



BÁO CÁO BÀI TẬP LỚN  
IT3940: PROJECT III

**Parallel VRPD-time window**  
**(Thuật toán Genetic Programming)**

Giảng viên hướng dẫn: TS. Nguyễn Khánh Phương

Sinh viên thực hiện: Phạm Thanh An  
Mã số sinh viên: 20224911

Hà Nội, tháng 2 năm 2026

# Mục lục

<b>1</b>	<b>Giới thiệu</b>	<b>2</b>
1.1	Đặt vấn đề . . . . .	2
1.2	Phạm vi đề tài . . . . .	2
<b>2</b>	<b>Mô tả bài toán</b>	<b>3</b>
2.1	Các thành phần của hệ thống . . . . .	3
2.2	Đặc điểm của Request . . . . .	3
2.3	Ký hiệu và tham số . . . . .	3
2.4	Ràng buộc . . . . .	4
<b>3</b>	<b>Mô hình hóa giải pháp với GPHH</b>	<b>6</b>
3.1	Nút lá cho biểu diễn cây GP . . . . .	6
3.1.1	Routing Rule (Quy tắc định tuyến – cây R) . . . . .	6
3.1.2	Sequencing Rule (Quy tắc sắp xếp – cây S) . . . . .	6
3.2	Nút trong cho biểu diễn cây GP . . . . .	7
<b>4</b>	<b>Sơ đồ thuật toán và quy trình thực hiện</b>	<b>8</b>
4.1	Các thuật toán tiến hóa . . . . .	8
4.2	Các thuật toán phân bổ sự kiện và điều phối xe . . . . .	14
<b>5</b>	<b>Thực nghiệm và kết quả</b>	<b>24</b>
5.1	Tham số thực nghiệm Genetic Programming . . . . .	24
5.2	Kết quả thực nghiệm trên các instance . . . . .	24
<b>6</b>	<b>Kết luận</b>	<b>27</b>

# Chương 1

## Giới thiệu

### 1.1 Đặt vấn đề

Trong thời đại 4.0 hiện nay, sự dịch chuyển của ngành logistic trong kỷ nguyên số không chỉ dừng lại ở việc tối ưu lộ trình mà còn là khả năng thích ứng thời gian thực với các biến động của thị trường. Bài toán vận tải truyền thống vốn dựa trên các tài nguyên tĩnh đã không còn đáp ứng được tính phức tạp khi xuất hiện các nhu cầu giao hàng động (Dynamic). Trong bối cảnh đó, mô hình phối hợp song song giữa xe tải và thiết bị bay không người lái (Parallel Truck-Drone) nổi lên như một giải pháp tối ưu, tận dụng được thế mạnh về tải trọng của xe tải và tính linh hoạt về tốc độ của drone.

Tuy nhiên, thách thức lớn nhất nằm ở việc điều phối các thực thể này trong môi trường không chắc chắn, nơi các yêu cầu khách hàng xuất hiện rời rạc theo khung thời gian mà không có thông tin báo trước. Các thuật toán tối ưu hóa cổ điển thường tập trung tìm kiếm một lời giải duy nhất cho một kịch bản cố định, điều này dẫn đến sự thiếu hiệu quả khi môi trường thay đổi.

Để giải quyết vấn đề này, đề tài tiếp cận theo hướng **Genetic Programming Hyper-heuristic**. Trong cách tiếp cận này, thay vì tìm kiếm một lộ trình cụ thể, mục tiêu là tiến hóa các quy tắc điều phối (dispatching policies) để cuối cùng tìm ra tập hợp các chính quy tắc tối ưu không bị trị, đảm bảo sự cân bằng giữa khả năng phục vụ khách hàng tối đa và việc tối thiểu hóa tổng thời gian hoàn thành nhiệm vụ.

### 1.2 Phạm vi đề tài

Mục tiêu của bài toán là xác định lộ trình cho xe tải và drone sao cho tối thiểu hóa thời gian hoàn thành nhiệm vụ (Minimizing Makespan) cùng với đó tối đa lượng khách hàng phục vụ (Maximizing Served) với điều kiện môi trường động, các request xuất hiện ở thời điểm không xác định.

Để giải quyết bài toán phức tạp này, sinh viên đề xuất sử dụng khung giải thuật Genetic Programming. Đây là một phương pháp tìm kiếm metaheuristic mạnh mẽ, một meta-heuristic thuộc nhóm Evolutionary Algorithms, sử dụng cơ chế tiến hoá để tìm kiếm không gian chương trình mà không phụ thuộc vào bài toán cụ thể, phù hợp cho bài toán động bởi giải thuật cung cấp chiến lược ứng phó với sự xuất hiện của yêu cầu mới. Các công việc cần làm bao gồm:

- Tìm hiểu giải thuật Genetic Programming
- Đề xuất chiến lược giải quyết bài toán động (dynamic)
- Giải quyết vấn đề tối ưu đa mục tiêu
- Đánh giá kết quả trên tập dữ liệu có sẵn

# Chương 2

## Mô tả bài toán

### 2.1 Các thành phần của hệ thống

Hệ thống giao hàng bao gồm:

- Depot (kho) điểm xuất phát của tất cả phương tiện
- Phương tiện:
  - Tập hợp  $K$  truck đồng nhất với tải trọng  $M_T$
  - Tập hợp  $D$  drone đồng nhất với tải trọng  $M_D$  và bán kính bay tối đa  $L_D$
- Khách hàng (Requests): Tập khách hàng  $C$  được chia làm 2 loại:
  - $C_1$ : Khách hàng bắt buộc phục vụ bởi xe tải (do kích thước hàng lớn hoặc quá xa tầm bay drone)
  - $C_2$ : Khách hàng có thể phục vụ bởi cả drone hoặc xe tải

### 2.2 Đặc điểm của Request

Mỗi khách hàng  $i$  có các thuộc tính sau:

- Thời điểm xuất hiện  $r_i$  (dynamic)
- Khung thời gian phục vụ  $[e_i, l_i]$  (time window)
- Khối lượng gói hàng  $d_i$
- Thời gian đợi tối đa  $L_w$

### 2.3 Ký hiệu và tham số

a, Tập hợp

- $C = \{1, 2, \dots, n\}$ : Tập hợp các khách hàng (request)
- $C_1 \subset C$ : Tập khách hàng bắt buộc phải phục vụ bằng xe tải
- $C_2 \subset C$ : Tập khách hàng có thể phục vụ bằng xe tải hoặc drone
- $N = C \cup \{0\}$ : Tập các điểm, trong đó 0 là Depot.
- $K$ : Tập hợp tất cả các phương tiện

- $K_T$ : Tập hợp Truck
  - $K_D$ : Tập hợp Drone
  - $\mathcal{R}_k$ : Tập các chuyến đi của phương tiện  $k$
  - $pickup_k^r \subseteq C$ : Tập các đơn hàng mà phương tiện  $k$  đang mang theo trong chuyến đi  $r$
  - $served$ : Tập các đơn hàng được xử lý thành công (về kho)
- b, Tham số
- $d_i$ : Nhu cầu hàng hóa (demand) của khách hàng  $i$
  - $[e_i, l_i]$ : Khung thời gian phục vụ (Time Window) của khách hàng  $i$ 
    - $e_i$ : Thời điểm sớm nhất có thể phục vụ
    - $l_i$ : Thời điểm muộn nhất phải phục vụ
  - $t_{ij}^k$ : Thời gian di chuyển từ  $i$  đến  $j$  của phương tiện  $k$
  - $M_T$ : Tải trọng tối đa của xe tải
  - $M_D$ : Tải trọng tối đa của drone
  - $L_D$ : Phạm vi bay tối đa của drone cho một chuyến (Trip)
  - $L_w$ : Thời gian chờ đợi tối đa của khách hàng (từ khi đơn của họ được lấy cho đến khi phương tiện về kho)
- c, Biến quyết định
- $a_i^k \in R^+$ : Thời điểm phương tiện  $k$  đến phục vụ khách hàng  $i$

## 2.4 Ràng buộc

- Ràng buộc về sức chứa: Tổng khối lượng hàng hóa đang mang theo trong chuyến đi không được vượt quá tải trọng tối đa của phương tiện tương ứng.

$$\sum_{i \in pickup_k^r} d_i \leq \begin{cases} M_T & \text{nếu } k \in K_T \\ M_D & \text{nếu } k \in K_D \end{cases}, \quad \forall r \in \mathcal{R}_k, \forall k \in K$$

- Ràng buộc về khung thời gian:

- Phương tiện  $k$  phải đến khách hàng  $i$  trước thời gian đóng cửa  $l_i$ . Nếu đến sớm hơn  $e_i$ , phương tiện phải chờ:

$$a_i^k \leq l_i, \forall i \in C, \forall k \in K$$

- Mọi quan hệ giữa hai điểm liên tiếp  $i$  và  $j$  trên hành trình: Giả sử  $j$  theo sau  $i$  trong chuyến  $r$  của  $k$ , thì :

$$a_j^k = \max(a_i^k, e_i) + t_{ij}^k$$

- Ràng buộc về bán kính bay Drone:

Giả sử chuyến bay  $r$  của drone  $k$  bao gồm tập các cung đường  $E_r = \{(0, i_1), (i_1, i_2), \dots, (i_m, 0)\}$  thì:

$$\sum_{(u,v) \in E_r} t_{uv}^k \leq L_D$$

- Ràng buộc về tính tương thích: Khách hàng thuộc nhóm  $C_1$  không được phục vụ bởi Drone.

$$pickup_k^r \cap C_1 = \emptyset, \forall k \in K_D, \forall r \in \mathcal{R}_k$$

- Ràng buộc về thời gian chờ đợi: Đối với mỗi khách hàng  $i \notin served$ , thời gian từ khi đơn  $i$  được đưa vào  $pickup_k^r$  (bắt đầu chờ) đến khi phương tiện  $k$  hoàn thành chuyến  $r$  (về depot) không vượt quá  $L_w$ .

$$t_{\text{depot}}^k - t_{\text{pickup}_i}^k \leq L_w, \quad \forall i \notin served, \forall k \in K, \forall r \in \mathcal{R}_k \text{ với } i \in pickup_k^r$$

# Chương 3

## Mô hình hóa giải pháp với GPHH

### 3.1 Nút lá cho biểu diễn cây GP

Thay vì giải bài toán tối ưu tĩnh, em sử dụng GPHH để tìm ra các hàm ưu tiên (Priority Functions). Hệ thống được phân rã thành hai bài toán con quyết định: Định tuyến (Routing) và Sắp xếp (Sequencing)

#### 3.1.1 Routing Rule (Quy tắc định tuyến – cây R)

Quy tắc này quyết định phương tiện nào sẽ được giao nhiệm vụ phục vụ một yêu cầu mới khi yêu cầu xuất hiện.

Khi một yêu cầu  $i$  xuất hiện tại thời điểm hiện tại, hệ thống tính giá trị ưu tiên  $R(k, i)$  cho từng phương tiện  $k$  khả dụng. Yêu cầu  $i$  sẽ được gán cho phương tiện  $k^*$  có giá trị  $R(k^*, i)$  lớn nhất (ưu tiên cao nhất).

Tập terminal (các biến đầu vào) cho cây Routing:

- **RT0:** Tỷ lệ số lượng yêu cầu đang chờ trong hàng đợi của phương tiện (ít việc hơn được ưu tiên).
- **RT1:** Tỷ lệ sức chứa còn lại của phương tiện (còn nhiều chỗ trống được ưu tiên).
- **RT2:** Thời gian di chuyển từ vị trí trung tâm (median) của các yêu cầu đang chờ đến yêu cầu mới (càng gần càng tốt).
- **RT3:** Thời gian di chuyển từ vị trí hiện tại của phương tiện đến yêu cầu mới (càng gần càng tốt).
- **RT4:** Tỷ lệ kích thước gói hàng (demand) của yêu cầu mới so với tổng nhu cầu đang chờ (ưu tiên gói hàng lớn).
- **RT5:** Ưu tiên loại phương tiện (1.0 nếu là Drone, 0.0 nếu là Truck).

#### 3.1.2 Sequencing Rule (Quy tắc sắp xếp – cây S)

Quy tắc này quyết định phương tiện đang rảnh sẽ phục vụ yêu cầu nào tiếp theo trong hàng đợi của chính nó.

Khi phương tiện  $k$  rảnh (sẵn sàng nhận nhiệm vụ mới), hệ thống tính giá trị ưu tiên  $S(k, i)$  cho từng yêu cầu  $i$  đang chờ trong hàng đợi của  $k$ . Yêu cầu có giá trị  $S(k, i)$  **nhỏ nhất** và thỏa mãn các ràng buộc khả thi sẽ được chọn để phục vụ tiếp theo.

Tập terminal cho cây Sequencing:

- **ST0:** Thời gian di chuyển từ vị trí hiện tại đến yêu cầu (ngắn nhất được ưu tiên).

- **ST1:** Thời gian khách hàng đã chờ kể từ khi yêu cầu xuất hiện (ưu tiên khách chờ lâu).
- **ST2:** Độ khẩn cấp – thời gian còn lại đến thời điểm đóng cửa sổ (Time until close) (ưu tiên yêu cầu sắp hết hạn).
- **ST3:** Kích thước gói hàng (demand) – càng lớn càng được ưu tiên.
- **ST4:** Độ trễ so với thời gian mở cửa sổ (thời gian sớm nhất có thể phục vụ được ưu tiên trước).
- **ST5:** Thời điểm yêu cầu xuất hiện (ưu tiên yêu cầu đến trước – FIFO cơ bản).

## 3.2 Nút trong cho biểu diễn cây GP

Các nút không phải lá (internal nodes) trong cây biểu diễn chương trình được xây dựng từ tập hàm sau:

**add (+), sub (-), mul (\*), protected division (div – chia bảo toàn), min, max.**

Lý do lựa chọn tập hàm này:

- **Các phép toán số học cơ bản (+, ,  $\times$ ,  $\div$  bảo toàn):** Đây là tập hợp tối thiểu cần thiết để biểu diễn hầu hết các hàm ưu tiên thực tế trong bài toán dynamic truck-drone. Chúng cho phép tạo ra các biểu thức tuyến tính và phi tuyến đơn giản như tổng trọng số, chênh lệch thời gian, tỷ lệ khoảng cách/thời gian, v.v. Protected division được sử dụng để tránh lỗi chia cho 0 (trả về giá trị mặc định an toàn, thường là 1 hoặc giá trị tử số).
- **min và max:** Đây là hai hàm phi tuyến quan trọng nhất trong bài toán có ràng buộc thời gian và thứ tự ưu tiên. Chúng giúp mô hình hóa các quyết định dạng “ngưỡng” và “chọn cái tốt nhất/tồi tệ nhất” một cách tự nhiên.
- **Lý do ưu tiên tập hàm nhỏ gọn và phi tuyến đơn giản:**
  - Phù hợp với đặc trưng của bài toán **quyết định thời gian thực**: Các chính sách dispatching trong môi trường dynamic VRPDTW thường dựa trên so sánh (min/max) và kết hợp tuyến tính/tỷ lệ hơn là các hàm phức tạp như sin, exp, log.
  - Dễ diễn giải và phân tích: Tập hàm  $\{+, , \times, \div, \text{min}, \text{max}\}$  sinh ra các biểu thức có thể dễ dàng chuyển thành các quy tắc if-then-else dễ hiểu cho con người – một đặc tính rất quan trọng trong hyper-heuristic.

# Chương 4

## Sơ đồ thuật toán và quy trình thực hiện

Thuật toán sử dụng là NSGA-II (Non-dominated Sorting Genetic Algorithm II) để tối ưu đa mục tiêu, tích hợp với Genetic Programming (GP) để tiến hóa cây quy tắc (routing rule, sequencing rule)

### 4.1 Các thuật toán tiến hóa

---

#### Algorithm 1 Thuật toán tiến hóa

---

```
1: Population ← CreateGreedyInitialPopulation(PopSize, MaxDepth, Problem)
2: Evaluate(Population, Problem)
3: Population ← ElitistIndividual                      ▷ Giữ lại 10% cá thể tốt từ quần thể cha
4: while gen < MaxGens do
5:   Offspring ← []
6:   while len(Population) < PopSize do
7:     Parent1 ← NSGAIIBinaryTournamentSelection(Population, TourSize)
8:     Parent2 ← NSGAIIBinaryTournamentSelection(Population, TourSize)
9:     if random() < c_rate then
10:      Child1, Child2 ← SubtreeCrossover(Parent1, Parent2, MaxDepth)
11:    else
12:      Child1, Child2 ← Copy(Parent1), Copy(Parent2)
13:    end if
14:    if random() < m_rate then
15:      Child1 ← SubtreeMutation(Child1, MaxDepth)
16:    end if
17:    if random() < m_rate then
18:      Child2 ← SubtreeMutation(Child2, MaxDepth)
19:    end if
20:    Offspring ← Offspring ∪ {Child1, Child2}
21:
22:   Evaluate(Offspring, Problem)
23:   Combined ← Population ∪ Offspring
24:   Population ← NSGAIISurvivalSelection(Combined, PopSize)
25: end while
26: return Population
```

---

---

**Algorithm 2** Thuật toán khởi tạo quần thể tham lam

---

**Require:** Kích thước quần thể  $P$ , độ sâu tối đa  $D$

**Ensure:** Quần thể ban đầu  $\mathcal{P}$

```
1:  $\mathcal{P} \leftarrow \emptyset$ 
2:  $P_g \leftarrow \lfloor P/3 \rfloor$ 
3:  $P_w \leftarrow \lfloor 2P/3 \rfloor$                                 ▷ Giai đoạn 1: Cá thể mạnh được thiết kế thủ công
4: for mỗi cặp heuristic  $(R_i, S_i)$  trong danh sách heuristic mạnh do
5:   if  $|\mathcal{P}| \geq P_g$  then
6:     break
7:   end if
8:    $T_R \leftarrow \text{BuildTreeFromString}(R_i)$ 
9:    $T_S \leftarrow \text{BuildTreeFromString}(S_i)$ 
10:  Thêm cá thể  $(T_R, T_S)$  vào  $\mathcal{P}$ 
11: end for                                              ▷ Giai đoạn 2: Cá thể ngẫu nhiên có trọng số
12: while  $|\mathcal{P}| < P_w$  do
13:    $T_R \leftarrow \text{GenerateWeightedRandomTree}(D, R)$ 
14:    $T_S \leftarrow \text{GenerateWeightedRandomTree}(D, S)$ 
15:   Thêm cá thể  $(T_R, T_S)$  vào  $\mathcal{P}$ 
16: end while                                         ▷ Giai đoạn 3: Cá thể ngẫu nhiên hoàn toàn
17: while  $|\mathcal{P}| < P$  do
18:   Chọn ngẫu nhiên  $grow \in \{\text{true}, \text{false}\}$ 
19:    $T_R \leftarrow \text{GenerateRandomTree}(D, grow, R)$ 
20:    $T_S \leftarrow \text{GenerateRandomTree}(D, grow, S)$ 
21:   Thêm cá thể  $(T_R, T_S)$  vào  $\mathcal{P}$ 
22: end while
23: return  $\mathcal{P} = 0$ 
```

