TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

BÁO CÁO MÔN HỌC

HỌC PHẦN: TỐI ƯU LẬP KẾ HOẠCH

(Mã học phần: IT4663)

Đề tài:

TSP with Time Windows

Sinh viên thực hiện	MSSV
Lê Đức Quân	20204682
Nguyễn Quốc Việt	20204704
Phạm Việt Hoàng	20200245

Lớp: **141329**

Giảng viên hướng dẫn: Bùi Quốc Trung

Hà Nội, tháng 7 năm 2023

Mục lục

Lời mở đầu

Bài toán người bán hàng du lịch với cửa sổ thời gian (TSPTW) là bài toán tìm đường đi có chi phí tối thiểu đi qua từng thành phố trong tập hợp các thành phố, mỗi thành phố đi qua chính xác một lần, với điều kiện mỗi thành phố phải được thăm trong một khoảng thời gian nhất định. TSPTW đã nhận được sự chú ý đáng kể, trong nhiều năm, vì nó là cốt lõi của nhiều ứng dụng lập lịch và định tuyến quan trọng trong đời thực. Ở đây, chúng em sẽ trình bày một số phương pháp giải được áp dụng trong môn **Tối ưu lập kế hoạch** như: Vét cạn (Brute Force), Quy hoạch ràng buộc (CP), Quy hoạch nguyên tuyến tính (IP), Tham lam (Greedy), Heuristic. Trong đó mỗi phương pháp sẽ đưa ra cách mô hình khác nhau để dễ dàng triển khai trên các phần mềm giải quyết bài toán tối ưu như **Ortools**. Sau đó sẽ đưa ra kết quả thực nghiệm và đánh giá từng phương pháp .

Trong quá trình làm mini-project, chúng em xin chân thành cảm ơn thầy Bùi Quốc Trung đã đánh giá và hướng dẫn, có những nhận xét bổ ích để chúng em có thể hoàn thiện.

CÔNG VIỆC CỦA TỪNG THÀNH VIÊN

Nhận xét: Các thành viên trong nhóm đều tham gia đầy đủ và đóng góp xây dựng các phương pháp để hoàn thành bài tập lớn

Sinh viên	Công việc thực hiện
Lê Đức Quân	Nhóm trưởng, mô hình và code phương pháp CP
Nguyễn Quốc Việt	Mô hình và code phương pháp nhánh cận, ILP
Phạm Việt Hoàng	Mô hình và code phương pháp Greedy, Heuristic

I. Giới thiệu bài toán

Bài toán: Một nhân viên giao hàng lấy hàng ở kho (điểm 0) và cần đi giao hàng cho N khách hàng 1,2, … , N. Khách hàng i nằm ở điểm i và có yêu cầu giao hàng trong khoảng thời gian từ $e(i)$ đến $l(i)$ và giao hàng hết $d(i)$ đơn vị thời gian (s). Biết rằng $t(i,j)$ là thời gian di chuyển từ điểm i đến điểm j. Nhân viên giao hàng xuất phát từ kho tại thời điểmt_0, hãy tính toán lộ trình giao hàng cho nhân viên giao hàng sao cho tổng thời gian di chuyển là ngắn nhất.

II. Các phương pháp giải

1. Thuật toán nhánh cận

1.1 Mô hình hóa bài toán

1.1.1 Tập các biến

- **Lời giải:**

$$X = \{X_0, X_1, X_2, \dots, X_n\} \text{ với } X_i \text{ là điểm giao hàng thứ i}$$

- **Thời điểm điểm đến X_i:**

$$TGD_i = \{TGD_0, TGD_1, TGD_2, \dots, TGD_n\} \text{ với } TGD_i \text{ là thời điểm đến điểm giao hàng thứ } X_i$$

- **Thời điểm kết thúc giao hàng ở điểm X_i:**

$$KT_i = \{KT_0, KT_1, KT_2, \dots, KT_n\} \text{ với } KT_i \text{ là thời điểm kết thúc giao hàng ở điểm thứ } X_i$$

1.1.2 Miền giá trị

- **Lời giải:**

$$D(X_i) = \{1, 2, 3, \dots, N\}$$

- **Thời điểm đến điểm X_i:**

$$D(TGĐ_i) = R +$$

- **Thời điểm kết thúc giao hàng ở điểm X_i:**

$$D(KT_i) = R +$$

1.1.3 Định nghĩa

- **Lời giải:**

$X_0 = 0$: Xuất phát từ điểm giao hàng

- **Thời điểm đến điểm X_i:**
  - $TGĐ_0$ = 0
  - $TGĐ_i = KT_{i-1}$ + t(A_{i-1}, A_i) với mọi i khác 0
- **Thời điểm kết thúc giao hàng ở điểm X_i:**
  - KT_0 = 0
  - KT_i = max($TGĐ_i, e_{A_i}$) + d_{A_i}với mọi i khác 0

1.1.4 Ràng buộc và hàm mục tiêu

- **Mỗi thành phố đi qua đúng một lần:**

$X_i \neq X_j$ với mọi i ≠ j

- **Shipper phải đến điểm giao hàng X_i trước thời điểm l_{X_i}:**

$TGĐ_i \leq l_{X_i}$ với mọi i

- **Hàm mục tiêu:**

Cost = $KT_N + t(A_N, A_0)$

1.2 Xử lý bài toán

1.2.1 Khởi tạo các biến

Start

```
X = [0] * (n+1)
min_cost = 999999
min_X = [0] * (n+1)
check_visited =  [0] * (n+1)
TGD = [0] * (n+1)
KT = [0] * (n+1)

X[0] = 0
check_visited[0] = 1
TGD[0] = 0
KT[0] = 0
```

[67] ✓ 0.0s

```
min_t, min_d
```

[68] ✓ 0.0s

```
(10.0, 5.0)
```

1.2.2 Giải thuật nhánh cận

```python
def check(k, i):
    tgd = KT[k-1] + t[X[k-1]][i]
    if check_visited[i] == 1 or tgd > l[i]:
        return 0
    return 1

def Try(k):
    global X, min_cost, min_X, check_visited, TGD, KT
    for i in range(1,n+1):
        if check(k,i):
            check_visited[i] = 1
            X[k] = i
            TGD[k] = KT[k-1] + + t[X[k-1]][i]
            KT[k] = max(TGD[k], e[i]) + d[i]
            if k == n:
                cost = KT[k] + t[i][0]
                if cost < min_cost:
                    min_cost = cost
                    min_X = X.copy()
            else:
                if KT[k] + (min_d + min_t)*(n-k) +  min_t < min_cost:
                    Try(k+1)
            check_visited[i] = 0
```

✓ 0.0s

1.2.3 Hàm cắt tỉa

```python
if KT[k] + (min_d + min_t)*(n-k) +  min_t < min_cost:
    Try(k+1)
```

1.3 Kết quả thực nghiệm và thời gian chạy

Tên bộ dữ liệu	Kết quả chạy	Thời gian chạy
N5	465	Sấp sỉ 0
N10	779	0.003s
N100	None	Run Time Error
N200	None	Run Time Error
N300	None	Run Time Error
N500	None	Run Time Error
N600	None	Run Time Error
N700	None	Run Time Error
N900	None	Run Time Error
N1000	None	Run Time Error

2. Quy hoạch ràng buộc (Constraint Programming)

Mô hình bài toán theo Quy hoạch Ràng buộc:

Gọi $V = 0,1,2,\ldots,n,n+1$ là tập các node, trong đó $1,\ldots,n$ là các thành phố sẽ thăm, và node 0 và $n+1$ là điểm bắt đầu và kết thúc. Mỗi node liên kết với 1 cửa sổ thời gian $TW_i = [a_i, b_i]$ là khung thời gian giao hàng của thành phố i . Mỗi node cũng có 1 khoảng thời gian dur_i , đại diện cho thời gian giao hàng tại thành phố i. Với mỗi cặp thành phố i và j , có thời gian di chuyển là $t_{i,j}$

- **Định nghĩa 3 biến cho mỗi node $i \in V$**:
  - $Next_i$ là thành phố sẽ thăm tiếp theo của node i
  - $Cost_i$ là chi phí từ i đến $Next_i$
  - $Start_i$ là thời điểm giao hàng của node i
  - $Wait_i$ là thời gian để giao hàng tại node i
- **Miền giá trị:**
  - $D(Next_i) = \{1, \ldots, n+1\}$
  - $D(Cost_i) = \{0, \ldots, \max\}$
  - $D(Start_i) = [a_i, b_i]$
  - $D(Wait_i) = \{0, \ldots, max\}$
  - $Start_0 = 0,\ Cost_{n+1} = 0,\ Next_{n+1} = 0$
- **Ràng buộc:**
  - **Các đỉnh chỉ thăm 1 lần:**

$$alldifferent([Next_0, Next_1, \ldots, Next_{n+1}])$$

  - **Không tồn tại chu trình con:**

$$nocycle([Next_0, Next_1, \ldots, Next_n])$$

  - **Cập nhật chi phí di chuyển:**

$$Next_i = j \rightarrow Cost_i = t_i, j \; với \; \forall \; i,j \in \; V$$

  - **Cập nhật thời gian chờ giao hàng:**

$$Next_i = j \rightarrow Wait_i = Start_j - Start_i - Cost_i - dur_i \; với \; \forall \; i,j \in \; V,\ j \neq \; 0$$

  - **Thỏa mãn thời gian giao hàng:**

$$Next_i = j \; \rightarrow Start_i + dur_i + Cost_i \leq Start_j \; với \; \forall \; i,j \in \; V,\ j \neq \; 0$$

- **Hàm mục tiêu:**

$$f = \min\{Start_{n+1}\}$$

Sử dụng công cụ **OR-Tools** để giải quyết bài toán

- OR-Tools là một bộ phần mềm nguồn mở tối ưu hoá, được điều chỉnh để giải quyết các vấn đề khó khăn nhất trên thế giới trong việc định tuyến xe, luồng, lập trình số nguyên và tuyến tính cũng như lập trình ràng buộc.
- Sau khi mô hình hóa vấn đề, sử dụng trình giải **CP-SAT Solver** để giải

Chi tiết cài đặt phương pháp ở trong file ./Code/CP.ipynb

Kết quả cài đặt được thống kế ở mục **III**

3. Integer Linear Programing

3.1 Mô hình hóa bài toán

3.1.1 Tập các biến

- **Lời giải:**

$$X = \{X_{i,j}\}$$

Với $X_{i,j} = 1$ nếu chu trình đi từ thành phố i đến thành phố j và $X_{i,j} = 0$ nếu chu trình không đi từ thành phố i đến thành phố j (với mọi i,j = 0,1,...N-1 và i khác j)

- **Thứ tự thăm các thành phố:**

$Y = \{Y_i\}$ với Y_i là thành phố i được thăm theo thứ tự Y_i

- **Thời gian bắt đầu giao hàng ở thành phố thứ i:**

$t = \{t_i\}$ với t_i là thời điểm bắt đầu giao hàng ở thành phố thứ i

3.1.2 Miền giá trị

- **Lời giải:**

$$D(X_{i,j}) = \{0, 1\}$$

- **Thứ tự thăm các thành phố :**

$$D(Y_i) = \{1, 2, \dots, N\}$$

- **Thời gian bắt đầu giao hàng ở thành phố thứ i :**

$$D(t_i) = R +$$

3.1.3 Ràng buộc

- **Mỗi thành phố sẽ có một đường đi vào và một đường đi ra:**

$$\sum_{i\in\{0,1,\dots,N-1\}\setminus\{j\}} X(i,j) = \sum_{i\in\{0,1,\dots,N-1\}\setminus\{j\}} X(j,i) = 1, \text{ với mọi } j = 0,1,\dots, N\text{-}1$$

- **Ràng buộc cấm tạo chu trình con (Miller – Tucker - Zemlin):**

$$y(j) - y(i) \geq 1 - N\big(1 - x(i,j)\big), \forall i \neq j, i \geq 1\ i,j \in \{0,\dots,N-1\}$$

- **Cửa sổ thời gian:**

$$e_i \leq t_i \leq l_i$$

- **Thời gian bắt đầu giao ở điểm thứ j sẽ phải muộn hơn thời gian bắt đầu giao ở điểm thứ i cộng thêm thời gian giao ở điểm i và quãng đường đi từ i đến j:**

$$t_i + d_i + c_{ij} - \big[t_j + M * (1 - x_{ij})\big] \leq 0 \text{ (M là một số rất lớn)}$$

3.1.4 Hàm mục tiêu

- **Minimize:** cost
  - $0 \leq \text{cost} \leq 9999999$
  - $\text{cost} \geq t[i] + d_i + c_{i0} - M(1 - x(i, 0))$ với mọi i thuộc [1, N]
  - M là một số rất lớn

3.2 Giải quyết bài toán

3.2.1 Khởi tạo các biến

```
x = {}
for i in range(data['N']):
    for j in range(data['N']):
        x[i, j] = solver.IntVar(0, 1, 'x[' + str(i)+','+str(j)+']')
y = {}
for i in range(data['N']):
    y[i] = solver.IntVar(1, data['N'], 'y[' + str(i) + ']')
```

```
t = {}
for i in range(data['N']):
    t[i] = solver.IntVar(0, 9999, 't[' + str(i) + ']')
```

3.2.2 Khai báo ràng buộc

```
# Mỗi Thành phố thăm một lần
for i in range(data['N']):
    cons1 = solver.Constraint(1, 1)
    for j in range(data['N']):
        cons1.SetCoefficient(x[i, j], 1)

for i in range(data['N']):
    cons1 = solver.Constraint(1, 1)
    for j in range(data['N']):
        cons1.SetCoefficient(x[j, i], 1)
```

```
# Không tạo chu trình con
for i in range(0, data['N']):
    for j in range(1, data['N']):
        if i == j:
            continue
        solver.Add(y[i] - y[j] + data['N']*x[i, j] <= data['N'] - 1)

# xii == 0
for i in range(data['N']):
    cons4 = solver.Constraint(0, 0)
    cons4.SetCoefficient(x[i,i], 1)
```

```
# Cửa sổ thời gian: e[i] <= t[i] <= l[i]
for i in range(data['N']):
    cons5 = solver.Constraint(e[i], l[i])
    cons5.SetCoefficient(t[i], 1)

"""Thời gian bắt đầu giao ở điểm thứ j sẽ phải muộn hơn thời gian bắt đầu giao ở điểm
thứ i cộng thêm thời gian giao ở điểm i và quãng đường đi từ i đến j:"""

for i in range(0, data['N']):
    for j in range(1, data['N']):
        if i == j:
            continue
        solver.Add(t[i] + data['cost'][i][j] + d[i] - (t[j] + 999999*(1 - x[i, j])) <= 0)
```

3.2.3 Khai báo hàm mục tiêu

```
#-----------------------------------------------------------------------------
# Hàm mục tiêu

cost = solver.IntVar(0, 999999999, 'cost')
for i in range(1, data['N']):
    solver.Add(cost >= t[i] + d[i] + data['cost'][i][0] - 999999999*(1-x[i, 0]))

solver.Minimize(cost)
```

3.3 Kết quả thực nghiệm và thời gian chạy

Tên bộ dữ liệu	Kết quả chạy	Thời gian chạy
N5	465	0.011s
N10	779	0.015s
N100	57815	0.609s
N200	101539	2.348s
N300	164756	5.391s
N500	267189	15.792s
N600	335883	21.716s
N700	368821	31.412s
N900	468147	50.658s
N1000	526522	66.320s

4. Thuật toán tham lam (Greedy)

- Mục tiêu của các chiến lược greedy là tổng thời gian di chuyển trên toàn bộ tuyến đường là nhỏ nhất

4.1 Các chiến lược tham lam

4.1.1 Chiến lược Nearest node first:

- Mô tả: Node có thời gian di chuyển gần nhất với node hiện tại sẽ được chọn làm node để đi tới tiếp theo
- Mã giả:

Mã giả:

```
def nearest_node_first:
  1) khởi tạo cur_time = 0, cur_node = 0, visited
  2) loop 1 ⇒ n-1
    + next_node = node gần nhất từ cur_node mà chưa được thăm
    + cập nhật thời gian cur_time = cur_time + thời gian (giao hàng + sang next_node)
    + if cur_time > L(next_node):
            return "Chiến lược greedy thất bại"
    + if cur_time < E(next_node):
            cur_time = E(next_node)
    + cập nhật node hiện tại cur_node = next_node
  3) return cur_time + thời gian (giao hàng + về node 0)
```

4.1.2 Chiến lược Nearest start-time first:

- Mô tả: Node có thời gian bắt đầu giao gần nhất với thời gian hiện tại sẽ được chọn làm node để đi tới tiếp theo
- Mã giả:

Mã giả:

```
def nearest_start_time_first:
  1) khởi tạo cur_time = 0
  2) sorted_nodes = sort(nodes(1 ⇒ n-1), start_time)
  3) loop next_node in sorted_nodes
    + cập nhật thời gian cur_time = cur_time + thời gian (giao hàng + sang next_node)
    + if cur_time > L(next_node):
            return "Chiến lược greedy thất bại"
    + if cur_time < E(next_node):
            cur_time = E(next_node)
  4) return cur_time + thời gian (giao hàng + về node 0)
```

