ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

PLANNING OPTIMIZATION

Order Picking up route in Warehouse

ONE LOVE. ONE FUTURE.

Phân công nhóm 8

Họ và tên	MSSV	Công việc
Vũ Trường An	20220058	Local search: Hill Climbing, Simulated Annealing, Tabu Search Sinh test
Trương Xuân Thông	20220044	Problem Modeling Integer Programming, Constraint Programming
Lê Danh Vinh	20220051	Backtracking, Branch and Bound, Beam Search
Nguyễn Đăng Phúc	20220040	Greedy, Genetic Algorithm

Mục lục

HUST

hust.edu.vn fb.com/dhbkhn

1. Bài toán

1. Bài toán

Mô tả bài toán

Có M kệ hàng trong một kho lớn, được đánh số $1, 2, \ldots, M$. Kệ hàng j được đặt tại một vị trí xác định trong kho $(j = 1, 2, \ldots, M)$

Có N loại sản phẩm, được đánh số $1, 2, \ldots, N$. Số lượng sản phẩm loại i có tại kệ j được ký hiệu là Q_{ij} .

Nhân viên kho bắt đầu từ cửa kho (điểm 0), cần ghé thăm một tập con các kệ hàng (mỗi kệ được ghé thăm nhiều nhất một lần) và sau đó quay trở lại cửa kho để lấy đủ sản phẩm cho đơn hàng của khách. Tổng số lượng sản phẩm loại i cần lấy là q_i $(i = 1,2, \ldots, N)$

Khoảng cách di chuyển từ kệ i đến kệ j là $d_{ij} (0 \leq i, j \leq M)$

Mục tiêu là tìm thứ tự các kệ cần ghé thăm sao cho tổng quãng đường di chuyển là nhỏ nhất, đồng thời đảm bảo lấy đủ số lượng sản phẩm yêu cầu.

Input:

- **Dòng 1:** Hai số nguyên N, M – số loại sản phẩm và kệ hàng
- N **dòng tiếp theo:** Ma trận Q, trong đó Q_{ij} là số lượng sản phẩm loại i tại kệ j
- $M + 1$ **dòng tiếp theo:** Ma trận khoảng cách d, với d_{ij} là khoảng cách từ kệ i đến kệ j (0 là cửa kho)
- **Dòng cuối:** Vector q trong đó q_i là số lượng cần lấy của sản phẩm loại i

Output:

- **Dòng 1:** Số kệ hàng được ghé thăm
- **Dòng 2:** Dãy n kệ hàng $x_1, x_2, \ldots, x_n$ (không trùng nhau), biểu diễn thứ tự các kệ ghé thăm

HUST

hust.edu.vn fb.com/dhbkhn

2. Các thuật toán đề xuất

2a. Backtracking

Backtracking

- Backtracking là kỹ thuật vét cạn cơ bản nhất cho nhiều bài toán tối ưu
- Duyệt toàn bộ các cấu hình và chọn nghiệm tốt nhất

Ưu điểm

- Luôn tìm được nghiệm tối ưu (nếu tồn tại)

Nhược điểm

- Chỉ hoạt động khi m nhỏ do độ phức tạp xấp xỉ $O(m!)$

Cài đặt trong bài toán

- Xây dựng dần thứ tự các kệ được ghé thăm
- Ở mỗi bước, chọn một kệ chưa được thăm và tiếp tục mở rộng lời giải
- Cập nhật tổng quãng đường di chuyển và số lượng sản phẩm đã lấy
- Khi thỏa mãn yêu cầu, cập nhật nghiệm tốt nhất

2b. Branch and Bound

Branch and Bound

- Cải tiến của Backtracking bằng cách cắt bỏ các nhánh không tiềm năng
- Sử dụng cận dưới (lower bound), ước lượng quãng đường tối thiểu còn phải đi xét để loại bỏ các nhánh không thể cho nghiệm tốt hơn
- Chỉ tiếp tục mở rộng các nhánh còn khả năng cải thiện nghiệm hiện tại

Ưu điểm:

- Giảm đáng kể không gian tìm kiếm
- Vẫn đảm bảo tìm nghiệm tối ưu

Nhược điểm:

- Hiệu quả phụ thuộc chất lượng của bound
- Trường hợp xấu vẫn có thể có độ phức tạp rất lớn, xấp xỉ $O(m!)$