---

---

**Algorithm 3** Thuật toán lai ghép cây con (Subtree Crossover)

---

**Require:** Hai cá thể cha mẹ  $P_1, P_2$ , độ sâu tối đa  $D_{\max}$

**Ensure:** Hai cá thể con  $C_1, C_2$

```
1:  $C_1 \leftarrow \text{Copy}(P_1)$ 
2:  $C_2 \leftarrow \text{Copy}(P_2)$ 
   ▷ Chọn ngẫu nhiên cây Routing hoặc Sequencing để lai
3: if random() < 0.5 then
4:    $T_1 \leftarrow C_1.R\_tree$ ,  $T_2 \leftarrow C_2.R\_tree$ 
5:    $type \leftarrow R$ 
6: else
7:    $T_1 \leftarrow C_1.S\_tree$ ,  $T_2 \leftarrow C_2.S\_tree$ 
8:    $type \leftarrow S$ 
9: end if
10:  $n_1 \leftarrow$  số node của  $T_1$ 
11:  $n_2 \leftarrow$  số node của  $T_2$ 
12: for  $k = 1$  to 10 do
13:    $i \leftarrow \text{RandomInteger}(0, n_1 - 1)$ 
14:    $j \leftarrow \text{RandomInteger}(0, n_2 - 1)$ 
15:    $sub_1 \leftarrow$  subtree tại chỉ số  $i$  của  $T_1$ 
16:    $sub_2 \leftarrow$  subtree tại chỉ số  $j$  của  $T_2$ 
17:   if depth( $T_1$ ) – depth( $sub_1$ ) + depth( $sub_2$ ) ≤  $D_{\max}$ 
18:   and
19:   depth( $T_2$ ) – depth( $sub_2$ ) + depth( $sub_1$ ) ≤  $D_{\max}$  then
20:     if  $type = R$  then
21:       Thay subtree tại  $i$  của  $C_1.R\_tree$  bằng  $sub_2$ 
22:       Thay subtree tại  $j$  của  $C_2.R\_tree$  bằng  $sub_1$ 
23:     else
24:       Thay subtree tại  $i$  của  $C_1.S\_tree$  bằng  $sub_2$ 
25:       Thay subtree tại  $j$  của  $C_2.S\_tree$  bằng  $sub_1$ 
26:     end if
27:     break
28:   end if
29: end for
30: return  $C_1, C_2$ 
```

---

---

**Algorithm 4** Đột biến cây con (Subtree Mutation)

---

**Require:** Cá thể  $Ind$ , độ sâu tối đa  $MaxDepth$

**Ensure:** Cá thể mới  $Ind'$

```
1:  $Ind' \leftarrow \text{Copy}(Ind)$ 
2: Chọn ngẫu nhiên cây mục tiêu  $T \in \{r\_tree, s\_tree\}$ 
3:  $size \leftarrow$  số lượng nút của cây  $T$ 
4: for  $i = 1$  to  $10$  do
5:    $idx \leftarrow$  chỉ số nút ngẫu nhiên trong  $[0, size - 1]$ 
6:    $Sub \leftarrow$  cây ngẫu nhiên với độ sâu từ 1 đến 3
7:    $T' \leftarrow$  thay thế cây con tại vị trí  $idx$  của  $T$  bằng  $Sub$ 
8:   if độ sâu của  $T' \leq MaxDepth$  then
9:     cập nhật  $T$  trong  $Ind'$ 
10:    return  $Ind'$ 
11:   end if
12: end for
13: return  $Ind'$                                  $\triangleright$  không thay đổi nếu đột biến thất bại
```

---

---

**Algorithm 5** Đột biến điểm (Point Mutation)

---

**Require:** Cây  $T$

**Ensure:** Cây mới  $T'$

```
1:  $T' \leftarrow \text{Copy}(T)$ 
2:  $size \leftarrow$  số lượng nút của  $T'$ 
3:  $idx \leftarrow$  chỉ số nút ngẫu nhiên trong  $[0, size - 1]$ 
4:  $node \leftarrow$  nút tại vị trí  $idx$ 
5: if  $node$  là InternalNode then
6:   Chọn toán tử mới khác toán tử hiện tại
7:   Tạo nút mới với toán tử mới và giữ nguyên 2 con
8:   Thay thế tại vị trí  $idx$ 
9: else if  $node$  là TerminalNode then
10:   Chọn chỉ số terminal mới khác chỉ số hiện tại
11:   Tạo terminal mới
12:   Thay thế tại vị trí  $idx$ 
13: end if
14: return  $T'$ 
```

---

---

**Algorithm 6** Đột biến nâng (Hoist Mutation)

---

**Require:** Cây  $T$   
**Ensure:** Cây mới  $T'$

```
1:  $size \leftarrow$  số lượng nút của  $T$ 
2: if  $size < 2$  then
3:   return  $T$ 
4: end if
5:  $idx \leftarrow$  chỉ số nút ngẫu nhiên trong  $[0, size - 1]$ 
6:  $Sub \leftarrow$  cây con tại vị trí  $idx$ 
7:  $T' \leftarrow \text{Copy}(Sub)$ 
8: return  $T'$ 
```

---

---

**Algorithm 7** Đột biến hoán vị (Permutation Mutation)

---

**Require:** Cây  $T$   
**Ensure:** Cây mới  $T'$

```
1:  $T' \leftarrow \text{Copy}(T)$ 
2:  $size \leftarrow$  số lượng nút của  $T'$ 
3: for  $i = 1$  to  $5$  do
4:    $idx \leftarrow$  chỉ số ngẫu nhiên trong  $[0, size - 1]$ 
5:    $node \leftarrow$  nút tại vị trí  $idx$ 
6:   if  $node$  là InternalNode then
7:     Hoán đổi con trái và con phải
8:   return  $T'$ 
9:   end if
10: end for
11: return  $T'$                                  $\triangleright$  không thay đổi nếu thất bại
```

---

---

**Algorithm 8** Áp dụng đột biến tổng hợp

---

**Require:** Cá thể  $Ind$ , độ sâu tối đa  $MaxDepth$

**Ensure:** Cá thể mới  $Ind'$

```
1:  $Ind' \leftarrow \text{Copy}(Ind)$ 
2: Chọn ngẫu nhiên cây mục tiêu  $T \in \{r\_tree, s\_tree\}$ 
3:  $r \leftarrow$  số ngẫu nhiên trong  $[0, 1]$ 
4: if  $r < 0.6$  then
5:   Thực hiện Subtree Mutation
6: else if  $r < 0.8$  then
7:   Thực hiện Point Mutation
8: else if  $r < 0.9$  then
9:   Thực hiện Hoist Mutation
10: else
11:   Thực hiện Permutation Mutation
12: end if
13: if đột biến thành công và  $depth(T') \leq MaxDepth$  then
14:   Cập nhật lại cây trong  $Ind'$ 
15: end if
16: return  $Ind'$ 
```

---

## 4.2 Các thuật toán phân bổ sự kiện và điều phối xe

---

**Algorithm 9** Giả lập sự kiện rời rạc

---

**Require:**  $P$ ,  $Ind$

**Ensure:**  $(f_1, f_2)$

```
1:  $P' \leftarrow DeepCopy(P)$ 
2:  $\text{ResetVehicleState}(P')$ 
3:  $EventQueue \leftarrow \emptyset$ 
4:  $PendingRequests \leftarrow \emptyset$ 
5:  $curTime \leftarrow 0$ 
6: for  $r \in P.requests$  do
7:    $EventQueue.push((r.release\_time, ARRIVE, r.id))$ 
8: end for
9:  $EventQueue.push((P.depotClose + \epsilon, END, ))$ 
10: while  $EventQueue \neq \emptyset$  do
11:    $(t, type, payload) \leftarrow EventQueue.pop\_min()$ 
12:    $curTime \leftarrow t$ 
13:   if  $type = END$  then
14:     break
15:   end if
16:   if  $type = ARRIVE$  then
17:      $req \leftarrow DeepCopy(Request[payload])$ 
18:      $P'.requests.append(req)$ 
19:      $ok \leftarrow TryAssign(req)$ 
20:     if  $ok = False$  then
21:        $PendingRequests.append(req)$ 
22:        $WakeUpTrigger(req)$ 
23:     end if
24:   end if
25:   if  $type = VEH\_FREE$  then
26:      $veh \leftarrow P'.vehicles[payload]$ 
27:      $RetryPendingRequests()$ 
28:      $DispatchVehicle(veh)$ 
29:   end if
30: end while
31:  $served \leftarrow CountServed(P')$ 
32:  $total \leftarrow |P.requests|$ 
33:  $makespan \leftarrow \max(v.busy\_until)$ 
34:  $f_1 \leftarrow served/total$ 
35:  $f_2 \leftarrow \max(0, 1 - makespan/P.depotClose)$ 
36: return  $(f_1, f_2)$ 
```