4.1.2 Chiến lược Nearest end-time first:

- Mô tả: Node có thời gian kết thúc giao gần nhất với thời gian hiện tại sẽ được chọn làm node để đi tới tiếp theo
- Mã giả:

Mã giả:

```
def nearest_end_time_first:
  1) khởi tạo cur_time = 0
  2) sorted_nodes = sort(nodes(1 ⇒ n-1), end_time)
  3) loop next_node in sorted_nodes
    + cập nhật thời gian cur_time = cur_time + thời gian (giao hàng + sang end_node)
    + if cur_time > L(next_node):
          return "Chiến lược greedy thất bại"
    + if cur_time < E(next_node):
          cur_time = E(next_node)
  4) return cur_time + thời gian (giao hàng + về node 0)
```

4.2 Kết quả:

- Kết quả thực nghiệm và thời gian chạy trên bộ dữ liệu được cung cấp

	file	best_cost	path_result	time
0	N5.txt	None	Sai	0.000070
1	N10.txt	None	Sai	0.000015
2	N100.txt	None	Sai	0.000145
3	N200.txt	None	Sai	0.000124
4	N300.txt	None	Sai	0.000286
5	N500.txt	None	Sai	0.000213
6	N600.txt	None	Sai	0.000561
7	N700.txt	None	Sai	0.000280
8	N900.txt	None	Sai	0.000366
9	N1000.txt	None	Sai	0.000568

Nearest node first

	file	best_cost	path_result	time
0	N5.txt	465	Đúng	0.000366
1	N10.txt	779	Đúng	0.000019
2	N100.txt	57815	Đúng	0.000087
3	N200.txt	101539	Đúng	0.000171
4	N300.txt	164756	Đúng	0.000241
5	N500.txt	267189	Đúng	0.000358
6	N600.txt	335883	Đúng	0.000478
7	N700.txt	368821	Đúng	0.000610
8	N900.txt	468147	Đúng	0.000869
9	N1000.txt	526522	Đúng	0.000768

Nearest start-time first

	file	best_cost	path_result	time
0	N5.txt	465	Đúng	0.000025
1	N10.txt	779	Đúng	0.000010
2	N100.txt	57815	Đúng	0.000054
3	N200.txt	101539	Đúng	0.000142
4	N300.txt	164756	Đúng	0.000174
5	N500.txt	267189	Đúng	0.000310
6	N600.txt	335883	Đúng	0.000428
7	N700.txt	368821	Đúng	0.000623
8	N900.txt	468147	Đúng	0.000822
9	N1000.txt	526522	Đúng	0.000870

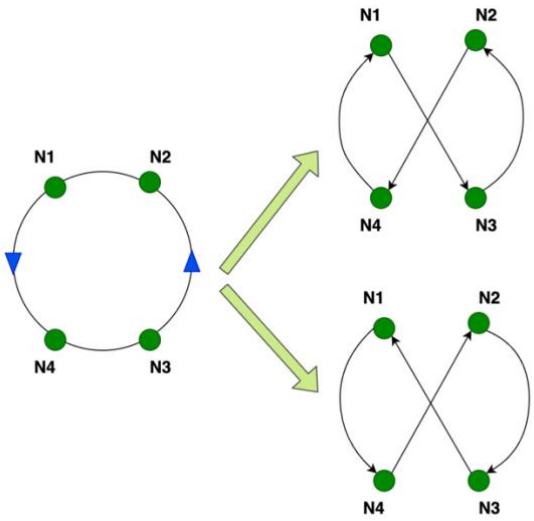
Nearest end-time first

5. Thuật toán Heuristic

5.1 Nguyên tắc cơ bản thuật toán 2-OPT:

- Thay thế hai cạnh không liền kề trong lời giải hiện tại bằng hai cạnh khác và kiểm tra điều kiện, tạo thành một lời giải mới có chi phí thấp hơn.
- Điều này được thực hiện liên tục cho đến khi không còn cải thiện nữa.
- Về lý thuyết số lượng chu trình mới tối đa có thể sinh ra là n(n-3) (O(n^2)) từ một nút khởi đầu

Hàm mục tiêu: tổng thời gian di chuyển nhỏ nhất



5.2 Các hàm quan trọng

- Hàm khởi tạo trạng thái ban đầu

+ Thực hiện vét cạn không gian đường đi đã shuffle để tạo tính ngẫu nhiên
+ Khi có đường đi thoả mãn điều kiện ràng buộc lấy đường đi đó làm giá trị khởi tạo

```python
def generate_routes(self, do_shuffle=True):
    cur_route, cur_cost, visited = [0], 0, [False] * (self.n+1)
    def Try(k):
        nonlocal cur_route, cur_cost
        loop = list(range(1, self.n+1))
        if do_shuffle: shuffle(loop)
        for i in loop:
            if visited[i]: continue
            added_time = self.time_matrix[cur_route[-1]][2] + self.dist_matrix[cur_route[-1]][i]
            if cur_cost + added_time > self.time_matrix[i][1]: continue
            if cur_cost + added_time < self.time_matrix[i][0]:
                added_time = self.time_matrix[i][0] - cur_cost

            cur_route.append(i)
            visited[i] = True
            cur_cost += added_time

            if k == self.n:
                yield list(cur_route)
            else:
                yield from Try(k+1)

            cur_route.pop()
            visited[i] = False
            cur_cost -= added_time

    yield from Try(1)
```

- Mã giả thuật toán Local search

```
def local_search (init_route, max_iteration, temperature):
    best_route = init_route
    best_cost = cal_cost(init_route)
    iteration = 0

    while True:
        iteration += 1
        if iteration > max_iteration: break

        route, cost = generate_2_opt_change()

        if cost < best_cost:
            update best_route, best_cost

        elif temperature not none and can_degrade(best_cost, cost, temperature):
            update best_route, best_cost
            route, cost  = generate_2_opt_change()
```

- Mã giả thuật toán Iterated local search

```
def iterated_local_search(max_iteration)
    best_route = None
    best_cost = 1e9
    iteration = 0

    for i in generate_routes():
        iteration += 1
        if iteration > max_iteration: break

        route, cost = local_search(init_route)
        if cost < best_cost:
            update route, cost

    return best_route, best_cost
```

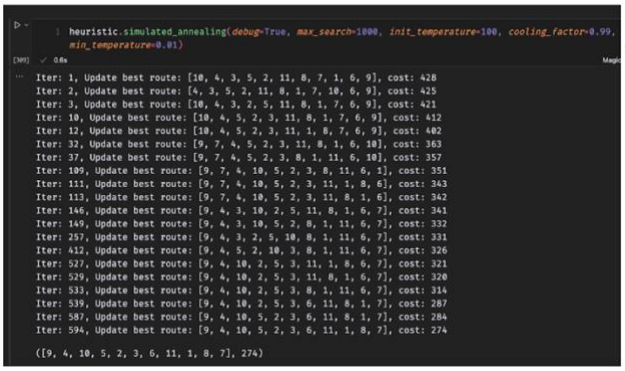
- Mã giả thuật toán Simulated annealing:

```
def simulated_annealing(max_iteration)
    best_route = None
    best_cost = 1e9
    iteration = 0
    temperature = INIT_TEMPERATURE (100)
    cooling_factor = INIT_COOLING_FACTOR (0.97)
    min_temperature = INIT_MIN_TEMPERATURE (0.01)

    for i in generate_routes():
        iteration += 1
        if iteration > max_iteration: break

        route, cost = local_search(init_route, temperature)
        if cost < best_cost:
            update route, cost

        temperature = max(temperature, min_temperature)
    return best_route, best_cost
```

5.3 Thử nghiệm trên bộ dữ liệu tự tạo

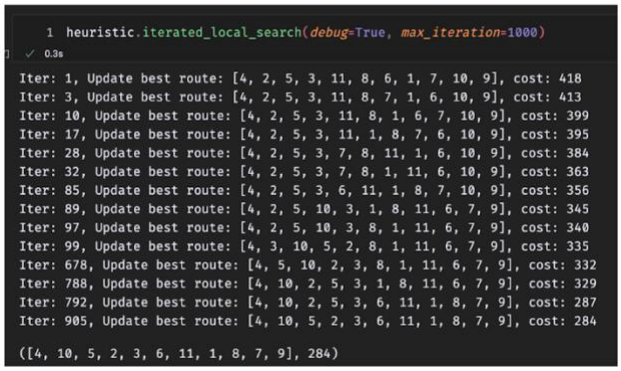
- Với bộ dữ liệu tự tạo

- Với các khởi tạo khác nhau, kết quả cuối có thể là local optimum hoặc global optimum
- Tăng số lượng vòng lặp làm tăng khả năng hội tụ của thuật toán



```
heuristic.simulated_annealing(debug=True, max_search=1000, init_temperature=100, cooling_factor=0.99, min_temperature=0.01)
```

```
Iter: 1, Update best route: [10, 4, 3, 5, 2, 11, 8, 7, 1, 6, 9], cost: 428
Iter: 2, Update best route: [4, 3, 5, 2, 11, 8, 1, 7, 10, 6, 9], cost: 425
Iter: 3, Update best route: [10, 4, 3, 2, 5, 11, 8, 1, 7, 6, 9], cost: 421
Iter: 10, Update best route: [10, 4, 5, 2, 3, 11, 8, 1, 7, 6, 9], cost: 412
Iter: 12, Update best route: [10, 4, 5, 2, 3, 11, 1, 8, 7, 6, 9], cost: 402
Iter: 32, Update best route: [9, 7, 4, 5, 2, 3, 11, 8, 1, 6, 10], cost: 363
Iter: 37, Update best route: [9, 7, 4, 5, 2, 3, 8, 1, 11, 6, 10], cost: 357
Iter: 109, Update best route: [9, 7, 4, 10, 5, 2, 3, 8, 11, 6, 1], cost: 351
Iter: 111, Update best route: [9, 7, 4, 10, 5, 2, 3, 11, 1, 8, 6], cost: 343
Iter: 113, Update best route: [9, 7, 4, 10, 5, 2, 3, 11, 8, 1, 6], cost: 342
Iter: 146, Update best route: [9, 4, 3, 10, 2, 5, 11, 8, 1, 6, 7], cost: 341
Iter: 149, Update best route: [9, 4, 3, 10, 5, 2, 8, 1, 11, 6, 7], cost: 332
Iter: 257, Update best route: [9, 4, 3, 2, 5, 10, 8, 1, 11, 6, 7], cost: 331
Iter: 412, Update best route: [9, 4, 5, 2, 10, 3, 8, 1, 11, 6, 7], cost: 326
Iter: 527, Update best route: [9, 4, 10, 2, 5, 3, 11, 1, 8, 6, 7], cost: 321
Iter: 529, Update best route: [9, 4, 10, 2, 5, 3, 11, 8, 1, 6, 7], cost: 320
Iter: 533, Update best route: [9, 4, 10, 2, 5, 3, 8, 1, 11, 6, 7], cost: 314
Iter: 539, Update best route: [9, 4, 10, 2, 5, 3, 6, 11, 8, 1, 7], cost: 287
Iter: 587, Update best route: [9, 4, 10, 5, 2, 3, 6, 11, 8, 1, 7], cost: 284
Iter: 594, Update best route: [9, 4, 10, 5, 2, 3, 6, 11, 1, 8, 7], cost: 274

([9, 4, 10, 5, 2, 3, 6, 11, 1, 8, 7], 274)
```



```
heuristic.iterated_local_search(debug=True, max_iteration=1000)
```

```
Iter: 1, Update best route: [4, 2, 5, 3, 11, 8, 6, 1, 7, 10, 9], cost: 418
Iter: 3, Update best route: [4, 2, 5, 3, 11, 8, 7, 1, 6, 10, 9], cost: 413
Iter: 10, Update best route: [4, 2, 5, 3, 11, 8, 1, 6, 7, 10, 9], cost: 399
Iter: 17, Update best route: [4, 2, 5, 3, 11, 1, 8, 7, 6, 10, 9], cost: 395
Iter: 28, Update best route: [4, 2, 5, 3, 7, 8, 11, 1, 6, 10, 9], cost: 384
Iter: 32, Update best route: [4, 2, 5, 3, 7, 8, 1, 11, 6, 10, 9], cost: 363
Iter: 85, Update best route: [4, 2, 5, 3, 6, 11, 1, 8, 7, 10, 9], cost: 356
Iter: 89, Update best route: [4, 2, 5, 10, 3, 1, 8, 11, 6, 7, 9], cost: 345
Iter: 97, Update best route: [4, 2, 5, 10, 3, 8, 1, 11, 6, 7, 9], cost: 340
Iter: 99, Update best route: [4, 3, 10, 5, 2, 8, 1, 11, 6, 7, 9], cost: 335
Iter: 678, Update best route: [4, 5, 10, 2, 3, 8, 1, 11, 6, 7, 9], cost: 332
Iter: 788, Update best route: [4, 10, 2, 5, 3, 1, 8, 11, 6, 7, 9], cost: 329
Iter: 792, Update best route: [4, 10, 2, 5, 3, 6, 11, 1, 8, 7, 9], cost: 287
Iter: 905, Update best route: [4, 10, 5, 2, 3, 6, 11, 1, 8, 7, 9], cost: 284

([4, 10, 5, 2, 3, 6, 11, 1, 8, 7, 9], 284)
```

