonstrates the highest effectiveness due to its compactness, interpretability, and expressive power.

In this paper, we employ a **dual-tree representation**, where each individual is modeled as a pair  $(R, S)$ , with  $R$  denoting the *routing rule* and  $S$  the *sequencing rule*. Both rules are encoded as genetic programming (GP) trees. Specifically:

- **Terminal nodes:** problem-specific attributes drawn from the simulation environment, such as current time, current demand, buffer occupancy, or operation duration.
- **Internal nodes:** mathematical operations (e.g.,  $+$ ,  $-$ ,  $\times$ ,  $\div$ , min, max) used to combine terminal values into composite expressions.

This dual-tree structure enables the independent yet coordinated evolution of policies for sequencing and routing, allowing GPHH to handle complex decision-making tasks with improved flexibility and performance. An example of such an individual is illustrated in Figure ??.

**Initialization** We adopt the *ramped half-and-half* initialization technique to construct the initial population. In this approach, half of the individuals are generated using the *full method*, while the remaining half are initialized using the *grow method*. This combination promotes structural diversity in the initial population and helps avoid premature convergence.

- **Full initialization:** All trees are constructed to reach a predefined maximum depth. Each node in the tree is deterministically assigned a function (for internal nodes) or a terminal (for leaves), ensuring that all branches are fully populated. While this approach enables exhaustive exploration of the search space, it often results in large and complex trees, which may increase computational overhead and reduce the efficiency of subsequent genetic operations.
- **Grow initialization:** Trees are generated with variable depths within a specified range. At each node, either a function or a terminal is randomly chosen, which leads to trees of different shapes and sizes. This method contributes to a more heterogeneous initial population by producing individuals with varying levels of complexity.

**Genetic Operators** Genetic operators are a fundamental component of the evolutionary algorithm, as they are responsible for generating new individuals through recombination and variation. In this study, two widely used operators in GPHH are employed: **sub-tree crossover** and **sub-tree mutation**.

*Sub-tree crossover* This operator creates offspring by exchanging substructures between two parent trees. Specifically, a random subtree is selected from each parent and swapped, forming two new individuals. Consider  $depth_{sub-tree\_A}$  and  $depth_{sub-tree\_B}$  denote the depths of the subtrees selected from Parent A and B,

respectively. Let  $depth_{cut\_A}$  and  $depth_{cut\_B}$  represent the depths at which these subtrees are cut. To ensure that the resulting trees do not exceed the maximum allowable depth  $L$ , the following constraints must be satisfied:

$$\begin{aligned} depth_{sub-tree\_A} + depth_{cut\_B} &\leq L, \\ depth_{sub-tree\_B} + depth_{cut\_A} &\leq L \end{aligned}$$

*Sub-tree mutation* Mutation introduces random variation into the population and plays a crucial role in maintaining genetic diversity and avoiding premature convergence to local optima. As illustrated in Figure ??, for each individual selected for mutation, a random node (mutation point) is chosen within its tree. The subtree rooted at this node is replaced with a newly generated random tree. To maintain the maximum allowed tree depth  $L$ , the depth of the inserted tree is constrained to  $L - depth_{sub-tree_p}$ , where  $depth_{sub-tree_p}$  is the depth of the tree at the mutation point. This constraint ensures that the resulting tree does not exceed the maximum allowable depth.

**Elitism selection** This paper employs elitism selection, where the offspring are combined with the parent population, and the highest-performing individuals are chosen to advance to the next generation. This strategy preserves the best solutions, ensuring their genetic traits contribute to future breeding. As a result, the population retains high-quality characteristics, promoting faster convergence toward an optimal solution.

## References