---

**Giải thích thuật toán Giả lập sự kiện rời rạc**

Thuật toán này mô phỏng toàn bộ hệ thống bằng cơ chế mô phỏng sự kiện rời rạc (Discrete Event Simulation). Thay vì tăng thời gian theo từng bước nhỏ, hệ thống chỉ cập nhật trạng thái tại các thời điểm xảy ra sự kiện.

Ban đầu, tất cả các request được chuyển thành sự kiện *ARRIVE* và được đưa vào hàng đợi sự kiện *EventQueue* theo thời điểm *release\_time*. Một sự kiện *END* được thêm vào để đảm bảo mô phỏng kết thúc tại thời điểm đóng depot.

Trong mỗi vòng lặp:

- Sự kiện có thời gian nhỏ nhất được lấy ra khỏi *EventQueue*.
- *curTime* được cập nhật bằng thời điểm của sự kiện.
- Nếu sự kiện là *ARRIVE*, request mới được tạo và thử gán cho xe bằng thuật toán TryAssign.
- Nếu gán không thành công, request được đưa vào *PendingRequests*.
- Nếu sự kiện là *VEH\_FREE*, xe tương ứng sẽ thử gán lại các request đang chờ và thực hiện điều phối tiếp theo.

Sau khi vòng lặp kết thúc, hai giá trị mục tiêu được tính:

$$f_1 = \frac{\text{số request được phục vụ}}{\text{tổng số request}}$$

$$f_2 = \max \left( 0, 1 - \frac{\text{makespan}}{\text{depotClose}} \right)$$

Trong đó *makespan* là thời điểm muộn nhất mà một xe hoàn tất hoạt động.

---

**Algorithm 10** TryAssign request với Rank Tier

---

**Require:**  $r$

**Ensure:**  $True/False$

```
1:  $Candidates \leftarrow ComputeCombinedCandidates(r)$ 
2: if  $Candidates = \emptyset$  then
3:     return False
4: end if
5:  $assigned \leftarrow False$ 
6:  $ReassignQueue \leftarrow \emptyset$ 
7: for  $i = 0 \rightarrow |Candidates| - 1$  do
8:      $(combined, veh) \leftarrow Candidates[i]$ 
9:      $rank(v, r) \leftarrow i$ 
10:    if  $arrival\_time(veh, r) > r.time\_window^{end}$  then
11:        continue
12:    end if
13:     $(ok, removed) \leftarrow AttemptAssignToVehicle(r, veh, combined)$ 
14:    if  $ok$  then
15:         $assigned \leftarrow True$ 
16:        for  $rr \in removed$  do
17:             $ReassignQueue.push((rr, veh))$ 
18:        end for
19:        break
20:    end if
21: end for
22: while  $ReassignQueue \neq \emptyset$  do
23:      $(rr, removedVeh) \leftarrow ReassignQueue.pop()$ 
24:      $CandList \leftarrow ComputeCombinedCandidates(rr)$ 
25:      $k \leftarrow rank(removedVeh, rr)$ 
26:     for  $j = k + 1 \rightarrow |CandList| - 1$  do
27:          $(score, v') \leftarrow CandList[j]$ 
28:          $(ok, removed2) \leftarrow AttemptAssignToVehicle(rr, v', score)$ 
29:         if  $ok$  then
30:             for  $x \in removed2$  do
31:                  $ReassignQueue.push((x, v'))$ 
32:             end for
33:             break
34:         end if
35:     end for
36:     if  $rr$  chưa được gán then
37:          $PendingRequests.append(rr)$ 
38:     end if
39: end while
40: return  $assigned$ 
```

---

## Giải thích thuật toán TryAssign với Rank Tier

Thuật toán TryAssign thực hiện việc gán một request mới  $r$  vào hệ thống dựa trên cơ chế **Rank Tier động**.

**1. Xây dựng danh sách ứng viên và xác định Rank** Với mỗi request  $r$ , hệ thống xây dựng danh sách ứng viên:

$$Candidates(r) = \text{sort\_desc}(\text{combined}(v, r))$$

trong đó:

$$\text{combined}(v, r) = \alpha r_{norm} + \beta arr_{norm}$$

Danh sách được sắp xếp giảm dần theo giá trị *combined*.

Rank của vehicle  $v$  đối với request  $r$  được định nghĩa là:

$$rank(v, r) = \text{index của } v \text{ trong } Candidates(r)$$

Do đó:

- $rank(v, r) = 0$  nghĩa là  $v$  là ứng viên tốt nhất của  $r$ .
- $rank(v, r) = 1$  là lựa chọn thứ hai.
- Rank tăng dần tương ứng với mức độ phù hợp giảm dần.

Rank được tính động mỗi khi cần và có thể thay đổi theo thời gian do phụ thuộc vào trạng thái của vehicle và thời điểm hiện tại *curTime*.

**2. Gán request theo thứ tự Rank** Thuật toán duyệt các vehicle theo thứ tự rank tăng dần (tức là theo thứ tự giảm dần của *combined*).

Với mỗi vehicle:

- Kiểm tra điều kiện time window.
- Thực hiện thử gán thông qua thủ tục `AttemptAssignToVehicle`.

Nếu gán thành công, quá trình dừng lại.

**3. Cơ chế bảo vệ Rank 0** Khi xét thay thế trên một vehicle  $v$ , hệ thống chỉ xem xét các request  $q$  đang nằm trong hàng đợi của  $v$  mà:

$$rank(v, q) = 0$$

Tức là  $v$  phải là lựa chọn tốt nhất của request đó.

Tập các request này được ký hiệu là:

$$TopRequests(v) = \{ q \in v.\text{req\_queue} \mid rank(v, q) = 0 \}$$

Chỉ các request trong *TopRequests(v)* mới được dùng để:

- Tính tổng tải trọng hiện tại.
- Xem xét khả năng thay thế.

Điều này đảm bảo vehicle chỉ bảo vệ những request mà nó là ứng viên tối ưu nhất.

**4. Replacement dựa trên Combined Score** Nếu tổng tải trọng vượt quá capacity, hệ thống xét loại bỏ các request trong  $TopRequests(v)$  theo thứ tự điểm *combined* tăng dần.

Việc thay thế chỉ xảy ra nếu:

$$combined(v, r_{new}) > combined(v, q)$$

Nghĩa là request mới phải có điểm cao hơn request bị loại trên cùng vehicle. Các request bị loại sẽ được đưa vào hàng đợi gán lại.

**5. Cascading Reassignment theo Tier tiếp theo** Giả sử một request  $q$  bị loại khỏi vehicle  $v$  và:

$$rank(v, q) = k$$

Khi gán lại, hệ thống không thử lại từ đầu mà chỉ xét các vehicle có:

$$rank(v', q) > k$$

Tức là bắt đầu từ Tier kế tiếp trong danh sách ứng viên của  $q$ .

Quá trình này có thể tiếp tục theo dạng lan truyền (cascade), nếu không gán được cho bất kỳ vehicle nào còn lại, request được đưa vào *PendingRequests*.

#### Algorithm 11 Tính điểm ứng viên tổng hợp

**Require:** *RawCandidates*

**Ensure:** *SortedCandidates*

- 1: **for**  $c \in RawCandidates$  **do**
- 2:      $c.r\_norm \leftarrow (c.r - r_{min}) / (r_{max} - r_{min})$
- 3:      $c.arr\_norm \leftarrow 1 - (c.arrival - arr_{min}) / (arr_{max} - arr_{min})$
- 4:      $c.combined \leftarrow \alpha c.r\_norm + \beta c.arr\_norm$
- 5: **end for**
- 6:  $SortedCandidates \leftarrow sort\_desc(RawCandidates, key = combined)$
- 7: **return** *SortedCandidates*

## Giải thích thuật toán Tính Combined Candidate Score

Thuật toán này chuẩn hóa và kết hợp hai yếu tố:

$$r_{norm} = \frac{r - r_{min}}{r_{max} - r_{min}}$$

$$arr_{norm} = 1 - \frac{arrival - arr_{min}}{arr_{max} - arr_{min}}$$

Trong đó:

- $r$  là điểm đánh giá từ R-tree.
- $arrival$  là thời gian xe đến request.

Điểm tổng hợp được tính bằng:

$$combined = \alpha r_{norm} + \beta arr_{norm}$$

Hai hệ số  $\alpha$  và  $\beta$  điều chỉnh mức độ ưu tiên giữa chiến lược và thời gian.

Danh sách ứng viên được sắp xếp giảm dần theo *combined*.

### Algorithm 12 AttemptAssign với Rank 0 Protection

**Require:**  $r, v, combined$

**Ensure:**  $(ok, removed)$

```

1:  $TopRequests \leftarrow \emptyset$ 
2: for  $q \in v.req\_queue$  do
3:    $CandList \leftarrow ComputeCombinedCandidates(q)$ 
4:   if  $rank(v, q) = 0$  then
5:      $TopRequests.append(q)$ 
6:   end if
7: end for
8:  $total \leftarrow \sum_{q \in TopRequests} demand(q)$ 
9: if  $total + demand(r) \leq capacity(v)$  then
10:    $v.req\_queue.append(r)$ 
11:   return  $(True, \emptyset)$ 
12: end if
13:  $Removable \leftarrow sort\_asc(TopRequests, key = combined)$ 
14:  $freed \leftarrow 0$ 
15:  $removed \leftarrow \emptyset$ 
16: for  $q \in Removable$  do
17:   if  $combined \leq combined(v, q)$  then
18:     continue
19:   end if
20:    $removed.append(q)$ 
21:    $freed \leftarrow freed + demand(q)$ 
22:   if  $total - freed + demand(r) \leq capacity(v)$  then
23:     loại bỏ  $removed$  khỏi  $v.req\_queue$ 
24:     thêm  $r$  vào  $v.req\_queue$ 
25:     return  $(True, removed)$ 
26:   end if
27: end for
28: return  $(False, \emptyset)$ 

```

## Gán request vào vehicle với thay thế

Thuật toán này xử lý trường hợp xe không còn đủ capacity.

Bước 1: Kiểm tra nếu còn đủ tải trọng thì gán trực tiếp.

Bước 2: Nếu không đủ, sắp xếp các request có thể loại bỏ theo điểm tăng dần.

Bước 3: Loại bỏ dần các request có điểm thấp hơn request mới cho đến khi đủ capacity.

Việc thay thế chỉ xảy ra nếu:

$$combined\_score_{new} > combined\_score_{old}$$

Nếu sau khi loại bỏ vân không đủ capacity, thao tác gán thất bại.

---

**Algorithm 13** Điều phối xe với JIT

**Require:**  $v$

```
1:  $ready\_time \leftarrow \max(curTime, v.busy\_until)$ 
2:  $Candidates \leftarrow \emptyset$ 
3: for  $r \in v.req\_queue$  do
4:   if  $\text{Feasible}(v, r, ready\_time) = False$  then
5:     continue
6:   end if
7:    $score \leftarrow S\_tree.evaluate(v, r)$ 
8:    $Candidates.append((score, r))$ 
9: end for
10: if  $Candidates \neq \emptyset$  then
11:    $(score^*, r^*) \leftarrow \arg \min Candidates$ 
12:   if  $v.atDepot \wedge ready\_time < r^*.jit\_departure$  then
13:      $EventQueue.push((r^*.jit\_departure, VEH\_FREE, v.id))$ 
14:      $v.waiting\_score \leftarrow score^*$ 
15:   return
16:   end if
17:    $Pickup(v, r^*)$ 
18:   return
19: end if
20: if  $v.atDepot = False$  then
21:    $ReturnDepot(v)$ 
22: end if
```

---

## Giải thích thuật toán DispatchVehicle với Just-In-Time Waiting

Thuật toán này quyết định hành động tiếp theo của xe khi nó trở nên rảnh.

Các request trong hàng đợi của xe được kiểm tra:

- Không vi phạm time window
- Không vi phạm  $l_w$
- Không vượt capacity

Request có điểm S-tree nhỏ nhất được chọn.

Nếu xe đang ở depot và chưa đến thời điểm xuất phát tối ưu (JIT departure), hệ thống sẽ lên lịch một sự kiện VEH\_FREE mới tại thời điểm phù hợp để trì hoãn xuất phát.

Nếu không có request hợp lệ và xe không ở depot, xe sẽ quay về depot.

---

**Algorithm 14** Dánh thức xe khi có request mới

---

**Require:**  $r$

```
1: for  $v \in Vehicles$  do
2:   if  $v.sleeping = False$  then
3:     continue
4:   end if
5:    $newScore \leftarrow S\_tree.evaluate(v, r)$ 
6:   if  $newScore < v.waiting\_score$  then
7:      $EventQueue.push((curTime, VEH\_FREE, v.id))$ 
8:      $v.sleeping \leftarrow False$ 
9:   end if
10: end for
```

---

## Giải thích thuật toán Wake-Up Trigger

Thuật toán này kích hoạt lại các xe đang chờ tại depot khi có request mới.

Với mỗi xe đang chờ:

$$newScore = S\_tree.evaluate(v, r)$$

Nếu:

$$newScore < waiting\_score$$

Xe sẽ được đánh thức ngay bằng cách đưa sự kiện VEH\_FREE vào EventQueue.

---

**Algorithm 15** Lấy đơn hàng

---

**Require:**  $v, r, ready\_time$

```
1:  $travel \leftarrow TravelTime(v.location, r.location)$ 
2:  $arrival \leftarrow ready\_time + travel$ 
3:  $service \leftarrow \max(arrival, r.tw\_start)$ 
4: if  $v.atDepot$  then
5:    $v.routes.append(new\_route)$ 
6: end if
7: if  $v.type = DRONE$  then
8:    $v.remaining\_range \leftarrow v.remaining\_range - travel$ 
9: end if
10:  $r.is\_picked \leftarrow True$ 
11:  $r.pickup\_time \leftarrow service$ 
12:  $v.remaining\_capacity \leftarrow v.remaining\_capacity - r.demand$ 
13:  $v.picked.append(r)$ 
14:  $v.location \leftarrow r.location$ 
15:  $v.busy\_until \leftarrow service$ 
16:  $v.req\_queue.remove(r)$ 
17:  $EventQueue.push((v.busy\_until, VEH\_FREE, v.id))$ 
```

---

## Giải thích thuật toán Thực hiện lấy đơn hàng

Thuật toán cập nhật trạng thái khi xe thực hiện pickup:

- Cập nhật thời gian đến và thời gian phục vụ.
- Nếu là drone, trừ tầm bay còn lại.
- Giảm capacity còn lại của xe.
- Thêm request vào danh sách đang chở.
- Cập nhật vị trí và busy\_until.
- Lên lịch sự kiện VEH\_FREE tại thời điểm hoàn tất phục vụ.

#### **Algorithm 16** Quay về depot

**Require:**  $v, ready\_time$

```

1:  $travel \leftarrow TravelTime(v.location, depot)$ 
2:  $arrival \leftarrow ready\_time + travel$ 
3: for  $r \in v.picked$  do
4:    $r.is\_served \leftarrow True$ 
5: end for
6:  $v.picked \leftarrow \emptyset$ 
7:  $v.location \leftarrow depot$ 
8:  $v.remaining\_capacity \leftarrow v.capacity$ 
9:  $v.busy\_until \leftarrow arrival$ 
10: if  $v.type = DRONE$  then
11:    $v.remaining\_range \leftarrow v.max\_range$ 
12: end if
13:  $EventQueue.push((arrival, VEH\_FREE, v.id))$ 

```

#### **Giải thích thuật toán Quay về depot**

Thuật toán này xử lý khi xe kết thúc một tuyến.

Các bước chính:

- Tính thời gian di chuyển về depot.
- Đánh dấu tất cả request đang chở là đã phục vụ.
- Reset capacity.
- Nếu là drone, reset trạng thái pin.
- Lên lịch sự kiện VEH\_FREE tại thời điểm đến depot.

---

**Algorithm 17** Chuỗi trả hàng bắt buộc

---

**Require:**  $v, r$ 

- 1:  $travel \leftarrow TravelTime(v.location, r.location)$
  - 2:  $arrival \leftarrow curTime + travel$
  - 3:  $r.is\_picked \leftarrow False$
  - 4:  $r.pickup\_time \leftarrow None$
  - 5: **if**  $arrival \leq r.time\_window^{end}$  **then**
  - 6:      $PendingRequests.append(r)$
  - 7: **end if**
  - 8:  $v.remaining\_capacity \leftarrow v.remaining\_capacity + r.demand$
  - 9:  $v.picked.remove(r)$
  - 10:  $v.location \leftarrow r.location$
  - 11:  $v.busy\_until \leftarrow arrival$
  - 12:  $EventQueue.push((arrival, VEH\_FREE, v.id))$
- 

## Giải thích thuật toán Failed Return Sequence

Thuật toán xử lý trường hợp vi phạm ràng buộc  $l_w$ .

Xe phải quay lại vị trí request vi phạm để hoàn tất việc pickup.

Nếu thời điểm đến vẫn nằm trong time window:

$$arrival\_time \leq r.time\_window^{end}$$

Request được đưa lại vào PendingRequests.

Sau đó:

- Hoàn trả capacity.
- Cập nhật vị trí và busy\_until.
- Lên lịch VEH\_FREE mới.

# Chương 5

## Thực nghiệm và kết quả

### 5.1 Tham số thực nghiệm Genetic Programming

Để đánh giá hiệu quả của thuật toán Genetic Programming, em thực hiện thí nghiệm trên nhiều bộ tham số khác nhau, ngoài ra mỗi bộ dữ liệu được chạy trên các random seed khác nhau (10 seed). Bảng 5.1 dưới đây tóm tắt 7 bộ cấu hình được sử dụng cho 7 loại bộ dữ liệu.

Bảng 5.1: Các bộ tham số cấu hình Genetic Programming

Tham số	config_6	config_10	config_12	config_20	config_50	config_100
<i>pop_size</i>	300	300	300	300	300	300
<i>max_gen</i>	100	100	100	100	100	150
<i>max_depth</i>	5	5	5	5	5	6
<i>c_rate</i>	0.8	0.8	0.8	0.8	0.8	0.8
<i>m_rate</i>	0.3	0.3	0.3	0.3	0.3	0.3
<i>tourn_size</i>	2	2	2	2	2	2
<i>assignment_n</i>	1	1	1	2	3	4

### 5.2 Kết quả thực nghiệm trên các instance

Bảng dưới đây tổng hợp kết quả chạy thực nghiệm với thuật toán GPHH + NSGA-II trên các bộ dữ liệu khác nhau. Mỗi instance có số lượng cá thể không trội trong quần thể cuối cùng (Pareto Count), số lượng phục vụ (Served), makespan và thời gian thực thi trung bình (Avg Execution Time – đơn vị giây).

Bảng 5.2: Kết quả thực nghiệm trên các instance Parallel VRPD-time window

Instance	Pareto Count	Served	Makespan	Avg Execution Time (s)
6.5.1	2	6	753.5815	31.3763
6.5.2	4	5	546.882	35.8187
6.5.3	2	6	919.9824	33.2896
6.5.4	2	5	722.0002	32.4915
6.10.1	2	5	1058.9239	35.3066
6.10.2	1	5	1169.0393	35.8175
6.10.3	2	6	1385.8205	34.5038

Instance	Pareto Count	Served	Makespan	Avg Execution Time (s)
6.10.4	4	5	1100.823	37.3742
6.20.1	1	5	4306.6813	37.2068
6.20.2	1	6	4202.1827	35.8605
6.20.3	1	5	6233.2145	35.0666
6.20.4	1	5	6174.7861	34.8349
10.5.1	2	9	1185.6894	41.7611
10.5.2	1	8	709.2037	47.8563
10.5.3	3	8	922.5506	46.5905
10.5.4	4	9	753.1453	44.0436
10.10.1	2	8	1901.1013	46.1242
10.10.2	2	10	2060.5354	42.3756
10.10.3	1	5	1508.8685	44.444
10.10.4	4	10	1418.9701	40.2319
10.20.1	1	6	5738.0668	39.4332
10.20.2	1	6	6203.1798	38.6409
10.20.3	1	5	6468.8684	39.0233
10.20.4	1	5	5015.2341	38.764
12.5.1	2	11	996.4854	48.9752
12.5.2	6	11	823.6148	60.6726
12.5.3	7	11	1128.751	52.0482
12.5.4	3	11	1085.9387	56.3877
12.10.1	1	10	1725.2414	49.4265
12.10.2	5	12	1731.7758	50.751
12.10.3	4	11	1901.2105	48.343
12.10.4	2	9	1798.519	52.2014
12.20.1	1	8	6705.1533	50.9126
12.20.2	1	6	6365.2014	46.228
12.20.3	2	9	6006.4111	48.4221
12.20.4	2	10	7555.7183	45.6136
20.5.1	8	18	751.8918	104.1788
20.5.2	8	19	866.1723	100.3964
20.5.3	4	19	564.6323	99.8427
20.5.4	7	20	644.0757	98.6055
20.10.1	8	18	1458.7736	100.8906
20.10.2	4	16	1389.1459	83.5277
20.10.3	10	15	1327.0895	115.8563
20.10.4	8	20	1398.6672	86.4605
20.20.1	2	14	4420.6347	74.2097
20.20.2	2	18	4213.7226	82.6238
20.20.3	2	11	5491.3251	74.0835
20.20.4	2	12	2440.0265	86.4051
50.10.1	11	43	2031.7726	295.152

<b>Instance</b>	<b>Pareto Count</b>	<b>Served</b>	<b>Makespan</b>	<b>Avg Execution Time (s)</b>
50.10.2	5	45	1686.5148	346.4246
50.10.3	10	49	1815.7032	282.5603
50.10.4	13	48	1695.3279	385.0499
50.20.1	10	39	4171.1786	286.4681
50.20.2	5	29	4111.6578	295.7605
50.20.3	38	30	4865.9731	297.4294
50.20.4	11	19	4941.4782	241.8152
50.30.1	4	19	9010.177	218.8475
50.30.2	2	21	7987.9054	207.5759
50.30.3	5	21	8783.3467	206.6801
50.30.4	14	16	8208.8034	251.7298
50.40.1	1	24	14160.1392	225.7832
50.40.2	13	29	14590.2524	188.239
50.40.3	2	20	13166.9166	192.806
50.40.4	1	19	13180.8472	212.0234
100.10.1	21	86	1793.2276	2097.7815
100.10.2	12	90	1847.655	2597.1353
100.10.3	13	91	2010.2426	2109.4815
100.10.4	13	88	1857.1868	2401.2984
100.20.1	19	34	4451.3342	1508.6685
100.20.2	72	62	4111.5737	1912.0375
100.20.3	6	56	5290.6081	1506.2223
100.20.4	15	60	4568.0367	2298.4555
100.30.1	24	49	8929.2884	1568.3123
100.30.2	10	56	8826.6966	1886.5068
100.30.3	8	44	9131.3549	1490.6471
100.30.4	9	39	9284.0704	877.2802
100.40.1	9	40	14931.751	1878.3786
100.40.2	7	39	15249.1897	1405.7403
100.40.3	6	59	15323.1139	864.3476
100.40.4	8	41	14706.4801	1510.2987

# Chương 6

## Kết luận

Qua quá trình thực hiện đồ án Project III với đề tài “Parallel VRPD - time window” sử dụng giải thuật Genetic Programming, sinh viên đã có cơ hội tiếp cận sâu sắc với bài toán lập lịch vận tải động trong môi trường không chắc chắn, kết hợp giữa xe tải và drone—một hướng nghiên cứu đang rất được quan tâm trong lĩnh vực logistics hiện đại.

Các kết quả và đóng góp chính đạt được bao gồm:

- Xây dựng mô hình toán học hoàn chỉnh cho bài toán Parallel Vehicle Routing Problem with Time Windows với các ràng buộc phức tạp.
- Đề xuất và triển khai thành công khung giải pháp dựa trên **Genetic Programming Hyper-heuristic (GPHH)** kết hợp NSGA-II, trong đó:
  - Tiến hóa đồng thời hai cây quy tắc riêng biệt;
  - **Routing Rule (cây R)** – quyết định phương tiện nào được phân bổ để phục vụ yêu cầu mới xuất hiện.
  - **Sequencing Rule (cây S)** – quyết định thứ tự phục vụ các yêu cầu đang chờ trong hàng đợi của từng phương tiện.

Tập terminal được thiết kế phong phú, phản ánh các yếu tố thực tế quan trọng. Tập hàm được chọn gọn nhẹ nhưng mạnh mẽ nhằm đảm bảo tính diển giải cao và khả năng ứng dụng trong môi trường thời gian thực.

- Tích hợp NSGA-II để giải quyết bài toán tối ưu đa mục tiêu, cơ chế giả lập sự kiện rời rạc.
- Thực hiện thực nghiệm trên các bộ dữ liệu với quy mô từ 10 đến 100 yêu cầu, thử nghiệm nhiều cấu hình tham số (pop\_size, assignment\_n, ...), và thu được tập Pareto front chất lượng, chứng minh khả năng cân bằng hiệu quả giữa hai mục tiêu chính trong môi trường động.

Những kết quả đạt được không chỉ khẳng định tính khả thi của cách tiếp cận GPHH trong bài toán VRPD động mà còn mở ra tiềm năng ứng dụng thực tế trong lĩnh vực giao hàng nhanh và logistics đô thị.

Trong tương lai, em dự định tiếp tục phát triển theo các hướng sau:

- Mở rộng tập terminal và function set để tăng khả năng biểu diễn của các quy tắc tiến hóa.
- Thủ nghiệm kết hợp chuyển giao kiến thức giữa các instance bài toán khác nhau.
- Thêm các ràng buộc thực tế hơn như ảnh hưởng của thời tiết đến tốc độ drone, nhiều depot, hoặc tải trọng thay đổi theo tuyến đường.

- Tối ưu hóa hiệu suất tính toán (song song hóa đánh giá cá thể, giảm số lần giả lập) để áp dụng cho quy mô lớn hơn.
- Khai thác khả năng ứng dụng thực tế trong hệ thống giao hàng thông minh tại Việt Nam.

Cuối cùng, em xin cảm ơn TS. Nguyễn Khánh Phương và cô Trần Thị Huế đã tận tình hướng dẫn, định hướng và hỗ trợ em trong suốt quá trình thực hiện đồ án ạ!