- Với bộ dữ liệu được cung cấp:

	file	cost	time
0	N5.txt	465	0.000372
1	N10.txt	779	0.006454

Simulated annealing

	file	cost	time
0	N5.txt	465	0.000202
1	N10.txt	779	0.006258

Iterated local search

III. Kết quả thực nghiệm và nhận xét

Kiểm tra các phương pháp giải bằng bộ dữ liệu được cung cấp bởi lớp học, các test case có khoảng giá trị từ $[\mathbf{5}, \mathbf{1000}]$, phù hợp để kiểm tra tính hiệu quả của từng phương pháp.

Bảng sau đây là kết quả thu được sau khi chạy từng thuật toán, kết quả được biểu diễn bởi lời giải thu được, thời gian chạy chương trình (đơn vị giây)

n	Kết quả gốc	Brute Force		CP		ILP		Metaheuristic			
								Iterated local search		Simulated annealing	
		Result	Time	Result	Time	Result	Time	Result	Time	Result	Time
5	465	465	0.001	465	0.01	465	0.01	465	0.0003	465	0.0002
10	779	779	0.003	779	0.04	779	0.02	779	0.006	779	0.006
100	57815	-	-	57815	3.71	57815	0.61	-	-	-	-
200	101539	-	-	101539	10.63	101539	2.35	-	-	-	-
300	164756	-	-	164756	28.45	164756	5.39	-	-	-	-
500	267189	-	-	267189	86.78	267189	15.79	-	-	-	-
600	335883	-	-	335883	105.06	335883	21.18	-	-	-	-
700	368821	-	-	368821	144.68	368821	31.41	-	-	-	-
900	468147	-	-	468147	233.16	468147	50.66	-	-	-	-
1000	526522	-	-	526522	297.76	526522	66.32	-	-	-	-
Pass		2/10		10/10		10/10		2/10		2/10	

- Các thuật toán tham lam

n	Kết quả gốc	Neerest node first		Nearest start-time first		Nearest end-time first	
		Result	Time	Result	Time	Result	Time
5	465	-	-	465	Không đáng kể	465	Không đáng kể
10	779	-	-	779	Không đáng kể	779	Không đáng kể
100	57815	-	-	57815	Không đáng kể	57815	Không đáng kể
200	101539	-	-	101539	Không đáng kể	101539	Không đáng kể
300	164756	-	-	164756	Không đáng kể	164756	Không đáng kể
500	267189	-	-	267189	Không đáng kể	267189	Không đáng kể
600	335883	-	-	335883	Không đáng kể	335883	Không đáng kể
700	368821	-	-	368821	Không đáng kể	368821	Không đáng kể
900	468147	-	-	468147	Không đáng kể	468147	Không đáng kể
1000	526522	-	-	526522	Không đáng kể	526522	Không đáng kể
Pass		0/10		10/10		10/10	

Nhận xét:

- **Brute Force**: rất dễ để cài đặt nhưng thời gian chạy khá chậm, với bộ dữ liệu nhiều đỉnh thì thời gian chạy sẽ vô cùng lớn
- **CP** : Cách mô hình bài toán dễ hiểu, áp dụng được vào được các công cụ giải bài toán tối ưu ràng buộc. Vượt qua hết các test case với thời gian nhanh, với N = 1000 thời gian chạy gần 5 phút, khả thi với bài toán lập lịch trong thực tế
- **ILP** : chạy rất nhanh, tuy nhiên phần định nghĩa mô hình hóa bài toán khá là khó. Đối với những bộ test có nhiều lời giải thỏa mãn điều kiện, ILP tính toán rất chậm
- **Greedy:** Các thuật toán greedy thường đơn giản có thời gian tính toán thấp không đáng kể. Tuy nhiên chỉ cho đáp án khi giả định ban đầu là đúng. Cần tiếp tục tìm phương án cải tiến và đưa ra chiến lược bao quát hơn
- **Heuristic**: Thời gian chạy là trong khoảng chấp nhận được khi thử trên bộ dữ liệu bé hơn 20. Cần cải thiện phương thức khởi tạo thay vì vét cạn và tăng cường thêm các thuật toán metaheuristic khác: Guided local search, Tabu search, Variable neighborhood search