ĐẠI HỌC BÁCH KHOA HÀ NỘI

Báo cáo mini project -IT4663

People and Parcel Share a Ride Large Size Ext 1

NGUYỄN ĐỨC THẮNG
Thang.nd225089@sis.hust.edu.vn

Ngành Công nghệ thông tin và truyền thông

Giảng viên hướng dẫn: Ts. Bùi quốc Trung _____

Chữ kí GVHD

Mã lớp: 157529

MSSV: 20225089

Khoa: Khoa học máy tính

Trường: Công nghệ Thông tin và Truyền thông

HÀ NỘI, 6/2025

MỤC LỤC

CHƯƠNG 1. GIỚI THIỆU ĐỀ TÀI 1

CHƯƠNG 2. Các thuật toán thực hiện 2

2.1 Phân tích đề bài 2

2.1.1 Đánh giá lời giải 2

2.1.2 Phân tích ví dụ 3

2.2 Backtracking 4

2.2.1 Ý tưởng 4

2.2.2 Mã giả thuật toán 4

2.2.3 Kết quả 6

2.3 Branch and cut 6

2.3.1 Ý tưởng 6

2.3.2 Mã giả thuật toán 7

2.3.3 Kết quả 9

2.4 Constraint programming 10

2.4.1 Ý tưởng 10

2.4.2 Thuật toán 10

2.4.3 Kết quả 11

2.5 Thuật toán tham lam 12

2.5.1 Ý tưởng 12

2.5.2 Mã giả thuật toán 13

2.5.3 Kết quả 14

2.6 Metaheuristic 14

2.6.1 Các hàm dùng chung 14

2.6.2 Iterated local search 16

2.6.3 Simulated Annealing 18
2.6.4 Variable Neighborhood Search 20
2.6.5 Tabu Search 24
2.6.6 Guided Local Search 26
2.7 Kết hợp các lời giải sẵn có và trình giải ortools 30
2.7.1 Ý tưởng 30
2.7.2 Kết quả 30
CHƯƠNG 3. Tổng hợp kết quả 31
CHƯƠNG 4. Đề xuất nâng cao 32
4.1 Cải Thiện Lời Giải Ban Đầu 32
4.2 Tối Ưu Hàm Đánh Giá 32
4.3 Cải Tiến Hiệu Suất 32

DANH MỤC HÌNH VẼ

DANH MỤC BẢNG BIỂU

CHƯƠNG 1. GIỚI THIỆU ĐỀ TÀI

Bài toán

- Có K taxi (đặt tại điểm 0) được lên lịch để phục vụ các yêu cầu vận chuyển bao gồm N yêu cầu hành khách $(1, 2, \ldots, N)$ và M yêu cầu hàng hóa $(1, 2, \ldots, M)$.
- Yêu cầu hành khách thứ i $(i = 1, \ldots, N)$ có điểm đón là i và điểm trả là $i + N + M$.
- Yêu cầu hàng hóa thứ i $(i = 1, \ldots, M)$ có điểm đón là $i + N$ và điểm trả là $i + 2N + M$.
- $d(i, j)$ là khoảng cách di chuyển từ điểm i đến điểm j $(i, j = 0, 1, \ldots, 2N+2M)$.
- Mỗi hành khách phải được phục vụ bởi một chuyến đi trực tiếp không bị gián đoạn (không có điểm dừng giữa điểm đón và điểm trả của hành khách trong mỗi tuyến đường).
- Mỗi taxi k có sức chứa $Q[k]$ để phục vụ các yêu cầu hàng hóa.
- Yêu cầu hàng hóa thứ i $(i = 1, \ldots, M)$ có khối lượng $q[i]$.

Yêu cầu

- Tính toán các tuyến đường cho các taxi sao cho thỏa mãn các ràng buộc trên và độ dài của tuyến đường dài nhất trong số K taxi là nhỏ nhất (để cân bằng độ dài giữa các taxi).
- Một tuyến đường của taxi k được biểu diễn bằng một chuỗi các điểm mà tuyến đường đó đi qua: $r[0], r[1], \ldots, r[L_k]$ trong đó $r[0] = r[L_k] = 0$ (điểm xuất phát).

Đầu vào

- **Dòng 1**: chứa N, M, và K $\quad (1 \leq N, M \leq 500, 1 \leq K \leq 100)$.
- **Dòng 2**: chứa $q[1], q[2], \ldots, q[M]$ $\quad (1 \leq q[i] \leq 100)$.
- **Dòng 3**: chứa $Q[1], Q[2], \ldots, Q[K]$ $\quad (1 \leq Q[i] \leq 200)$.
- **Dòng** $i + 3$ $(i = 0, 1, \ldots, 2N + 2M)$: chứa hàng thứ i của ma trận khoảng cách.

Đầu ra

- **Dòng 1**: chứa một số nguyên K.
- **Dòng** $2k$ $(k = 1, \ldots, K)$: chứa một số nguyên dương L_k.
- **Dòng** $2k + 1$ $(k = 1, \ldots, K)$: chứa một chuỗi L_k số nguyên $r[0], r[1], \ldots, r[L_k]$.

CHƯƠNG 2. Các thuật toán thực hiện

2.1 Phân tích đề bài

Cho K taxi (đều xuất phát và kết thúc tại điểm 0 - gọi là depot) phục vụ các yêu cầu vận chuyển bao gồm:

- N yêu cầu chở hành khách $(i = 1, \ldots, N)$
- M yêu cầu chở hàng hóa $(i = 1, \ldots, M)$

Yêu cầu hành khách Mỗi yêu cầu hành khách i:

- Điểm đón: i
- Điểm trả: $i + N + M$
- Phải được vận chuyển **trực tiếp**, không ghé bất kỳ điểm nào khác giữa điểm đón và điểm trả

Yêu cầu hàng hóa Mỗi yêu cầu hàng hóa i:

- Điểm lấy hàng: $i + N$
- Điểm giao hàng: $i + 2N + M$
- Trọng lượng hàng: $q[i]$

Taxi Mỗi taxi k có giới hạn trọng tải hàng hóa là $Q[k]$. Mỗi taxi phải:

- Bắt đầu và kết thúc tại điểm 0
- Không vượt quá trọng tải $Q[k]$ tại bất kỳ thời điểm nào

Khoảng cách $d(i, j)$ là khoảng cách giữa điểm i và j, với $i, j = 0, 1, \ldots, 2N+2M$

Mục tiêu Tìm các tuyến đường cho K taxi thỏa mãn các ràng buộc trên, sao cho:

Chiều dài tuyến dài nhất trong số K tuyến là nhỏ nhất

2.1.1 Đánh giá lời giải

Các thuật toán sử dụng trong báo cáo sẽ sử dụng testcase:

```
3 3 2
8 4 5
16 16
0 8 7 9 6 5 11 6 11 12 12 12 13
8 0 4 1 2 8 5 13 19 12 4 8 9
7 4 0 3 3 8 4 12 15 8 5 6 7
```

```
9 1 3 0 3 9 4 14 19 11 3 7 8
6 2 3 3 0 6 6 11 17 11 6 9 10
5 8 8 9 6 0 12 5 16 15 12 15 15
11 5 4 4 6 12 0 16 18 7 4 3 4
6 13 12 14 11 5 16 0 15 18 17 18 19
11 19 15 19 17 16 18 15 0 13 21 17 17
12 12 8 11 11 15 7 18 13 0 11 5 4
12 4 5 3 6 12 4 17 21 11 0 7 8
12 8 6 7 9 15 3 18 17 5 7 0 1
13 9 7 8 10 15 4 19 17 4 8 1 0
```

Ngoài ra với các thuật toán với thời gian tính toán ngắn sẽ dùng thêm bộ testcase để có cái nhìn trực quan hơn :

2.1.2 Phân tích ví dụ

Dữ liệu đầu vào

- $N = 3$, $M = 3$, $K = 2$
- Trọng lượng các yêu cầu hàng: $q = [8, 4, 5]$
- Trọng tải các taxi: $Q = [16, 16]$

Các điểm liên quan

- Các điểm đón hành khách: $1, 2, 3$
- Các điểm lấy hàng: $4, 5, 6$
- Các điểm trả hành khách: $7, 8, 9$
- Các điểm giao hàng: $10, 11, 12$
- Tổng cộng: $2N + 2M + 1 = 13$ điểm (đánh số từ 0 đến 12)

Ghép cặp yêu cầu

- Hành khách:

$$(1, 7)$$
$$(2, 8)$$
$$(3, 9)$$

- Hàng hóa:

$$(4, 10), \quad q = 8$$
$$(5, 11), \quad q = 4$$
$$(6, 12), \quad q = 5$$

Ma trận khoảng cách Ma trận $d[i][j]$ với $i, j = 0, \ldots, 12$ được cho trong dữ liệu đầu vào.

Mỗi tuyến đường của taxi k được biểu diễn bởi chuỗi các điểm $r[0], r[1], \ldots, r[L_k]$ sao cho:

$$r[0] = r[L_k] = 0$$

2.2 Backtracking

2.2.1 Ý tưởng

- Tìm điểm di chuyển đầu tiên mỗi xe
- Với mỗi điểm đầu tiên của mỗi xe tìm đường đi tương ứng với mỗi xe

2.2.2 Mã giả thuật toán

Algorithm 1: Thuật toán định tuyến xe với Backtracking

```
check_nextPoint (curClient, nextClient, vehicle)
// Kiểm tra điểm nextClient có thể là điểm tiếp
   theo của curClient trên xe vehicle
```

if *nextClient* > 0 ***và*** *visited[nextClient]* **then**
    return FALSE;

if *vượt quá dung lượng xe* **then**
    return FALSE;

if *vi phạm ràng buộc giao nhận* **then**
    return FALSE;

return TRUE;

```
try_schedule(point, vehicle)
// Hàm đệ quy xây dựng lộ trình cho từng xe
if point == 0 then
    if vehicle < NumTaxis then
        firstPoint[vehicle + 1], vehicle + 1;
    return;
for nextClient ← 0 to num_clients do
    if check_nextPoint(point, nextClient, vehicle) then
        // Ghi nhận nextClient là điểm tiếp theo của
           point của xe vehicle
        nextPoint[point] ← nextClient;
        if nextClient > 0 then
            nextClient, vehicle;
        else
            if vehicle == NumTaxis then
                // Cập nhật lộ trình tối ưu
                minRoute ← curRoute;
            else
                // Tiếp tục với xe tiếp theo
                try_schedule(firstPoint[vehicle + 1], vehicle + 1);
        // Quay lui
        nextPoint[point] ← -1;
```

```
check_firstPoint(point, vehicle)
if point có thể trở thành điểm thăm đầu tiên của xe vehicle then
    return TRUE;
return FALSE;
```

```
try_firstPoint(vehicle)
// Hàm chọn điểm xuất phát cho từng xe
for client ← 0 to num_clients do
    if check_firstPoint(client, vehicle) then
        // Gán client làm điểm đầu tiến cho xe
           vehicle
        firstPoint[vehicle] ← client;
        if vehicle < NumTaxis then
            try_firstPoint(vehicle + 1);
        else
            // Bắt đầu xây dựng lộ trình từ xe đầu
               tiên
            try_schedule(firstPoint[1], 1);
        // Quay lui

// main()
Khởi tạo tất cả biến;
try_firstPoint(1);
```

2.2.3 Kết quả

Output

```
2
2
0 0
14
0 1 7 5 4 10 6 3 9 12 11 2 8 0
```

- **Thời gian thực hiện:** 30.57734 giây
- **Số lời giải đã kiểm tra:** 1,846,014
- **Tổng quãng đường các xe đã đi:** 94

2.3 Branch and cut

2.3.1 Ý tưởng

- Tìm điểm di chuyển đầu tiên mỗi xe
- Với mỗi điểm đầu tiên của mỗi xe tìm đường đi tương ứng với mỗi xe
- sử dụng biến nbr để ước lượng số tổng quãng đường sẽ đi để tiến hành cắt bỏ

một số đường đi quá dài

2.3.2 Mã giả thuật toán

Algorithm 1: Thuật toán định tuyến xe với Branch and cut

```
check_nextPoint(curClient, nextClient, vehicle)
// Kiểm tra điểm nextClient có thể là điểm tiếp
   theo của curClient trên xe vehicle
if nextClient > 0 và visited[nextClient] then
    return FALSE;
if vượt quá dung lượng xe then
    return FALSE;
if vi phạm ràng buộc giao nhận then
    return FALSE;
return TRUE;
```

```
try_schedule(point, vehicle) // Hàm đệ quy xây dựng lộ
    trình cho từng xe
if point == 0 then
    if vehicle < NumTaxis then
        try_schedule(firstPoint[vehicle + 1], vehicle + 1);
    return;
for nextClient ← 1 to num_clients do
    if check_nextPoint(point, nextClient, vehicle) then
        // Ghi nhận nextClient là điểm tiếp theo của
            point của xe vehicle
        nextPoint[point] ← nextClient;
        num_visited ← num_visited + 1;
        if nextClient > 0 then
            predict ← curRoute + (num_clients + nbr - num_visited) *
             disMin;
            if predict < minRoute then
                try_schedule(nextClient, vehicle);
        else
            if vehicle == NumTaxis then
                // Cập nhật lộ trình tối ưu
                minRoute ← curRoute;
            else
                // Tiếp tục với xe tiếp theo
                predict ← curRoute + (num_clients + nbr - num_visited) *
                 disMin;
                if predict < minRoute then
                    try_schedule(firstPoint[vehicle + 1], vehicle + 1);
        // Quay lui
        nextPoint[point] ← -1;
        num_visited ← num_visited - 1;
```

```
check_firstPoint(point, vehicle)
if point có thể trở thành điểm thăm đầu tiên của xe vehicle then
    return TRUE;
return FALSE;
```

```
try_firstPoint(vehicle)
// Hàm chọn điểm xuất phát cho từng xe
s ← 0;
if firstPoint[vehicle - 1] > 0 then
    s ← firstPoint[vehicle - 1] + 1;
for client ← s to num_clients do
    if check_firstPoint(client, vehicle) then
        // Gán client làm điểm đầu tiên cho xe
           vehicle
        firstPoint[vehicle] ← client;
        visited[client] ← TRUE;
        num_visited ← num_visited + 1;
        if vehicle < NumTaxis then
            try_firstPoint(vehicle + 1);
        else
            // Bắt đầu xây dựng lộ trình từ xe đầu
               tiên
            nbr ← num_visited;
            try_schedule(firstPoint[1], 1);
        // Quay lui
        firstPoint[vehicle] ← -1;
        visited[client] ← FALSE;
        num_visited ← num_visited - 1;

// main()
Khởi tạo tất cả biến;
try_firstPoint(1);
```

2.3.3 Kết quả

Output

```
2
```

```
2
0 0
14
0 1 7 5 4 10 6 3 9 12 11 2 8 0
```

- **Thời gian thực hiện:** 2.38397 giây
- **Số lời giải đã kiểm tra:** 159,083
- **Tổng quãng đường các xe đã đi:** 94

2.4 Constraint programming

2.4.1 Ý tưởng

Ánh xạ điểm bắt đầu và kết thúc của các xe thanhg xe i điểm bắt đầu và điểm kết thúc được xánh xạ thành 2*(N+M) +i và 2*(N+M) +k +i

Sử dụng các constraint để thiết thiệt các ràng buộc cho bài toán và giải bằng ortools

2.4.2 Thuật toán

a, Biến và miền giá trị

- $X[i] \in \{0, 1, \ldots, N + 2K\}$ với $i \in \{0, 1, \ldots, N + K\}$: Điểm tiếp theo của điểm i trên lộ trình.
- $ir[i] \in \{1, 2, \ldots, K\}$ với $i \in \{1, \ldots, N + 2K\}$: Taxi phục vụ điểm i.
- $L[i] \in \{0, 1, \ldots, infinity\}$ với $i \in \{1, \ldots, N + 2K\}$: Khoảng cách tích lũy đến điểm i.
- $W[i] \in \{0, 1, \ldots, \max(\text{Capacity})\}$ với $i \in \{1, \ldots, N + 2K\}$: Trọng lượng tích lũy đến điểm i.
- Matrix[i][j] khoảng cách từ i đến j
- Request[i] lượng hàng hóa lấy tại điểm i (điểm nhận sẽ có giá trị dương diểm trả sẽ có giá trị dương)

b, Constraints

Các ràng buộc ban đầu

- $X[i] \neq X[j]$ for all $i, j \in \{1, 2, \ldots, 2(N + M) + K\}$ with $i \neq j$
- $ir[2(N + M) + i] \neq ir[2(N + M) + j]$ for all $i, j \in \{1, 2, \ldots, K\}$ with $i \neq j$
- $X[i] > 0$ for all $i \in \{1, \ldots, N + K\}$

Ràng buộc tích lũy khoảng cách và trọng lượng

Với mọi $i \in \{0, \ldots, N+K\}$ và $j \in \{1, \ldots, N+2K\}$:

$$X[i] = j \implies \begin{cases} L[j] = L[i] + \text{Matrix}[i][j], \\ W[j] = W[i] + \text{Request}[j], \\ ir[i] = ir[j], \\ i \neq j. \end{cases}$$

Ràng buộc cho PassengerRequest

Với mỗi request $(a_0, a_1) \in$ PassengerRequest:

$$ir[a_0] = ir[a_1], \quad X[a_0] = a_1.$$

Ràng buộc cho ParcelRequest

Với mỗi request $(a_0, a_1) \in$ ParcelRequest:

$$ir[a_0] = ir[a_1], \quad L[a_0] < L[a_1].$$

Ràng buộc khoảng cách tích lũy ban đầu

$$L[i] = 0 \quad \forall i \in \{N+1, \ldots, N+K\}.$$

Hàm mục tiêu Tối thiểu tổng khoảng cách tích lũy tại các điểm kết thúc của taxi:

$$\text{Minimize} \sum_{i=2*(N+M)+K+1}^{2*(N+M)+2K} L[i].$$

2.4.3 Kết quả

Output

```
2
14
0 1 7 5 4 10 6 3 9 12 11 2 8 0
2
0 0
```

- **Thời gian thực hiện:** 29.0997 giây

- **Tổng quãng đường các xe đã đi:** 94

2.5 Thuật toán tham lam

2.5.1 Ý tưởng

- Duyệt lần lượt từng request để phân cho các xe
- Khi chọn điểm tiếp theo cho các xe lựa chọn sao cho không vi phạm các ràng buộc và điểm tiếp theo sẽ được phân cho xe đang có lộ trình ngắn hơn.

2.5.2 Mã giả thuật toán

Algorithm 1: Thuật toán tham lam định tuyến xe với heuristic

Khởi tạo:

- cur_index_cap = [[] for _ in range(data['NumTaxis'])]: các điểm phải đi qua của các xe (khi điểm đón được thêm vào route của xe, điểm trả sẽ được thêm vào)
- cur_cap = [0] * data['NumTaxis']: số lượng hàng hóa tích lũy của các xe
- cur_dis = [0] * data['NumTaxis']: quãng đường tích lũy của các xe
- route = [[] for _ in range(data['NumTaxis'])]: lộ trình các xe
- add remove là các hành động cập nhật các biến khi thêm điểm hoặc loạ bỏ điểm khỏi đối tượng.

```
for (điểm đón, điểm trả) in PassengerRequest do
    taxi ← index of min(cur_dis) route[taxi] add (điểm đón) route[taxi] add
    (điểm trả)
for (điểm đón, điểm trả) in ParcelRequest do
    taxi ← index of min(cur_dis) if cur_index_cap[taxi] is empty then
        route[taxi] add (điểm đón) route[taxi] add (điểm trả)
    else
        if cur_cap[taxi] + Request[điểm đón] ≤ Capacity[taxi] then
            route[taxi] add (điểm đón) route[taxi] add (điểm trả)
        else
            while cur_cap[taxi] + Request[điểm đón] > Capacity[taxi] do
                point = cur_index_cap[taxi][0] for i in
                cur_index_cap[taxi][1:] do
                    if data['Matrix'][route[taxi][điểm cuối]][i] <
                    data['Matrix'][route[taxi][điểm cuối][point] then
                        point = i
                route[taxi] add (point) cur_index_cap[taxi] remove (point)
            route[taxi] add (điểm đón) cur_index_cap[taxi] add (điểm trả)
for taxi in range(NumTaxis) do
    while cur_index_cap[taxi] not empty do
        point = cur_index_cap[taxi][0] for i in cur_index_cap[taxi][1:] do
            if data['Matrix'][route[taxi][điểm cuối]][i] <
            data['Matrix'][route[taxi][điểm cuối]][point] then
                point = i
        route[taxi].add(point) cur_index_cap[taxi].remove(point)
    route[taxi].add(0) ;                                  // thêm depot
```

2.5.3 Kết quả

Output

```
2
6
0 1 7 3 9 0
10
0 2 8 4 5 10 6 11 12 0
```

- **Thời gian thực hiện:** ≈ 0 giây
- **Tổng quãng đường các xe đã đi:** 134
- **Thời gian thực hiện với testcase lớn hơn (N = 500, M= 500 ,K =50):** 0.031 giây
- **Tổng quãng đường các xe đã đi trong testcase lớn hơn :** 3739

2.6 Metaheuristic

2.6.1 Các hàm dùng chung

Algorithm 2: Các hàm dùng chung

`initial_solution()` Mỗi điểm được gán ngẫu nhiên vào lộ trình của các xe;

`// Tiến hành xử lý sơ bộ các yêu cầu`

Đưa các điểm trả về đúng vị trí tương ứng với điểm đón:;

+ Với yêu cầu chở người (passenger request), điểm trả được chèn ngay sau điểm đón;
+ Với yêu cầu chở hàng (parcel request), điểm trả được đưa vào cuối lộ trình;

return Lời giải ban đầu;

```
calculate_error(routes)
error_count ← 0;
// scheduleError lỗi về điểm đón và trả không cùng
   xe, pointError là thứ tự thăm lỗi
problematic_points ← {'scheduleError': [], 'pointError': []};
point_in_route ← {};
error_cap ← {};
for idx, route in enumerate(routes) do
    cap ← 0;
    for point in route do
        cap ← cap + Request[point];
        point_in_route[point] ← idx;
        if cap < 0 cap > data['Capacity'][idx] then
            error_count ← error_count + 1;
            error_cap[idx] ← 1;
    // Kiểm tra ràng buộc hành khách và bưu kiện
    Kiểm tra và lưu lại các reqest bị lỗi và lưu vào dạng lỗi tương ứng
return error_count, problematic_points, point_in_route, error_cap;
```

```
generate_neighbors(solution, error_cap)
// Tạo danh sách láng giềng
neighbors ← [];
// Di chuyển điểm trả
for each point in solution do
    if condition for moving drop point then
        new_solution ← modify with drop point change;
        neighbors.append(new_solution);
// Di chuyển điểm đón
for each point in solution do
    if condition for moving pickup point then
        new_solution ← modify with pickup point change;
        neighbors.append(new_solution);
// Đưa 1 cặp (điểm đón, điểm trả) sang xe khác
if condition for transferring pair to another vehicle then
    new_solution ← modify with pair transfer;
    neighbors.append(new_solution);
return neighbors;
```

2.6.2 Iterated local search

a, Ý tưởng

- khởi tạo lộ trình trình cho các xe
- Lặp lại max_iterations lần mỗi lần tìm lời giải lân cận tốt nhất và cập nhật
- Lời giải trả về sẽ là lời giải tốt nhất được tìm thấy

b, Mã giả thuật toán

Algorithm 2: Thuật toán Iterated Local Search

```
iterated_local_search(solution, max_iterations=100000,
max_time=30) start_time ← time.time();
best_solution ← copy.deepcopy(solution);
best_error, best_problematic_points, best_point_in_route, best_error_cap
 ← calculate_error(best_solution);
iteration ← 0;
while iteration < max_iterations ;
& (time.time() − start_time)<max_time & best_error>0 do
    // Tạo các lân cận từ giải pháp hiện tại
    neighbor_solutions ← generate_neighbors(best_solution,
     best_error_cap);
    for solution in neighbor_solutions do
        // Đánh giá giải pháp lân cận
        current_error, problematic_points, point_in_route, error_cap ←
         calculate_error(solution);
        if current_error < best_error then
            // Cập nhật giải pháp tốt nhất nếu tốt hơn
            best_solution ← copy.deepcopy(solution);
            best_error ← current_error;
        else if current_error == best_error then
            // Nếu cùng số lỗi, chọn giải pháp có tổng
               độ dài tuyến ngắn hơn
            if solution < best_solution then
                best_solution ← copy.deepcopy(solution);
                best_error ← current_error;
    iteration ← iteration + 1;
return best_solution;
```

c, Kết quả

Khi chạy thuật toán sử dụng với tham số max_iterations = 100000,max_time = 10

Output

2

```
8
0 3 9 6 12 4 10 0
8
0 2 8 5 11 1 7 0
```

- **Thời gian thực hiện:** ≈ 0 giây
- **Tổng quãng đường các xe đã đi:** 139
- **Thời gian thực hiện với testcase lớn hơn (N = 500, M= 500 ,K =50):** 6.22812 giây
- **Tổng quãng đường các xe đã đi trong testcase lớn hơn :** 105878

2.6.3 Simulated Annealing

a, Ý tưởng

- Khởi tạo lộ trình ban đầu cho các xe
- Sử dụng nguyên lý nhiệt độ giảm dần để chấp nhận các lời giải "tồi" với xác suất giảm theo thời gian
- Tại mỗi bước, tạo lời giải lân cận và quyết định chấp nhận hay không dựa trên nhiệt độ hiện tại
- Lời giải trả về là lời giải tốt nhất tìm được trong quá trình tìm kiếm

b, Mã giả thuật toán

Algorithm 2: Thuật toán Simulated Annealing

```
simulated_annealing(initial_solution, initial_temp=1000,
cooling_rate=0.99, min_temp=0.1, max_iterations=100000, max_time=1)
// Khởi tạo ban đầu
Khởi tạo các biến cần thiết
while temp > min_temp ;
& iteration<max_iterations & best_error>0 do
    // Tạo các lân cận
    neighbors ← generate_neighbors(current_solution,
     current_error_cap);
    if neighbors == ∅ then
        // Nếu không có lân cận, giảm nhiệt độ và
           tiếp tục
        temp ← temp × cooling_rate;
        iteration ← iteration + 1;
        continue;
    // Chọn ngẫu nhiên một lân cận
    neighbor_solution ← random.choice(neighbors);
    neighbor_error, _, _, neighbor_error_cap ←
     calculate_error(neighbor_solution);
    // Tính toán sự thay đổi chi phí
    delta ← neighbor_error - current_error;
    // Quyết định chấp nhận lân cận
    if delta < 0 random.random() < exp(-delta / temp) then
        current_solution ← neighbor_solution;
        current_error ← neighbor_error;
        current_error_cap ← neighbor_error_cap;
        // Cập nhật giải pháp tốt nhất nếu cần
        if current_error < best_error then
            best_solution ← copy.deepcopy(current_solution);
            best_error ← current_error;
    // Giảm nhiệt độ
    temp ← temp × cooling_rate;
    iteration ← iteration + 1;
return best_solution;
```

c, Kết quả

Khi chạy thuật toán sử dụng với tham số: initial_temp=1000, cooling_rate=0.9, min_temp=0.1, max_iterations=10000,max_time = 10

Output

```
2
10
0 1 7 2 8 3 9 6 12 0
6
0 4 5 10 11 0
```

- **Thời gian thực hiện:** ≈ 0 giây
- **Tổng quãng đường các xe đã đi:** 145
- **Thời gian thực hiện với testcase lớn hơn (N = 500, M= 500 ,K =50):** 10.85751 giây
- **Tổng quãng đường các xe đã đi trong testcase lớn hơn :** 106237

2.6.4 Variable Neighborhood Search

a, Ý tưởng

- Khởi tạo lộ trình ban đầu cho các xe
- Sử dụng một hàm tạo lân cận ngẫu nhiên (generateRandomNeighbor) kết hợp nhiều loại thao tác:
  - Hoán đổi điểm trong cùng tuyến (swap)
  - Di chuyển điểm trong cùng tuyến (relocate)
  - Hoán đổi điểm giữa các tuyến (cross_swap)
  - Di chuyển điểm sang tuyến khác (cross_relocate)
  - Đảo ngược đoạn tuyến (reverse)
- Lặp lại quá trình tìm kiếm, chấp nhận lời giải tốt hơn
- Dừng khi đạt max_iterations hoặc hết thời gian hoặc không còn lỗi

b, Mã giả thuật toán

Algorithm 2: Thuật toán Variable Neighborhood Search

```
neighborhood_One(data, solution, error_cap) neighbors ← [];
if error_cap is empty then
    return neighbors;
idx ← random.choice(list(error_cap.keys()));
route ← solution[idx];
// Chọn 1 điểm dón và thay đổi thức tự thăm
pickupPoints ← findPickupPoints(route);
// Choose one pickup point and change its order in
   the route
if len(pickupPoints) > 1 then
    pointToChange ← random.choice(pickupPoints);
    newOrderRoute ← changePickupOrder(route, pointToChange);
    neighbors.append(newOrderRoute);
return neighbors;
```

```
data, solution, error_cap neighbors ← [];
if error_cap is empty then
    return neighbors;
idx ← random.choice(list(error_cap.keys()));
route ← solution[idx];
// Chọn 1 điểm trả và thay đổi thức tự thăm
dropoffPoints ← findDropoffPoints(route);
// Choose one dropoff point and change its order
   in the route
if len(dropoffPoints) > 1 then
    pointToChange ← random.choice(dropoffPoints);
    newOrderRoute ← changeDropoffOrder(route, pointToChange);
    neighbors.append(newOrderRoute);
return neighbors;
```

```
neighborhood_three(data, solution, error_cap)
neighbors ← [];
if error_cap is empty then
    return neighbors;
idx ← random.choice(list(error_cap.keys()));
route ← solution[idx];
// chọn 1 reqest và đưa sang lộ trình khác
requests ← findRequestPairs(route);
// Move one request to another route
if len(requests) > 0 then
    requestToMove ← random.choice(requests);
    newRoute ← moveRequestToAnotherRoute(route, requestToMove);
    neighbors.append(newRoute);
return neighbors;
```

`variable_neighborhood_search`(*data, initial_solution, max_iterations=100000, max_time=30*)

Khởi tạo các biến cần thiết neighborhoods ← oneMethodNeighborhood;

while *iteration < max_iterations ;*
& $(time.time() - start\_time) < max\_time$ & $best\_error > 0$ **do**
    neighbors ← neighborhoods[current_neighborhood](data, best_solution, best_error_cap);
    if *neighbors is empty* **then**
        current_neighborhood ← (current_neighborhood + 1) % len(neighborhoods);
        continue*;*
    for *neighbor in neighbors* **do**
        current_error, _, _, error_cap ← `calculate_error`*(neighbor) ;*
        if *current_error < best_error* **then**
            Cập nhật lời giải ***break****;*
        else if *current_error == best_error* **then**
            if *data, neighbor < data, best_solution* **then**
                Cập nhật lời giải ***break****;*
    current_neighborhood ← (current_neighborhood + 1) % len(neighborhoods);
    iteration ← iteration + 1;
return *best_solution;*

c, Kết quả

Khi chạy thuật toán sử dụng với tham số: max_iterations=1000, max_time = 10

Output

```
2
6
0 5 6 11 12 0
10
0 1 7 2 8 3 9 4 10 0
```

- **Thời gian thực hiện:** 0.0025 giây
- **Tổng quãng đường các xe đã đi:** 141

- **Thời gian thực hiện với testcase lớn hơn (N = 500, M= 500 ,K =50):** 12.76436 giây
- **Tổng quãng đường các xe đã đi trong testcase lớn hơn :** 105498

2.6.5 Tabu Search

a, Ý tưởng

- Khởi tạo lộ trình cho các xe
- Sử dụng danh sách Tabu để ghi nhớ các bước di chuyển gần đây, tránh quay lại các giải pháp đã xét
- Lặp lại quá trình tìm kiếm lân cận, chọn giải pháp tốt nhất không nằm trong danh sách Tabu (trừ khi đáp ứng tiêu chí khát vọng)
- Cập nhật giải pháp tốt nhất tìm được
- Giải pháp trả về là giải pháp tốt nhất được tìm thấy sau số lần lặp tối đa hoặc khi hết thời gian

b, Mã giả thuật toán

Algorithm 2: Thuật toán Tabu Search

`tabu_search`(*data, initial_solution, max_iterations=1000, max_time=1, tabu_tenure=10*)
Khởi tạo các biến ban đầu
tabu_list ← new deque with maximum length tabu_tenure;
iteration ← 0;
while *iteration* < *max_iterations* ***and*** *(- start_time)* < *max_time* ***and*** *best_error* > *0* **do**
    neighbors ← `generate_neighbors`(*current_solution, current_error_cap, data*) ;
    if *neighbors is empty* **then**
        continue;
    for *each neighbor in neighbors* **do**
        neighbor_error, _, _, neighbor_error_cap ← `calculate_error`(*neighbor, data*) ;
        neighbor_distance ← neighbor, data['Matrix'];
        move_hash ← tuple of tuples representation of neighbor routes;
        is_tabu ← move_hash ∈ tabu_list;
        if *(not is_tabu* ***or*** *neighbor_error* < *best_error)* ***and*** *neighbor_error* ≤ *best_neighbor_error* **then**
            if *neighbor_error* < *best_neighbor_error* ***or*** *(neighbor_error == best_neighbor_error* ***and*** *neighbor_distance* < *best_neighbor_distance)* **then**
                cập nhật lời giải
    if *best_neighbor is null* **then**
        continue;
    tabu_list.append(move_hash of current_solution);
    if *current_error* < *best_error* ***or*** *(current_error == best_error* ***and*** *current_solution, data['Matrix']* < *best_solution, data['Matrix'])* **then**
        best_solution ← current_solution;
        best_error ← current_error;
    iteration ← iteration + 1;
return best_solution;

c, Kết quả

Khi chạy thuật toán sử dụng với tham số: max_iterations=1000, max_time = 10

Output

```
2
4
0 2 8 0
12
0 4 1 5 7 11 10 6 3 9 12 0
```

- **Thời gian thực hiện:** 0.03861 giây
- **Tổng quãng đường các xe đã đi:** 115
- **Thời gian thực hiện với testcase lớn hơn (N = 500, M= 500 ,K =50):** 13.4046 giây
- **Tổng quãng đường các xe đã đi trong testcase lớn hơn :** 106467

2.6.6 Guided Local Search

a, Ý tưởng

- Khởi tạo lộ trình cho các xe
- Sử dụng hàm phạt để hướng dẫn tìm kiếm tránh các cực trị địa phương
- Các cạnh (điểm liền kề) có chi phí cao sẽ bị phạt nhiều hơn
- Lặp lại quá trình tìm kiếm cục bộ với hàm mục tiêu được điều chỉnh bởi các mức phạt
- Lời giải trả về là lời giải tốt nhất được tìm thấy

b, Mã giả thuật toán

Algorithm 2: Thuật toán Thuật toán Guided Local Search

`calculateAugmentedObjective(`*solution, penalties, lambdaParam=0.3*`)` errorCount, problematicPoints, ← `calculate_error(`*solution*`)`;

totalDistance ← solution;

penaltySum ← 0;

for *mỗi req trong problematic_points['pointError']* **do**

    penaltyKey ← tuple(req);

    penaltySum ← penaltySum + penalties.get(penaltyKey, 0);

for *mỗi req trong problematic_points['scheduleError']* **do**

    penaltyKey ← tuple(req);

    penaltySum ← penaltySum + penalties.get(penaltyKey, 0);

for *mỗi capError trong problematic_points['capacityError']* **do**

    penaltyKey ← capError;

    penaltySum ← penaltySum + penalties.get(penaltyKey, 0);

return totalDistance + lambdaParam · penaltySum;

```
updatePenalties(solution, penalties, lambdaParam=0.3)
problematicPoints ← calculate_error(solution);
features ← danh sách rỗng;
utilities ← danh sách rỗng;
totalDistance ← solution;
for mỗi req trong problematic_points['pointError'] hoặc
 problematic_points['scheduleError'] do
    penaltyKey ← tuple(req);
    utility ← 1.0 / (1 + penalties.get(penaltyKey, 0));
    Thêm penaltyKey vào features;
    Thêm utility vào utilities;
for mỗi capError trong problematic_points['capacityError'] do
    penaltyKey ← capError;
    utility ← 1.0 / (1 + penalties.get(penaltyKey, 0));
    Thêm penaltyKey vào features;
    Thêm utility vào utilities;
if utilities không rỗng then
    maxUtility ← giá trị lớn nhất trong utilities;
    for mỗi i, feature trong enumerate(features) do
        if utilities[i] xấp xỉ bằng maxUtility then
            penalties[feature] ← penalties.get(feature, 0) + 1;
return penalties;
```

`guidedLocalSearch`(*solution, maxIterations=100000, maxTime=1, lambdaParam=0.3*) startTime ← thời gian hiện tại;
bestSolution ← bản sao sâu của solution;
bestError, $_{,,bestErrorCap}$← `calculate_error`(*bestSolution*);
penalties ← từ điển rỗng;
bestObjective ← `calculateAugmentedObjective`(*bestSolution, penalties, lambdaParam*);
iteration ← 0;
while *iteration < maxIterations và (thời gian hiện tại - startTime) < maxTime và bestError > 0* **do**
    neighborSolutions ← `generate_neighbors`(*bestSolution, bestErrorCap, penalties*);
    for *mỗi neighbor trong neighborSolutions* **do**
        currentError, $_{,,currentErrorCap}$← `calculate_error`(*neighbor*);
        currentObjective ← `calculateAugmentedObjective`(*neighbor, penalties, lambdaParam*);
        if *currentObjective < bestObjective hoặc (currentObjective xấp xỉ bằng bestObjective và currentError < bestError)* **then**
            bestSolution ← bản sao sâu của neighbor;
            bestError ← currentError;
            bestErrorCap ← currentErrorCap;
            bestObjective ← currentObjective;
    penalties ← `updatePenalties`(*bestSolution, penalties, lambdaParam*);
    iteration ← iteration + 1;
return bestSolution;

c, Kết quả

Khi chạy thuật toán sử dụng với tham số: max_iterations=1000, max_time = 10

Output

```
2
10
0 1 7 4 5 10 6 11 12 0
6
0 2 8 3 9 0
```

- **Thời gian thực hiện:** 0.00103 giây
- **Tổng quãng đường các xe đã đi:** 135
- **Thời gian thực hiện với testcase lớn hơn (N = 500, M= 500 ,K =50):** 10.83104 giây
- **Tổng quãng đường các xe đã đi trong testcase lớn hơn :** 105909

2.7 Kết hợp các lời giải sẵn có và trình giải ortools

2.7.1 Ý tưởng

Thư viện ortools cùng cấp trình giải riêng cho bà toán VehicleRouting cung cấp sẵn các function để dễ dàng hơn để mô hình hóa cho bài toán. Để kiểm tra sức mạnh thử nghiệm công cụ các thuật đều sẽ chạy với max_time = 10 gây với bộ test case lớn hơn (với N=500, M = 500, K =50) để rõ ràng so sánh

Các constraints để dể giải bài toán:

- K xe tại điểm xuất phát 0 mỗi xe có thể chở Q[i] với i= 1,..,k
- N khách hàng và M gói đồ cần giao có trọng tải q[i] với i= 1,. . . ,N
- N yêu cầu trở khách từ I đến i+N+M
- M yêu cầu chờ hàng từ i+N đến I +2N+M
- Các yêu cầu chở khách phải thực hiện ngay lập tức

2.7.2 Kết quả

Để có cái nhìn dễ dàng so sánh giưa các thuật toán khi sử dụng ortools trong phần này sẽ sử dụng

Algorithm	Tổng độ dài đường	Thời gian thực hiện
PATH_CHEAPEST_ARC	55933	10.0738
GREED DESCENT	56120	10.0704
GUIDED_LOCAL_SEARCH	56060	10.0642
TABU_SEARCH	55933	10.1572
SIMULATED_ANNEALING	56107	10.0664
GENERIC_TABU_SEARCH	55923	10.0514

CHƯƠNG 3. Tổng hợp kết quả

Sau khi cài đặt và sử dụng nhiều thuật toán và so sánh chất lượng lời giải với nhau ta thu được bảng so sánh như sau:

Algorithm	**Tổng độ dài đường đi**	**Thời gian thực hiện**
ORTOOLS PATH CHEAPEST ARC	110	0.03414
ORTOOLS GUIDED LOCAL SEARCH	110	1.005
ORTOOLS GUIDED LOCAL SEARCH	108	0.9977
ORTOOLS TABU SEARCH	108	1.0099
ORTOOLS SIMULATED ANNEALING	110	1.0065
ORTOOLS GENERIC TABU SEARCH	108	0.99995
GREEDY	134	≈ 0
BACKTRACKING	94	30.57734
BRACH AND CUT	94	2.38397
CONSTRAINT PROGRAMMING	94	29.0997
ITEREATED LOCAL SEARCH	139	≈ 0
SIMULATED ANNEALING	145	≈ 0
TABU SEARCH	115	0.03861
GUIDED LOCAL SEARCH	135	0.00103
VARIABLE NEIGHBORHOOD SEARCH	141	0.0025

Các thuật toán đều có thể tim được lời giải cho bài toán Vehicle Routing và cho được giải chấp nhận được tổng thời gian cho phép.

Nhận xét

- Các thuật toán Backtracking , Branch and cut n constraint programming có thể tìm ra các lời giải tốt nhất nhưng tốn rất nhiều chi phí tính toán
- Các lời giải sử dụng heuristic và metaheuristic cho lời giả nhanh hơn đáng kể nhưng chr có thể tìm được nhưng cực trị địa phương.
- Các thuật toán đều có thể tim được lời giải cho bài toán Vehicle Routing và cho được giải chấp nhận được tổng thời gian cho phép.

CHƯƠNG 4. Đề xuất nâng cao

Các thuật toán vẫn cònnhiều điểm để cải tiến và nâng cấp. Sau đâu là một số đề xuất có thể thực hiện với các thuật toán.

4.1 Cải Thiện Lời Giải Ban Đầu

Thay thế phương pháp ngẫu nhiên bằng heuristic thông minh:

Algorithm 3: Tạo lời giải ban đầu cải tiến

Khởi tạo k tuyến taxi rỗng

for *mỗi yêu cầu khách hàng* (p_i, d_i) **do**

Chọn taxi t có công suất dư lớn nhất Thêm p_i và d_i liên tiếp vào tuyến t

for *mỗi yêu cầu hàng hóa* (p_j, d_j) **do**

Chọn taxi t có công suất phù hợp Thêm p_j vào vị trí ngẫu nhiên trước d_j

Thêm điểm depot (0) vào đầu và cuối mỗi tuyến

4.2 Tối Ưu Hàm Đánh Giá

Đưa vào hàm đánh giá có trọng số:

$$\text{Error} = w_1 \cdot E_{\text{capacity}} + w_2 \cdot E_{\text{schedule}} + w_3 \cdot E_{\text{order}} \tag{4.1}$$

4.3 Cải Tiến Hiệu Suất

Áp dụng kỹ thuật cập nhật gia tăng:

$$\Delta D = D_{\text{new}} - D_{\text{old}} = c(a, c) + c(b, d) - c(a, b) - c(c, d) \tag{4.2}$$