2b. Branch and Bound

Ý tưởng:

- Upper bound: nghiệm ban đầu (được khởi tạo bằng Greedy)
- Lower bound: ước lượng chi phí tối thiểu còn lại
- Nếu cận dưới ≥ nghiệm tốt nhất hiện tại → loại nhánh

Cách ước lượng số 1: sử dụng khoảng cách nhỏ nhất giữa hai kệ

- Tổng quãng đường đã đi là total_dist
- Số kệ đã đi qua trong lời giải hiện tại là là k
- Khoảng cách nhỏ nhất giữa hai kệ là c_{min}
- Ước lượng số lượng kệ còn phải đi (trong trường hợp xấu nhất) là $m - k$
- Khi đó ước lượng cận dưới của lời giải hiện tại sẽ lại

$$LB = \text{total\_dist} + c_{min} \times (m - k)$$

2b. Branch and Bound

Cách ước lượng số 2: Sử dụng cây khung nhỏ nhất

- Tổng khoảng cách qua các kệ đã đi là total_dist
- Gọi tập các kệ đã đi qua trong lời giải hiện tại là visited
- Tập các kệ chưa đi qua là unvisited
- Ước lượng khoảng cách tối thiểu còn cần phải đi là $\text{MST}(\text{unvisited} \cup \{0, \text{current\_node}\})$ (cây khung nhỏ nhất của tất các điểm còn phải đi trong trường hợp xấu nhất)
- Ước lượng cận dưới của lời giải hiện tại:

$$\text{LB} = \text{total\_dist} + \text{MST}(\text{unvisited} \cup \{0, \text{current\_node}\})$$

2c. Integer Programming - Mô hình hoá bài toán

1. Biến quyết định

- $\mathbf{x}[\mathbf{i}][\mathbf{j}] \in \{0, 1\}$: Bằng 1 nếu di chuyển từ điểm i đến điểm j, bằng 0 nếu ngược lại $(\mathrm{i}, \mathrm{j} = 0, \dots, M)$.
- $\mathbf{y}[\mathbf{j}] \in \{0, 1\}$: Bằng 1 nếu ghé thăm kệ j, bằng 0 nếu ngược lại $(\mathrm{j} = 1, 2, 3, \dots, M)$.
- $\mathbf{u}[\mathbf{j}]$ là số nguyên $(1,2,3, \dots, M)$: Thứ tự ghé thăm của kệ j trong lộ trình.

2. Hàm mục tiêu

- Tối thiểu hóa tổng quãng đường di chuyển

$$\textbf{Minimize}: \sum_{\mathrm{i} \neq \mathrm{j}} (\mathrm{d}[\mathrm{i}][\mathrm{j}] \times \mathrm{x}[\mathrm{i}][\mathrm{j}])$$

2c. Integer Programming - Mô hình hoá bài toán

3. Các ràng buộc chính

- **Thỏa mãn nhu cầu:** $\Sigma_j (Q[i][j] * y[j]) \geq q[i]$, với mọi sản phẩm i.
- **Bảo toàn luồng:**
  a. $\Sigma_i x[i][j] = y[j]$
  b. $\Sigma_i x[j][i] = y[j]$
- **Ràng buộc cửa kho:**
  a. $\Sigma_j x[0][j] = 1$
  b. $\Sigma_i x[i][0] = 1$
- **Khử chu trình con (MTZ Constraints):** $u[i] - u[j] + M * x[i][j] \leq M - 1$ (với $i, j \neq 0$ và $i \neq j$)

2c. Integer Programming

Algorithm 4 Integer Programming Solution using OR-Tools

```
function SolveWithIP(n, m, Q, d, q)
    Initialize SCIP solver from OR-Tools library
    Define decision variables
    Add constraints (as defined in model)
    Set objective function
    status ← Solve
    if status = OPTIMAL or status = FEASIBLE then
        Reconstruct route by following x_ij = 1 edges starting from depot
        return extracted route and objective value
    end if
end function
```

2d. Constraint Programming

1. Biến quyết định

- $\mathbf{y}[\mathbf{j}] \in \{0, 1\}$: Trạng thái chọn kệ (1: Chọn, 0: Bỏ qua).
- $\mathbf{x}[\mathbf{i}][\mathbf{j}] \in \{0, 1\}$: Cung đường nối điểm i và j.

2. Các ràng buộc

- **Logic kéo theo :** $x[i][j] = 1 => (y[i] = 1, y[j] = 1)$
- **Logic tự lặp :** $x[j][j] = 1 <=> y[j] = 0$
- **Ràng buộc toàn cục (Global Constraint):** $\text{Circuit}(\{x[i][j]\})$:
  a. Mọi nút có trong danh sách phải có đúng 1 cung đi vào và 1 cung đi ra.
  b. Tất cả các cung được chọn $(x[i][j] = 1)$ phải kết nối với nhau để tạo thành một chu trình khép kín duy nhất.
  c. Không cho phép chu trình con

2d. Constraint Programming

Algorithm 5 Constraint Programming Solution using CP-SAT

function SOLVEWITHCPSAT(n, m, Q, d, q)
  Initialize **CP-SAT model** and **Solver** from OR-Tools
  Define decision variables:
    $y_j \in \{0,1\}$, $x_{ij} \in \{0,1\}$
    $loop_j \in \{0,1\}$
  Add Logic Constraints:
    $\sum(Q_{ij} \cdot y_j) \geq q_i$
    $x_{jj} = 1 \iff y_j = 0$
    $x_{ij} = 1 \implies (y_i = 1 \wedge y_j = 1)$
  Add Global Constraint:
    ADDCIRCUIT(all arcs x_{ij} including self-loops)
  Set Objective: Minimize $\sum(d_{ij} \cdot x_{ij})$
  $status \leftarrow$ SOLVE(model)
  if $status$ = OPTIMAL **or** $status$ = FEASIBLE **then**
    Reconstruct route by following $x_{ij} = 1$ edges starting from depot
    return extracted route and objective value
  end if
end function

2e. Greedy - Chọn theo khoảng cách

- Thuật toán tham lam chọn kệ gần nhất chứa hàng còn thiếu tại mỗi bước, dừng lại khi đủ số lượng sản phẩm yêu cầu

Chiến lược:

- Bắt đầu từ cửa kho (điểm 0)
- Mỗi bước chọn kệ chưa thăm, có hàng đang thiếu có khoảng cách nhỏ nhất
- Cập nhật lượng sản phẩm đã lấy

Ưu điểm:

- Cài đặt đơn giản, chạy rất nhanh $O(m^2 \times n)$
- Tạo nghiệm ban đầu cho các thuật toán khác như Branch and Bound, Local Search, Beam Search

Nhược điểm:

- Không đảm bảo nghiệm tối ưu
- Quyết định cục bộ có thể dẫn đến lời giải kém

2e. Greedy - Chọn theo score

- Thay vì chọn kệ gần nhất, mỗi kệ được gán một score
- Ưu tiên kệ mang lại nhiều sản phẩm cần thiết trên mỗi đơn vị quãng đường

Score:

- value được tính bằng tổng số sản phẩm còn thiếu có thể lấy tại kệ

$$\text{score} = \frac{\text{value}}{\text{khoảng cách di chuyển}}$$

2f. Beam Search

Beam Search

- Cải tiến so với Greedy bằng cách duy trì nhiều trạng thái tiềm năng song song
- Thay vì chỉ chọn một bước đi tốt nhất, giữ lại B lời giải tốt nhất tại mỗi mức mở rộng
- Cho phép tìm được nghiệm tốt hơn Greedy trong không gian tìm kiếm lớn

Cài đặt cho bài toán

- Khởi tạo lời giải từ cửa kho.
- Ở mỗi bước, mở rộng tất cả các lời giải hiện tại bằng cách thêm B kệ chưa được ghé thăm.
- Đánh giá các lời giải mở rộng và chỉ giữ lại B lời giải tốt nhất dựa trên tổng khoảng cách đã đi.
- Lặp lại cho đến khi thỏa mãn yêu cầu sản phẩm hoặc đạt độ sâu tối đa (độ sâu tối đa D được giới hạn dựa theo lời giải được khởi tạo bằng Greedy).

Ưu điểm:

- Cài đặt tương đối đơn giản
- Độ phức tạp thấp, $O(m^2 \times \log(m) + m^2 \times n)$

Nhược điểm:

- Không đảm bảo nghiệm tối ưu
- Chất lượng phụ thuộc vào cách chọn B và hàm đánh giá

2g. Local Search

- Thuật toán cải tiến lặp đi lặp lại một nghiệm duy nhất (iterative improvement)
- Bắt đầu từ một nghiệm ban đầu và tìm nghiệm tốt hơn trong lân cận

Ý tưởng:

- Khởi tạo nghiệm (Random, Greedy)
- Lặp: tìm nghiệm lân cận tốt hơn
- Dừng khi không còn cải thiện

Đặc điểm:

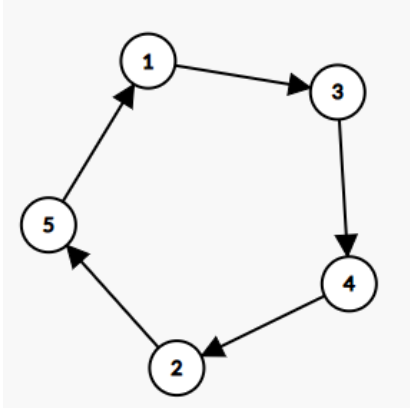
- Không duyệt toàn bộ không gian nghiệm
- Thường cho nghiệm ổn trong thời gian ngắn

2g. Local Search - Operators

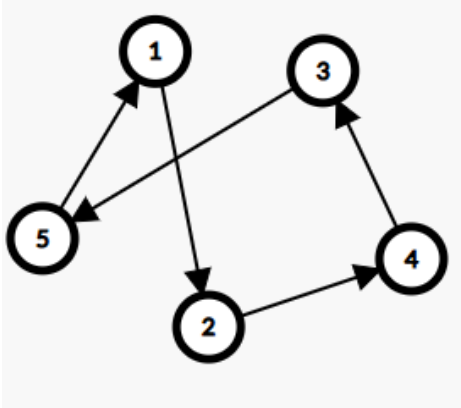
- Mỗi operator tạo ra một nghiệm lân cận

Các operator sử dụng:

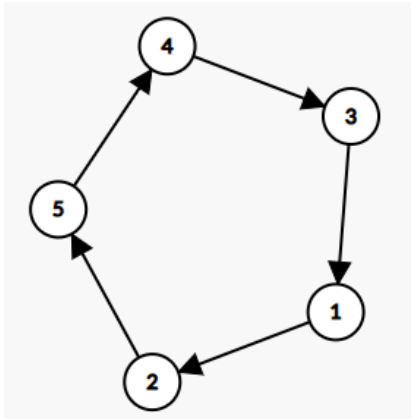
- 2-opt
- Swap
- Remove
- Add



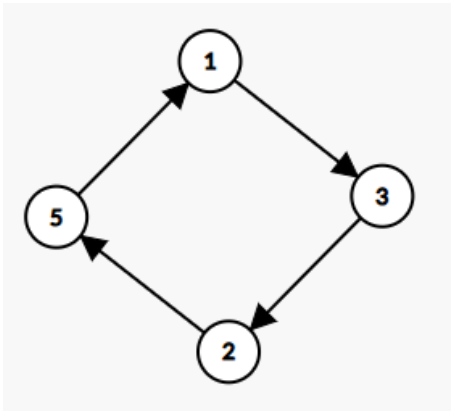
Nghiệm ban đầu



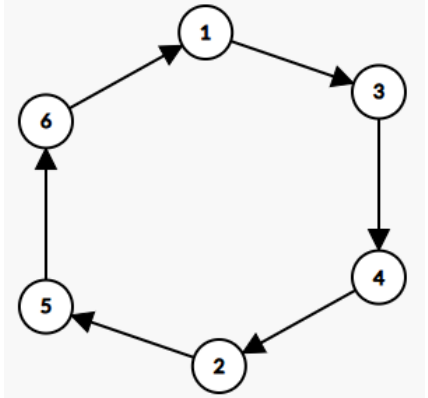
2-opt((1,3),(2,5))



Swap(1,4)



Remove(4)



Add(6, 5)

2g. Local Search - Hill Climbing

- Chiến lược chỉ chấp nhận nghiệm tốt hơn

Quy trình:

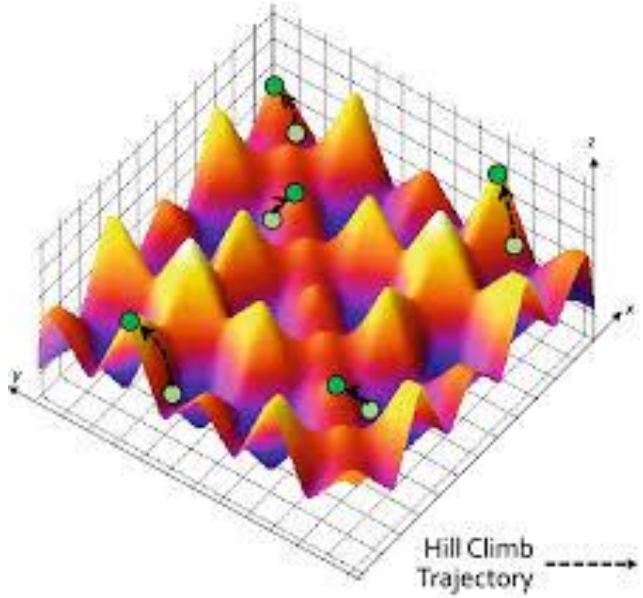
- Ngẫu nhiên thực hiện các operator có ưu tiên
  - 2-opt (55%), swap (15%), remove (15%), add (15%)
- Nếu tìm được nghiệm tốt hơn → chấp nhận ngay
- Lặp đến khi không còn cải thiện

Ưu điểm:

- Đơn giản, hội tụ nhanh, dễ cài đặt

Nhược điểm:

- Dễ mắc kẹt tại cực trị địa phương
- Phụ thuộc nghiệm ban đầu



2g. Local Search - Simulated Annealing

- Mở rộng của Hill Climbing
- Cho phép chấp nhận nghiệm kém hơn với một xác suất nhất định

Ý tưởng:

- Bắt đầu với nhiệt độ (T) cao
- Dần giảm nhiệt độ theo thời gian $T_{(k+1)} = \alpha \times T_k$
- Khi $T \rightarrow 0$, thuật toán trở về Hill Climbing

Acceptance rule:

- Nếu nghiệm mới tốt hơn → chấp nhận
- Nếu kém hơn → chấp nhận với xác suất $f(x) = \begin{cases} 1, & \text{if } \Delta \leq 0 \\ e^{-\frac{\Delta}{T}}, & \text{if } \Delta > 0 \end{cases}$
  - $\boldsymbol{\Delta}$: độ tăng của hàm mục tiêu

2g. Local Search - Tabu Search

- Sử dụng cấu trúc bộ nhớ Tabu List để tránh quay lại các giải pháp đã được khám phá trước đó
- Ngăn chặn việc quay trở lại các điểm tối ưu cục bộ và khuyến khích khám phá các khu vực mới.

Tabu list:

- Lưu các bước đi gần đây (2-opt, swap, add, remove)
- Ngăn quay lại nghiệm cũ → tránh kẹt local optimum
- Kích thước động $\tau = \max\left(7, \left\lfloor \frac{|\text{route}|}{2} \right\rfloor\right)$

Aspiration Criterion:

- Bỏ qua tabu nếu nghiệm mới tốt hơn nghiệm tốt nhất toàn cục

Diversification

- Nếu không cải thiện sau 500 bước:
  - Thực hiện hoán đổi ngẫu nhiên một phần lộ trình
  - Xóa tabu list → khám phá vùng nghiệm mới

2h. Genetic Algorithm

- Dựa trên framework Biased Random-Key Genetic Algorithm được đề xuất bởi Gonçalves và Resende
- Kết hợp giữa GA và Local search
- Ý tưởng chính
  - Mã hóa: Mỗi cá thể trong quần thể được đại diện bởi một vectơ các số thực ngẫu nhiên (keys) trong khoảng $[0, 1]$.
  - Giải mã: để chuyển đổi vectơ số thực thành một lộ trình khả thi, thuật toán thực hiện các bước:
    - Sắp xếp các kệ hàng dựa trên giá trị của các key (key nhỏ hơn có độ ưu tiên cao hơn).
    - Thuật toán duyệt qua danh sách các kệ đã sắp xếp và lần lượt thêm chúng vào lộ trình bằng cách chọn vị trí tốt nhất đến khi đủ hàng
    - Cắt tỉa các kệ hàng thừa
    - Tìm kiếm cục bộ để tìm lời giải tối ưu (chỉ áp dụng tại thế hệ cuối cùng)

2h. Genetic Algorithm (tiếp)

- Cơ chế tiến hóa:
  - Quần thể được chia thành nhóm **Elite** và nhóm **Non-Elite**.
  - Nhóm Elite được sao chép nguyên vẹn sang thế hệ sau.
  - Đột biến: Một phần của thế hệ mới được sinh ra hoàn toàn ngẫu nhiên để đảm bảo tính đa dạng cho quần thể trong quá trình lai ghép
  - Lai ghép: Lai ghép từ 1 cá thể cha Elite, 1 cá thể mẹ (Elite hoặc không Elite), trong đó xác suất lai ghép từ cha là 70%

HUST

hust.edu.vn fb.com/dhbkhn

3. Kết quả thực nghiệm

3. Kết quả thực nghiệm - Chiến lược sinh test

- **Mô hình không gian**
  - Mô phỏng kho hàng trên mặt phẳng 2D
  - Cửa kho tại điểm $(0, 0)$
  - Mỗi kệ được gán tọa độ ngẫu nhiên
- **Khoảng cách di chuyển**
  - Ma trận d được sinh bởi khoảng cách giữa 2 điểm $d(i,j) = \text{round}\left(\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}\right)$
- **Ma trận tồn kho thưa**
  - Ma trận Q được sinh ngẫu nhiên với 40% giá trị 0
  - Khó khăn cho Greedy, heuristic
- **Nhu cầu khách hàng**
  - Tổng cung sản phẩm $S_i = \sum_{j=1}^{M} Q[i][j]$
  - Nhu cầu sinh ngẫu nhiên $q_i \in [1, \alpha \times S_i]$
  - Đảm bảo luôn tồn tại nghiệm khả thi

3. Kết quả thực nghiệm

Algorithm	Total distance	Runtime (s)
Backtracking	**2865**	0.65
Greedy	3002	0.44
Branch And Bound	2865	0.53
Beam Search	3002	0.42
Integer Programming (SCIP)	2865	0.23
Constraint Programming	2865	0.62
Hill Climbing	2865	0.42
Simulated Annealing	2865	2
Tabu Search	2865	2
GA	2865	0.67

$N = 5, M = 9$

Algorithm	Total distance	Runtime (s)
Backtracking	**2263**	11.85
Greedy	2916	0.59
Branch And Bound	2263	0.51
Beam Search	2916	0.60
Integer Programming (SCIP)	2263	0.80
Constraint Programming	2263	0.64
Hill Climbing	2315	0.53
Simulated Annealing	2263	2
Tabu Search	2263	2
GA	2263	0.67

$N = 7, M = 12$

Algorithm	Total distance	Runtime (s)
Backtracking	-	TLE
Greedy	2900	0.53
Branch And Bound	**2188**	0.51
Beam Search	2261	0.50
Integer Programming (SCIP)	2188	0.88
Constraint Programming	2188	0.65
Hill Climbing	2346	0.54
Simulated Annealing	2188	2.53
Tabu Search	2188	2.57
GA	2188	0.85

$N = 7, M = 15$

Algorithm	Total distance	Runtime (s)
Backtracking	-	TLE
Greedy	3615	0.65
Branch And Bound	**2611**	10.33
Beam Search	3095	0.82
Integer Programming (SCIP)	2611	26.80
Constraint Programming	2611	1.05
Hill Climbing	2792	0.52
Simulated Annealing	2611	2.43
Tabu Search	2611	2.54
GA	2611	0.90

$N = 8, M = 18$

3. Kết quả thực nghiệm

Algorithm	Total distance	Runtime (s)
Backtracking	-	TLE
Greedy	5512	0.71
Branch And Bound	-	TLE
Beam Search	5340	0.57
Integer Programming (SCIP)	-	TLE
Constraint Programming	**3521**	17.09
Hill Climbing	4553	0.54
Simulated Annealing	3840	10.56
Tabu Search	3521	10.69
GA	3521	2.73

$N = 7, M = 47$

Algorithm	Total distance	Runtime (s)
Backtracking	-	TLE
Greedy	12719	1.05
Branch And Bound	-	TLE
Beam Search	12015	0.42
Integer Programming (SCIP)	-	TLE
Constraint Programming	-	TLE
Hill Climbing	10206	0.96
Simulated Annealing	8820	80.61
Tabu Search	8640	80.67
GA	**8244**	60.84

