TRƯỜNG ĐẠI HỌC SÀI GÒN
KHOA CÔNG NGHỆ THÔNG TIN
---o0o---

BÁO CÁO BÀI TẬP NHÓM

LAB - 03

Giảng viên hướng dẫn: TS. Đỗ Như Tài

Môn học: Trí tuệ nhân tạo nâng cao

Nhóm thực hiện: 7

Danh sách thành viên:

3122410489 – Lê Huỳnh Trúc Vy

3122410495 – Trần Mỹ Yên

3120410470 – Lê Quốc Thái

3122410174 – Thái Minh Khang

TP. HỒ CHÍ MINH, THÁNG 10 NĂM 2025

MỤC LỤC

BẢNG PHÂN CÔNG

MSSV	Họ tên	Công việc
3122410489	Lê Huỳnh Trúc Vy (Trưởng nhóm)	- Solving the n-Queens Problem using Local Search: Task 1, Task 5, Advanced Task. - Solving the Traveling Salesman Problem using Local Search: Stochastic Hill Climbing, Simulated Annealing. - Solving the Traveling Salesman Problem using Local Search 2: Stochastic Hill Climbing
3122410495	Trần Mỹ Yên	- Solving the n-Queens Problem using Local Search: Task 2, Task 6: + Comparison, More things to do - Solving the Traveling Salesman Problem using Local Search: Compare Performance - Solving the Traveling Salesman Problem using Local Search 2: Compare Performance
3120410470	Lê Quốc Thái	- Solving the n-Queens Problem using Local Search: Task 4, Task 6: + Problem Size Scalability. - Solving the Traveling Salesman Problem using Local Search: Steepest-ascend Hill Climbing Search with Random Restarts, First-choice Hill Climbing. - Solving the Traveling Salesman Problem using Local Search 2: Steepest-ascend Hill Climbing Search with Random Restarts

3122410174	Thái Minh Khang	- Solving the n-Queens Problem using Local Search: Task 3, Task 6: + Algorithm Convergence. - Solving the Traveling Salesman Problem using Local Search: Steepest-ascend Hill Climbing Search, Bonus: Genetic Algorithm. - Solving the Traveling Salesman Problem using Local Search 2: Steepest-ascend Hill Climbing Search, Bonus: Genetic Algorithm.

Solving the n-Queens Problem using Local Search

(Giải bài toán N-Queens bằng Local Search)

I. Bài toán N-Queens

- Mục tiêu: Tìm một cách sắp xếp n quân hậu trên bàn cờ n x n sao cho không có quân hậu nào nằm cùng hàng, cùng cột hoặc cùng đường chéo với quân hậu khác.

- Không gian trạng thái: Một cách sắp xếp các quân hậu trên bàn cờ. Ta giới hạn không gian trạng thái ở những sắp xếp chỉ có đúng một quân hậu trong mỗi cột. Ta biểu diễn một trạng thái dưới dạng một vector số nguyên q = {q1, q2, q3,…}, trong đó mỗi số biểu thị vị trí hàng của quân hậu từ trái sang phải. Ta gọi một trạng thái là một "bàn cờ."

- Hàm mục tiêu: Số lượng xung đột theo cặp (tức là hai quân hậu nằm cùng hàng/cột/đường chéo).

- Bài toán tối ưu được phát biểu như sau:

  + minimize : conflict (q)

  + subject to: q chỉ chứa đúng một quân hậu trên mỗi cột

- Lưu ý: ràng buộc (subject to) được đảm bảo bởi định nghĩa của không gian trạng thái.

- Nước đi cải thiện cục bộ: Di chuyển một quân hậu sang một hàng khác trong cùng một cột.

- Điều kiện dừng: Với bài toán này, luôn tồn tại một sắp xếp q* *sao cho conflicts (q*) = 0*. Tuy nhiên, các bước cải thiện cục bộ có thể dẫn đến việc mắc kẹt trong cực tiểu địa phương.

II. Các hàm hỗ trợ.

- Để giải bài toán n-Queens, một số hàm hỗ trợ được xây dựng nhằm phục vụ cho việc khởi tạo, đánh giá và trực quan hóa trạng thái bàn cờ.

  - Hàm random_board(n)

    Hàm này tạo một bàn cờ ngẫu nhiên kích thước $n \times n$, trong đó mỗi cột chỉ

chứa một quân hậu duy nhất. Mỗi giá trị trong mảng board biểu diễn vị trí hàng của quân hậu trong cột tương ứng.

Ví dụ: board = [3, 1, 0, 2] nghĩa là:

- Cột 1 có hậu ở hàng 3
- Cột 2 có hậu ở hàng 1
- Cột 3 có hậu ở hàng 0
- Cột 4 có hậu ở hàng 2

- Hàm conflicts(board)

  Đây là hàm mục tiêu (objective function), dùng để tính số lượng xung đột giữa các quân hậu.

  Một xung đột xảy ra nếu có hai hậu nằm trên cùng hàng hoặc cùng đường chéo.

  Hàm này trả về tổng số cặp hậu đang tấn công nhau, được tính qua ba loại mảng đếm:

  - horizontal_cnt: số hậu trên mỗi hàng
  - diagonal1_cnt: số hậu trên đường chéo chính
  - diagonal2_cnt: số hậu trên đường chéo phụ

    Sau đó, tổng số xung đột được tính bằng tổng của các tổ hợp cặp hậu trên cùng hàng hoặc cùng chéo.

- Hàm show_board(board)

  Hàm này dùng để hiển thị bàn cờ và vị trí các quân hậu bằng thư viện matplotlib.

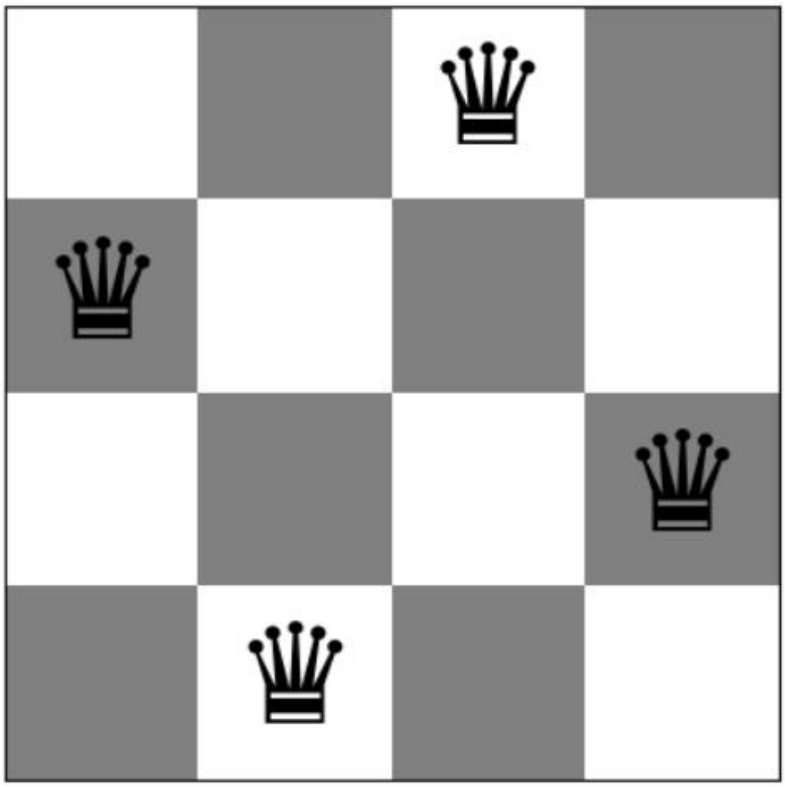
  Mỗi ô cờ được tô xen kẽ hai màu trắng – xám, còn quân hậu được biểu diễn bằng ký tự Unicode ♛.

  Đồng thời, hàm này cũng in ra số lượng xung đột hiện tại của bàn cờ.

- Ba hàm trên kết hợp giúp việc sinh, đánh giá và quan sát trạng thái bàn cờ trở nên dễ dàng — là nền tảng cho các thuật toán leo đồi (Hill Climbing) và mô phỏng tôi luyện (Simulated Annealing) trong bài toán n-Queens.

III. Tạo bàn cờ.

- Để bắt đầu bài toán n-Queens, ta cần khởi tạo một bàn cờ có kích thước $n \times n$ với n quân hậu được đặt trên đó. Mỗi quân hậu được bố trí sao cho mỗi cột chỉ có đúng một quân hậu, điều này giúp thu hẹp không gian tìm kiếm và đảm bảo rằng không có hai quân hậu nào cùng cột ngay từ đầu.
- Cách biểu diễn bàn cờ được thực hiện thông qua một vector (mảng) một chiều, trong đó mỗi phần tử biểu thị vị trí hàng của quân hậu trong cột tương ứng. Ví dụ, với bàn cờ 4×4, vector [1, 3, 0, 2] có nghĩa là:
    - Quân hậu ở cột 1 nằm tại hàng 1
    - Quân hậu ở cột 2 nằm tại hàng 3
    - Quân hậu ở cột 3 nằm tại hàng 0
    - Quân hậu ở cột 4 nằm tại hàng 2



- Khi bàn cờ được khởi tạo ngẫu nhiên, có thể xảy ra tình trạng các quân hậu tấn công lẫn nhau do cùng nằm trên một hàng hoặc cùng đường chéo. Số lượng các xung đột này được xem là giá trị mục tiêu (objective value) cần được giảm thiểu trong quá trình tìm kiếm lời giải.
- Việc hiển thị bàn cờ giúp ta quan sát rõ vị trí các quân hậu và đánh giá trực quan mức độ xung đột hiện tại. Trên bàn cờ, các ô được tô xen kẽ hai màu trắng – xám, và mỗi quân hậu được ký hiệu bằng biểu tượng ♛.
- Một bàn cờ được xem là nghiệm hợp lệ khi không tồn tại bất kỳ xung đột nào, tức là không có hai quân hậu nào cùng hàng, cùng cột hoặc cùng đường chéo. Ví dụ, với bàn cờ kích thước 4×4, cấu hình [1, 3, 0, 2] là một trong những lời giải chính xác, vì trong trường hợp này, số lượng xung đột bằng 0.

IV. Nhiệm vụ.

1. Nhiệm vụ 1: Tìm kiếm leo đồi (Steepest-ascend Hill Climbing).

- Trong bài toán n-Queens, mỗi trạng thái của bàn cờ được biểu diễn bằng một mảng board có độ dài n, trong đó phần tử board[i] cho biết vị trí hàng của quân hậu nằm ở cột thứ i.
- Mục tiêu của bài toán là tìm ra một cấu hình bàn cờ mà không có hai quân hậu nào tấn công lẫn nhau, tức là số xung đột (conflicts) bằng 0.
- Thuật toán Steepest-Ascent Hill Climbing là một phương pháp tìm kiếm leo đồi (hill climbing search) có định hướng tối ưu cục bộ, trong đó ở mỗi bước lặp, thuật toán sẽ đánh giá toàn bộ các trạng thái lân cận (neighbors) và chọn trạng thái tốt nhất — tức là có số xung đột nhỏ nhất — để tiếp tục di chuyển.

Quy trình thực hiện

1. Khởi tạo trạng thái ban đầu:
   Bắt đầu từ một bàn cờ ngẫu nhiên, trong đó mỗi cột có một quân hậu được đặt ở vị trí hàng bất kỳ.

2. Sinh các trạng thái lân cận:
   Từ trạng thái hiện tại, lần lượt di chuyển từng quân hậu sang các hàng khác trong cùng cột để tạo ra tất cả các cấu hình bàn cờ có thể.

3. Đánh giá hàm mục tiêu:

   Với mỗi cấu hình sinh ra, tính toán số lượng xung đột giữa các quân hậu (bao gồm xung đột hàng và đường chéo).

4. Chọn trạng thái tốt nhất:

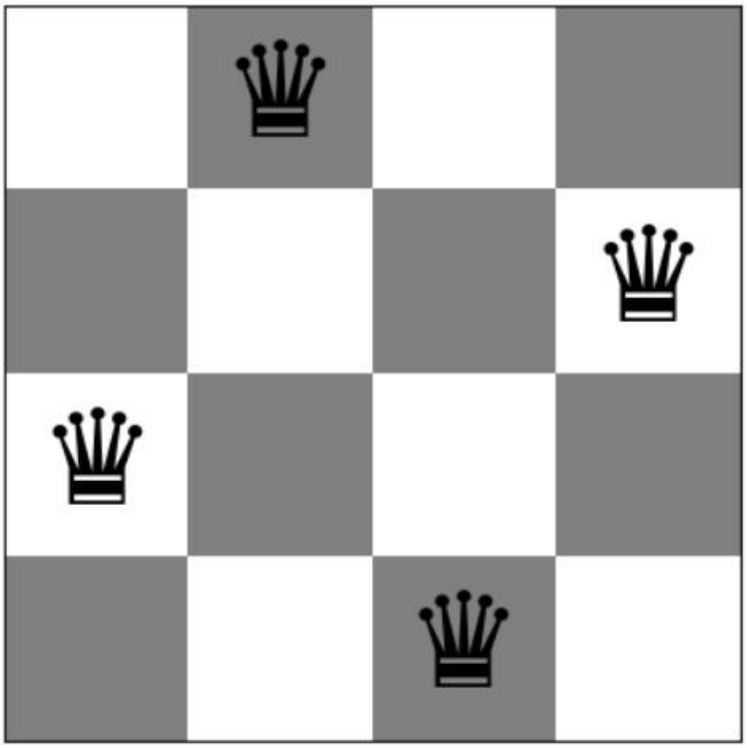
   Trong tất cả các trạng thái lân cận, chọn ra trạng thái có ít xung đột nhất. Nếu không có trạng thái nào tốt hơn, thuật toán dừng lại — lúc này đã đạt tới cực trị địa phương (local optimum).

5. Lặp lại quá trình:

   Di chuyển đến trạng thái tốt nhất vừa tìm được và tiếp tục lặp lại quy trình cho đến khi:

   - Tìm được nghiệm hợp lệ (số xung đột = 0), hoặc
   - Không thể cải thiện thêm (đạt cực trị địa phương).

Kết quả minh họa



2. Nhiệm vụ 2: Tìm kiếm leo đồi ngẫu nhiên 1 (Stochastic Hill Climbing 1).

- Stochastic Hill Climbing 1 (SHC 1) là một thuật toán tìm kiếm cục bộ (Local Search) dựa trên độ dốc, thuộc nhóm các thuật toán cải thiện dần (iterative improvement). Nó khác với Steepest Ascent Hill Climbing ở quy tắc lựa chọn nước đi.
- Nguyên tắc hoạt động: Thuật toán sẽ tìm tất cả các nước đi "lên dốc" (các nước đi làm giảm số xung đột) và sau đó chọn ngẫu nhiên một trong các nước đi đó. Quá trình dừng lại khi đạt đến trạng thái giải pháp (0 xung đột) hoặc một cực tiểu cục bộ (không còn nước đi nào giảm xung đột).

  + Hàm Mục tiêu (Objective Function): H (được gọi là conflicts trong mã) là số lượng cặp quân hậu tấn công nhau. Mục tiêu của thuật toán là giảm thiểu H về giá trị tối ưu là H=0.

  + Không gian Lân cận: Các trạng thái lân cận được tạo ra bằng cách di chuyển một quân hậu đến một vị trí hàng khác trong cùng cột.

- Thuật toán này là một biến thể của Hill Climbing nhằm giải quyết vấn đề mắc kẹt tại các bề mặt phẳng (plateaus) và cực tiểu cục bộ (local optima) cứng nhắc hơn so với Hill Climbing tiêu chuẩn.
- Tìm kiếm các Nước đi Lên dốc (Uphill Moves): Tại mỗi bước lặp, thuật toán xét tất cả các trạng thái lân cận (neighbor states) bằng cách di chuyển một quân hậu duy nhất trong cột của nó. Chỉ những nước đi nào làm giảm nghiêm ngặt số lượng xung đột (tức là Conflicts(Neighbor)<Conflicts(Current)) mới được coi là nước đi lên dốc (uphill moves).
- Lựa chọn Ngẫu nhiên: Thay vì chọn nước đi tốt nhất (Best-First Choice) như Hill Climbing tiêu chuẩn, thuật toán này chọn ngẫu nhiên một nước đi từ tập hợp các nước đi lên dốc đã tìm thấy. Việc chọn ngẫu nhiên giúp thuật toán tránh được việc đi theo cùng một con đường tối ưu cục bộ lặp đi lặp lại và có cơ hội thoát khỏi các bề mặt phẳng.
- Điều kiện Dừng: Thuật toán dừng lại khi một trong hai điều kiện sau được thõa mã:

  + Thành công (Success): Tìm thấy giải pháp tối ưu toàn cục: Conflicts(Current)=0.

+ Thất bại/Kẹt (Stuck): Đạt đến một Cực tiểu Cục bộ (Local Minimum): Tập hợp các nước đi lên dốc là trống rỗng (not uphill_moves). Điều này có nghĩa là không có bất kