SOICT Hackathon 2024:
TIKI-Container Shipping Route Coordination

Hoàng Văn Nhân
HUST
wtnbyz@gmail.com

Lê Nguyễn Phước Thành
HUST
ThanhLNP@sis.hust.edu.vn

1 Giới thiệu

Ngành vận tải container đóng vai trò then chốt trong nền kinh tế toàn cầu, đảm bảo sự lưu thông hàng hóa giữa các quốc gia và khu vực. Trong bối cảnh cạnh tranh ngày càng gia tăng, việc tối ưu hóa lộ trình vận chuyển không chỉ giúp giảm chi phí mà còn nâng cao hiệu quả hoạt động của doanh nghiệp.

Bài toán TIKI-Container Shipping Route Coordination đặt ra thách thức trong việc phối hợp hoạt động của nhiều đầu kéo, rơ mooc và container để đáp ứng một loạt các yêu cầu vận chuyển với các ràng buộc phức tạp về sức chứa, trạng thái và thời gian tác nghiệp.

Báo cáo này nhằm trình bày giải thuật được đề xuất để giải quyết bài toán trên, tập trung vào việc thiết kế các cấu trúc dữ liệu hiệu quả và thuật toán phân bổ yêu cầu tối ưu.

2 Mô tả bài toán

Bài toán đặt ra yêu cầu lập kế hoạch lộ trình cho một tập hợp các đầu kéo để vận chuyển container giữa các điểm cụ thể, đáp ứng các yêu cầu vận chuyển đã định trước. Các thành phần chính của bài toán bao gồm:

- **Đầu kéo (Trucks):**
  - Có N đầu kéo, mỗi đầu kéo có vị trí khởi đầu riêng.
  - Mỗi đầu kéo có khả năng gắn hoặc tháo rơ mooc.
  - Hành động của đầu kéo:
    * PICKUP_TRAILER: Gắn rơ mooc trống vào đầu kéo.
    * DROP_TRAILER: Tháo rơ mooc trống khỏi đầu kéo.
    * PICKUP_CONTAINER: Lấy container và đặt lên rơ mooc đang gắn với đầu kéo.
    * DROP_CONTAINER: Hạ container từ rơ mooc đang gắn với đầu kéo.
    * PICKUP_CONTAINER_TRAILER: Gắn rơ mooc có sẵn container vào đầu kéo.
    * DROP_CONTAINER_TRAILER: Tháo rơ mooc có chứa container khỏi đầu kéo.
    * STOP: Kết thúc lộ trình tại vị trí hiện tại.
- **Rơ mooc (Trailers):**
  - Rơ mooc trống có sẵn tại một số điểm cố định.
  - Rơ mooc có sẵn container có thể được lấy tại các điểm yêu cầu tương ứng.
- **Container:**
  - Có hai loại container: 20ft và 40ft.
  - Sức chứa của rơ mooc tối đa là 2 đơn vị (container 20ft) hoặc 1 đơn vị (container 40ft).

- **Điểm giao nhận (Points):** Mạng lưới gồm M điểm, với khoảng cách giữa các điểm được xác định trước.
- **Ràng buộc:**
  - Sức chứa: Rơ mooc không được chở quá sức chứa tối đa.
  - Trạng thái đầu kéo: Một số hành động chỉ thực hiện được khi đầu kéo ở trạng thái nhất định (ví dụ: phải có rơ mooc mới thực hiện được PICKUP_CONTAINER).
  - Hành động hợp lệ: Các hành động phải tuân thủ quy định về trạng thái và điều kiện.
- **Mục tiêu tối ưu hóa:**
  - Tối thiểu hóa hàm mục tiêu
    $$F = \alpha F_1 + F_2$$
    trong đó:
    * F1: Thời gian hoàn thành muộn nhất trong số các đầu kéo.
    * F2: Tổng thời gian di chuyển của tất cả các đầu kéo.
  - Việc tối ưu hóa hàm mục tiêu giúp giảm thiểu thời gian vận chuyển và tăng hiệu quả hoạt động.

3 Thiết kế cấu trúc dữ liệu

Trong quá trình triển khai thuật toán giải quyết bài toán TIKI-Container Shipping Route Coordination, chúng tôi đã thiết kế và sử dụng các cấu trúc dữ liệu chính sau đây:

1. Cấu trúc dữ liệu cho đầu kéo (Trucks)
2. Cấu trúc dữ liệu cho yêu cầu vận chuyển (Requests)
3. Cấu trúc dữ liệu cho khoảng cách giữa các điểm (Distances)
4. Các cấu trúc dữ liệu hỗ trợ khác.

3.1 Cấu trúc dữ liệu cho đầu kéo (Trucks)

- **Mục đích:** Lưu trữ thông tin về mỗi đầu kéo, bao gồm trạng thái hiện tại, lộ trình dự kiến, và các thông tin cần thiết để quản lý và cập nhật trong quá trình thực hiện thuật toán.
- **Thiết kế chi tiết:**
  - Biến `trucks`: Là một dictionary (từ điển) trong đó mỗi key là `truck_id` (ID của đầu kéo), và value là một dictionary chứa thông tin về đầu kéo đó.
  - Thông tin lưu trữ cho mỗi đầu kéo:

Thuộc tính	Kiểu dữ liệu	Mô tả
location	int	Vị trí ban đầu của đầu kéo.
capacity	int	Sức chứa tối đa của rơ mooc (theo đơn vị container 20ft, thường là 2).
route	list	Danh sách các hành động (lộ trình) mà đầu kéo sẽ thực hiện.
load	int	Tải trọng hiện tại trên rơ mooc.
time	float	Thời gian đã sử dụng của đầu kéo.
current_location	int	Vị trí hiện tại của đầu kéo trong quá trình thực hiện lộ trình.
has_trailer	bool	Trạng thái cho biết đầu kéo có đang gắn rơ mooc hay không.
trailer_load	int	Tải trọng hiện tại trên rơ mooc (theo đơn vị container 20ft).

- **Cách sử dụng:**
  - Khởi tạo: Khi đọc dữ liệu đầu vào, chúng tôi khởi tạo `trucks` với thông tin vị trí ban đầu và các thuộc tính mặc định cho mỗi đầu kéo.
  - Cập nhật trạng thái: Trong quá trình gán yêu cầu và xây dựng lộ trình, các thuộc tính như `route`, `time`, `current_location`, `has_trailer`, và `trailer_load` được cập nhật liên tục để phản ánh trạng thái hiện tại của đầu kéo.
  - Quản lý lộ trình: Thuộc tính route lưu trữ danh sách các hành động mà đầu kéo sẽ thực hiện, mỗi hành động bao gồm:
    - `point`: Điểm thực hiện hành động.
    - `action`: Loại hành động (ví dụ: PICKUP_TRAILER, DROP_CONTAINER, v.v.).
    - `request_id`: ID của yêu cầu liên quan (nếu có).

Ví dụ về một đầu kéo trong cấu trúc dữ liệu:

```
trucks = {
    1: {
        'location': 4,
        'capacity': 2,
        'route': [],
        'load': 0,
        'time': 0,
        'current_location': 4,
        'has_trailer': False,
        'trailer_load': 0
    },
    # Các đầu kéo khác...
}
```

3.2 Cấu trúc dữ liệu cho yêu cầu vận chuyển (Requests)

- **Mục đích:** Lưu trữ thông tin chi tiết về mỗi yêu cầu vận chuyển, bao gồm các điểm lấy và hạ container, loại container, và trạng thái gắn yêu cầu.
- **Thiết kế chi tiết:**
  - Biến `requests`: Là một dictionary trong đó mỗi key là `req_id` (ID của yêu cầu), và value là một dictionary chứa thông tin về yêu cầu đó.

Thuộc tính	Kiểu dữ liệu	Mô tả
size	int	Kích thước container (20 hoặc 40).
pickup_point	int	Điểm lấy container.
pickup_action	str	Hành động lấy container (PICKUP_CONTAINER hoặc PICKUP_CONTAINER_TRAILER).
pickup_duration	int	Thời gian tác nghiệp tại điểm lấy container.
drop_point	int	Điểm hạ container.
drop_action	str	Hành động hạ container (DROP_CONTAINER hoặc DROP_CONTAINER_TRAILER).
drop_duration	int	Thời gian tác nghiệp tại điểm hạ container.
assigned	bool	Trạng thái cho biết yêu cầu đã được gán cho đầu kéo hay chưa.

  - Thông tin lưu trữ cho mỗi yêu cầu:

- **Cách sử dụng:**
  - Khởi tạo: Khi đọc dữ liệu đầu vào, chúng tôi khởi tạo `requests` với thông tin chi tiết của từng yêu cầu.
  - Gán yêu cầu: Thuộc tính `assigned` được cập nhật thành `True` khi yêu cầu được gán cho một đầu kéo.

Ví dụ về một yêu cầu trong cấu trúc dữ liệu:

```
requests = {
    1: {
        'size': 20,
        'pickup_point': 8,
        'pickup_action': 'PICKUP_CONTAINER',
        'pickup_duration': 3,
        'drop_point': 7,
        'drop_action': 'DROP_CONTAINER',
        'drop_duration': 4,
        'assigned': False
    },
    # Các yêu cầu khác...
}
```

3.3 Cấu trúc dữ liệu cho khoảng cách giữa các điểm (Distances)

- **Mục đích:** Lưu trữ thông tin về khoảng cách giữa các điểm trong mạng lưới, phục vụ cho việc tính toán thời gian di chuyển giữa các điểm trong lộ trình.
- **Thiết kế chi tiết:**
  - Biến `distances`: Là một dictionary lồng nhau, trong đó:
    - Key cấp 1 là `point_i` (điểm xuất phát).
    - Key cấp 2 là `point_j` (điểm đến).
    - Value là `distance_ij` (khoảng cách từ `point_i` đến `point_j`).
- **Cách sử dụng:**

- **Truy xuất khoảng cách:** Trong quá trình tính toán thời gian di chuyển, chúng tôi truy xuất khoảng cách giữa hai điểm bằng cách sử dụng `distances[point_i][point_j]`.
- **Khởi tạo:** Khi đọc dữ liệu đầu vào, chúng tôi xây dựng `distances` dựa trên thông tin khoảng cách giữa các điểm được cung cấp.

Ví dụ về cấu trúc dữ liệu `distances`:

```
distances = {
    1: {2: 95, 3: 91, 4: 88, ...},
    2: {1: 95, 3: 92, 4: 56, ...},
    # Các điểm khác...
}
```

3.4 Các cấu trúc dữ liệu hỗ trợ khác

1. **Danh sách yêu cầu chưa được gán (`unassigned_requests`):**
   - **Mục đích:** Quản lý danh sách các yêu cầu chưa được gán cho đầu kéo, giúp thuật toán dễ dàng truy cập và xử lý.
   - **Cách sử dụng:**
     - Ban đầu, `unassigned_requests` được khởi tạo bằng danh sách tất cả các `req_id` trong `requests`.
     - Trong quá trình gán yêu cầu, khi một yêu cầu được gán, nó sẽ bị loại bỏ khỏi `unassigned_requests`.
2. **Bản sao tạm thời của đầu kéo (`temp_truck`):**
   - **Mục đích:** Sử dụng để thử nghiệm việc gán yêu cầu cho đầu kéo mà không ảnh hưởng đến trạng thái thực tế, giúp xác định đầu kéo phù hợp nhất.
   - **Cách sử dụng:**
     - Khi xem xét việc gán một yêu cầu cho một đầu kéo, chúng tôi tạo một bản sao của trạng thái đầu kéo hiện tại (`temp_truck`).
     - Thực hiện các cập nhật trên `temp_truck` để mô phỏng việc thực hiện yêu cầu.
     - Nếu việc gán là khả thi và hiệu quả, chúng tôi cập nhật trạng thái thực tế của đầu kéo dựa trên `temp_truck`.

3.5 Tóm tắt và vai trò của các cấu trúc dữ liệu

- Sự kết hợp giữa `trucks`, `requests`, và `distances` cho phép thuật toán có thể truy xuất và cập nhật thông tin cần thiết một cách hiệu quả trong quá trình gán yêu cầu và tối ưu hóa lộ trình.
- Việc sử dụng các cấu trúc dữ liệu dạng `dictionary` giúp truy cập nhanh chóng đến thông tin của các đầu kéo và yêu cầu thông qua `truck_id` và `req_id`.
- Các cấu trúc dữ liệu hỗ trợ khác như `unassigned_requests`, `temp_truck`, và `route` đóng vai trò quan trọng trong việc thử nghiệm và xây dựng lộ trình khả thi mà không làm ảnh hưởng đến trạng thái thực tế cho đến khi quyết định được đưa ra.
- Các hàm hỗ trợ được thiết kế để tương tác chặt chẽ với các cấu trúc dữ liệu, đảm bảo tính nhất quán và hiệu quả trong quá trình tính toán và cập nhật.

4 Mô tả thuật toán

Thuật toán được thiết kế để nhằm giải quyết bài toán tối ưu hóa lộ trình vận chuyển container, với mục tiêu tối thiểu hóa thời gian hoàn thành muộn nhất (F_1) và tổng thời gian di chuyển (F_2) của các đầu kéo, đồng thời tuân thủ các ràng buộc về sức chứa và trạng thái của đầu kéo.

4.1 Tổng quan thuật toán

Thuật toán bao gồm ba giai đoạn chính:

1. **Khởi tạo và đọc dữ liệu đầu vào:** Đọc và lưu trữ thông tin về các điểm, khoảng cách giữa các điểm, vị trí rơ mooc, thông tin đầu kéo và các yêu cầu vận chuyển.
2. **Phân bổ yêu cầu và xây dựng lộ trình ban đầu:**
   - **Sắp xếp yêu cầu theo thứ tự ưu tiên:** Yêu cầu có hành động `PICKUP_CONTAINER_TRAILER` được ưu tiên trước do chúng yêu cầu đầu kéo không có rơ mooc.
   - **Gán yêu cầu cho đầu kéo:**
     - Lặp qua các yêu cầu chưa được gán.
     - Với mỗi yêu cầu, tìm đầu kéo phù hợp nhất có thể thực hiện yêu cầu mà không vi phạm ràng buộc.
     - Cập nhật trạng thái đầu kéo và yêu cầu sau khi gán.
3. **Cải thiện giải pháp thông qua tối ưu hóa:**
   - Hoán đổi yêu cầu giữa các đầu kéo.
   - Chèn các yêu cầu chưa được gán vào lộ trình của các đầu kéo nếu khả thi.

4.2 Chi tiết thuật toán

4.2.1 Khởi tạo và đọc dữ liệu đầu vào

- Đọc thông tin về các điểm và khoảng cách: Sử dụng cấu trúc dữ liệu `distances` để lưu trữ khoảng cách giữa các điểm.
- Đọc thông tin rơ mooc: Lưu trữ vị trí và thời gian gắn/rút rơ mooc (`trailer_location`, `trailer_attach_time`).
- Đọc thông tin đầu kéo: Khởi tạo danh sách các đầu kéo với trạng thái ban đầu (vị trí, sức chứa, không có rơ mooc, tải trọng bằng 0).
- Đọc yêu cầu vận chuyển: Lưu trữ chi tiết của mỗi yêu cầu trong `requests`, bao gồm kích thước container, điểm lấy/hạ, hành động, thời gian tác nghiệp, và trạng thái gán.

4.2.2 Phân bổ yêu cầu và xây dựng lộ trình ban đầu

(a) **Sắp xếp yêu cầu theo thứ tự ưu tiên**

- Yêu cầu có hành động `PICKUP_CONTAINER_TRAILER` được ưu tiên vì chúng yêu cầu đầu kéo không có rơ mooc.

(b) **Gán yêu cầu cho đầu kéo**

- **Lặp qua các yêu cầu chưa được gán:**
  - **Khởi tạo biến lưu trữ:**

* `best_truck_id`: ID của đầu kéo tốt nhất cho yêu cầu hiện tại.
* `min_completion_time`: Thời gian hoàn thành sớm nhất cho yêu cầu.

– **Lặp qua các đầu kéo:**
* Tạo bản sao tạm thời của đầu kéo (`temp_truck`) để thử nghiệm việc gán yêu cầu mà không ảnh hưởng đến trạng thái thực tế.
* Kiểm tra khả năng thực hiện yêu cầu:
· **Nếu yêu cầu là `PICKUP_CONTAINER_TRAILER`:**
· Kiểm tra đầu kéo không có rơ mooc.
· Cập nhật trạng thái `has_trailer` và `trailer_load` sau khi gán rơ mooc có container.
· **Nếu yêu cầu là `PICKUP_CONTAINER` hoặc khác:**
· Nếu đầu kéo không có rơ mooc, phải di chuyển đến vị trí rơ mooc để gắn (`PICKUP_TRAILER`).
· Kiểm tra sức chứa sau khi thêm container.
· Cập nhật `trailer_load`.
* Cập nhật thời gian và vị trí của đầu kéo sau khi thực hiện các hành động.
* Kiểm tra ràng buộc về sức chứa: Đảm bảo `trailer_load` không vượt quá sức chứa tối đa.
* Cập nhật `best_truck_id` nếu thời gian hoàn thành dự kiến tốt hơn.

– **Gán yêu cầu cho đầu kéo tốt nhất:**
* Cập nhật lộ trình (`route`), thời gian (`time`), vị trí hiện tại (`current_location`), trạng thái rơ mooc (`has_trailer`), và tải trọng (`trailer_load`) của đầu kéo.
* Đánh dấu yêu cầu đã được gán và loại bỏ khỏi danh sách yêu cầu chưa được gán.

4.2.3 Cải thiện giải pháp thông qua tối ưu hóa

(a) **Hoán đổi yêu cầu giữa các đầu kéo**

- **Mục tiêu:** Giảm thời gian hoàn thành muộn nhất F_1 bằng cách cân bằng tải giữa các đầu kéo.
- **Phương pháp:**
  – Lặp qua các cặp đầu kéo và thử hoán đổi các yêu cầu giữa họ.
  – Kiểm tra tính khả thi của việc hoán đổi dựa trên sức chứa và trạng thái.
  – Nếu hoán đổi giúp giảm F_1, chấp nhận hoán đổi và cập nhật lộ trình.

(b) **Chèn các yêu cầu chưa được gán**

- **Mục tiêu:** Đảm bảo tất cả các yêu cầu được phục vụ nếu khả thi.
- **Phương pháp:**
  – Lặp qua các yêu cầu chưa được gán.
  – Thử chèn yêu cầu vào lộ trình của các đầu kéo tại các vị trí khác nhau.
  – Kiểm tra tính khả thi sau khi chèn và chọn vị trí chèn tối ưu (tăng thời gian ít nhất).

4.2.4 Hoàn thiện lộ trình

- **Đối với mỗi đầu kéo:**
  - Nếu đầu kéo vẫn còn rơ mooc, thực hiện hành động `DROP_TRAILER` tại vị trí rơ mooc.
  - Kết thúc lộ trình bằng hành động `STOP` tại vị trí ban đầu.

4.3 Đảm bảo tuân thủ ràng buộc

- **Sức chứa:** Kiểm tra và cập nhật `trailer_load` sau mỗi hành động để không vượt quá sức chứa tối đa.
- **Trạng thái đầu kéo:** Theo dõi `has_trailer` để đảm bảo chỉ thực hiện các hành động phù hợp với trạng thái hiện tại.
- **Thời gian tác nghiệp:** Cộng thời gian tác nghiệp tại mỗi điểm vào tổng thời gian của đầu kéo.

4.4 Tính toán hàm mục tiêu

- F_1 (Makespan): Thời gian hoàn thành muộn nhất trong số các đầu kéo.
- F_2: Tổng thời gian di chuyển của tất cả các đầu kéo, tính bằng tổng khoảng cách giữa các điểm trong lộ trình của họ.
- **Hàm mục tiêu:**
$$F = \alpha F_1 + F_2,$$
với α là hệ số trọng số (theo đề bài).

5 Thuật toán dưới dạng mã giả

5.1 Thuật toán chính

Algorithm 1 Main Algorithm

Input: N, *distances*, *trailer_location*, *trailer_attach_time*, *trucks*, *requests*
Output: Assigned routes for each truck, F_1, F_2

$unassigned\_requests \leftarrow$ **SortRequests**(*requests*)
InitializeTrucks(*trucks*)
AssignRequests(*trucks*, $unassigned\_requests$, *distances*, *trailer_location*, *trailer_attach_time*)
ImproveSolution(*trucks*, *requests*, *distances*)
$(F_1, F_2) \leftarrow$ **CalculateObjective**(*trucks*, *distances*)
OutputResult(*trucks*)

5.2 Hàm sắp xếp yêu cầu

Algorithm 2 SortRequests

```
Function SortRequests(requests)
    Return requests sorted by priority:
        Priority(req) ←
            if req.pickup_action == 'PICKUP_CONTAINER_TRAILER' then 0
            else 1
            end if
```

5.3 Khởi tạo trạng thái đầu kéo

Algorithm 3 InitializeTrucks

```
Procedure InitializeTrucks(trucks)
    For each truck in trucks do
        truck.route ← empty list
        truck.time ← 0
        truck.current_location ← truck.location
        truck.has_trailer ← False
        truck.trailer_load ← 0
    End For
End Procedure
```

5.4 Gán yêu cầu cho đầu kéo

Algorithm 4 AssignRequests

```
1: Procedure AssignRequests(trucks, unassigned_requests, distances, trailer_location, trailer_attach_time)
2: while unassigned_requests is not empty do
3:    assigned ← False
4:    for req_id in unassigned_requests do
5:       req ← requests[req_id]
6:       required_capacity ← 1 if req.size == 20 else 2
7:       best_truck_id ← None
8:       min_completion_time ← Infinity
9:       for truck_id in trucks do
10:         temp_truck ← Copy of trucks[truck_id]
11:         actions ← empty list
12:         if CanAssignRequest(temp_truck, req, required_capacity, distances, trailer_location, trailer_attach_time) then
13:            completion_time ← temp_truck.time
14:            if completion_time < min_completion_time then
15:               min_completion_time ← completion_time
16:               best_truck_id ← truck_id
17:               best_actions ← actions
18:               best_temp_truck ← temp_truck
19:            end if
20:         end if
21:      end for
22:      if best_truck_id is not None then
23:         UpdateTruck(trucks[best_truck_id], best_temp_truck, best_actions)
24:         requests[req_id].assigned ← True
25:         Remove req_id from unassigned_requests
26:         assigned ← True
27:         Break
28:      end if
29:   end for
30:   if not assigned then
31:      Print "Cannot assign remaining requests."
32:      Break
33:   end if
34: end while
35: FinalizeRoutes(trucks, distances, trailer_location, trailer_attach_time)
36: End Procedure
```

5.5 Cải thiện giải pháp

Algorithm 5 ImproveSolution

Procedure ImproveSolution(*trucks*, *requests*, *distances*)
  previous_makespan ← **None**
while True do
    SwapRequests(*trucks*, *requests*, *distances*)
    InsertUnassignedRequests(*trucks*, *requests*, *distances*)
    (F_1, F_2) ← **CalculateObjective**(*trucks*, *distances*)
  if *previous_makespan* is not **None and** $F_1 \geq$ *previous_makespan* **then**
      Break
  end if
    previous_makespan ← F_1
end while
End Procedure

5.6 Chèn các yêu cầu chưa được gán

Algorithm 6 InsertUnassignedRequests

Procedure InsertUnassignedRequests(*trucks*, *requests*, *distances*)
for *req_id* **in** *requests* **where** *requests[req_id].assigned* == *False* **do**
  best_truck_id ← **None**
  min_increase ← **Infinity**
  for *truck_id* **in** *trucks* **do**
    for *possible insertion positions* **in** *truck.route* **do**
      if *inserting req_id is feasible and time increase* < *min_increase* **then**
        best_truck_id ← *truck_id*
        best_insertion ← *new_route*
        min_increase ← *time increase*
      end if
    end for
  end for
  if *best_truck_id* **is not None then**
    Update truck route and assign *req_id*
  end if
end for
End Procedure

Chú thích

- **Biến và hàm chính:**
  - `AssignRequests`: Gán yêu cầu cho đầu kéo và xây dựng lộ trình ban đầu.
  - `ImproveSolution`: Cải thiện giải pháp thông qua hoán đổi yêu cầu và chèn yêu cầu chưa được gán.
  - `CalculateObjective`: Tính toán hàm mục tiêu F_1 và F_2.

- `OutputResult`: Xuất lộ trình của các đầu kéo.

- **Kiểm tra tính khả thi và tuân thủ ràng buộc:**

  - Trong các hàm `CanAssignRequest`, `SwapRequests`, và `InsertUnassignedRequests`, luôn kiểm tra tính khả thi dựa trên sức chứa (`trailer_load`), trạng thái rơ mooc (`has_trailer`), và các ràng buộc khác.

- **Cập nhật trạng thái đầu kéo:**

  - Sau mỗi hành động, cập nhật `time`, `current_location`, `has_trailer`, và `trailer_load` của đầu kéo.

- **Tối ưu hóa giải pháp:**

  - Thông qua việc hoán đổi và chèn yêu cầu, thuật toán cố gắng giảm thời gian hoàn thành muộn nhất F_1 và tổng thời gian di chuyển F_2.

Kết luận

Thuật toán được thiết kế một cách chi tiết để đảm bảo tuân thủ tất cả các ràng buộc của bài toán, đồng thời tối ưu hóa hàm mục tiêu. Việc sử dụng các cấu trúc dữ liệu hiệu quả và các hàm hỗ trợ giúp thuật toán hoạt động một cách hiệu quả và có thể mở rộng cho các bộ dữ liệu lớn hơn.

Tài liệu

1. Gendreau, M., & Potvin, J.-Y. (Eds.). (2010). *Handbook of Metaheuristics* (2nd ed.). Springer.
2. Toth, P., & Vigo, D. (Eds.). (2014). *Vehicle Routing: Problems, Methods, and Applications* (2nd ed.). SIAM.
3. Laporte, G. (2009). Fifty Years of Vehicle Routing. *Transportation Science, 43*(4), 408–416.
4. Bräysy, O., & Gendreau, M. (2005). Vehicle Routing Problem with Time Windows, Part I: Route Construction and Local Search Algorithms. *Transportation Science, 39*(1), 104–118.
5. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.
6. Nguyen, T. B., & Kim, K. H. (2010). A dispatching method for tractor-trailer routing in container terminals. *Maritime Economics & Logistics, 12*(3), 327–345.
7. *TIKI Container Shipping Route Coordination Problem Description*. (2024). Tài liệu từ ban tổ chức cuộc thi SOICT Hackathon 2024.