$N = 15, M = 424$

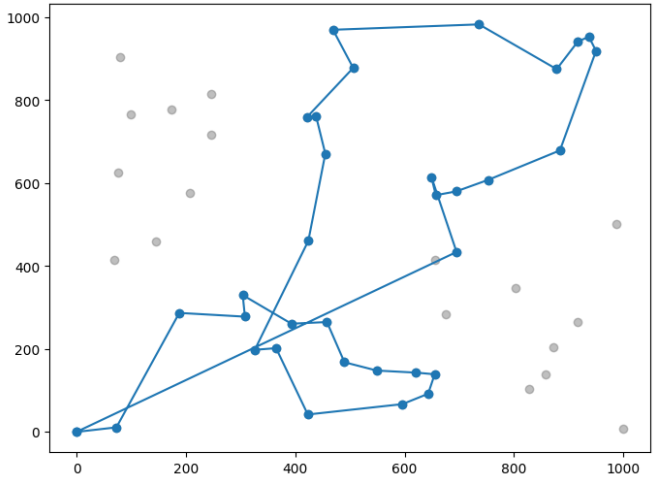
Algorithm	Total distance	Runtime (s)
Backtracking	-	TLE
Greedy	7222	0.64
Branch And Bound	-	TLE
Beam Search	6271	0.50
Integer Programming (SCIP)	-	TLE
Constraint Programming	-	TLE
Hill Climbing	5350	0.54
Simulated Annealing	4097	40.51
Tabu Search	3741	40.48
GA	**3725**	10.45

$N = 10, M = 100$

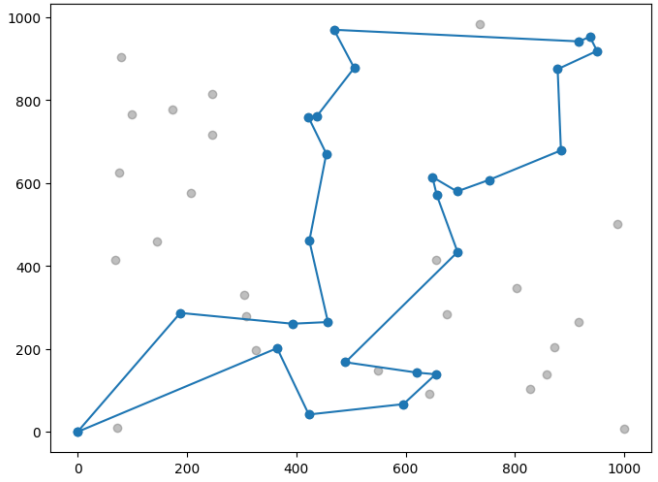
Algorithm	Total distance	Runtime (s)
Backtracking	-	TLE
Greedy	21375	0.67
Branch And Bound	-	TLE
Beam Search	21375	0.77
Integer Programming (SCIP)	-	TLE
Constraint Programming	-	TLE
Hill Climbing	17663	31.64
Simulated Annealing	16940	121.25
Tabu Search	16596	122.80
GA	**15291**	124.94

$N = 20, M = 1000$

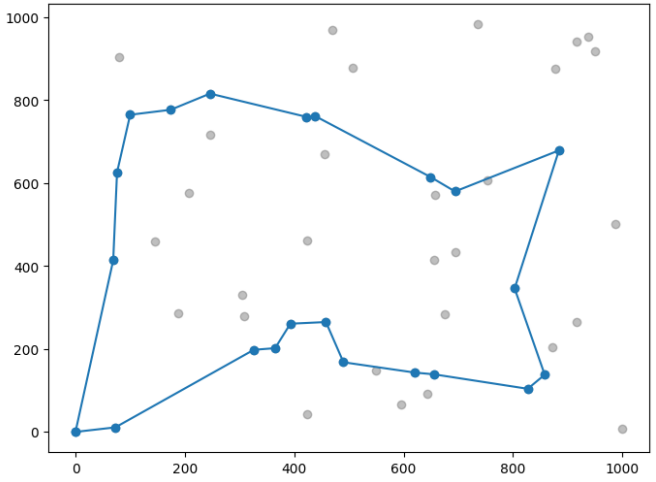
3. Kết quả thực nghiệm



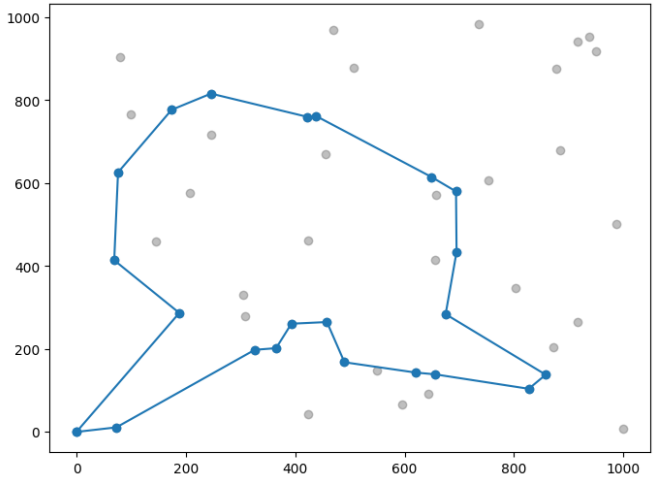
Greedy - 4459



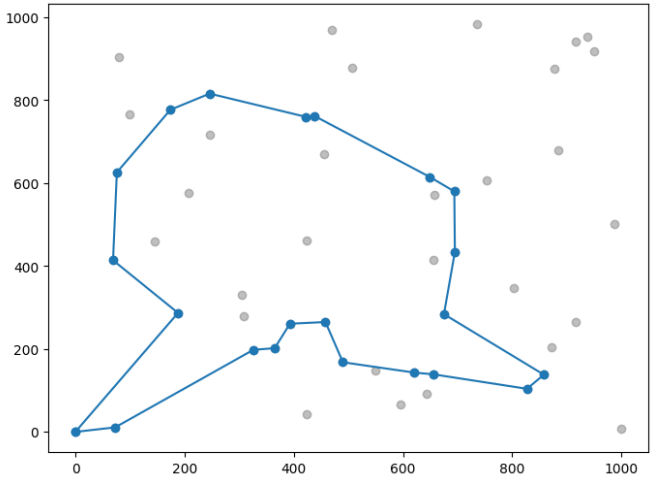
Hill Climbing - 3974



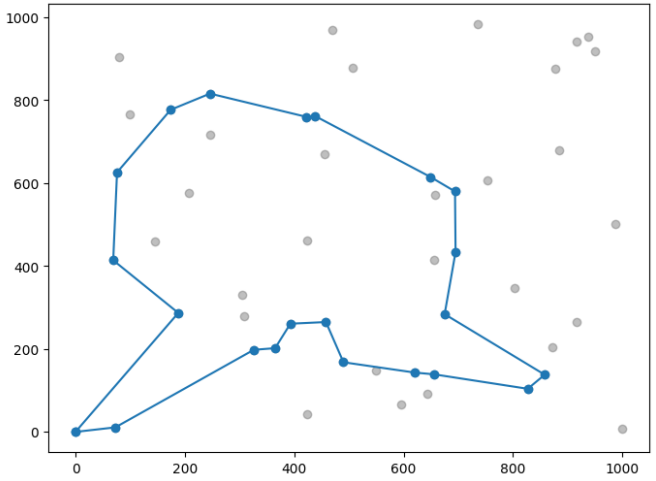
Simulated Annealing - 3266



Tabu Search - 3086



Genetic Algorithm - 3086



Constraint Programming - 3086

HUST

hust.edu.vn fb.com/dhbkhn

4. Kết luận

4. Kết luận

- Ở các bài toán quy mô nhỏ $(M \leq 13)$, các thuật toán chính xác hoạt động hiệu quả, các thuật toán heuristic cũng dễ dàng hội tụ ở kích thước này
- Ở quy mô trung bình $(15 \leq M < 50)$, Constraint Programming vẫn hoạt động tốt,các thuật toán greedy cho thấy chất lượng giảm rõ rệt, các thuật toán như local search, GA vẫn hội tụ tốt.
- Ở quy mô lớn $(100 \leq M \leq 1000)$, thuật toán Tabu Search và GA cho chất lượng tốt nhất.

HUST

hust.edu.vn fb.com/dhbkhn

THANK YOU !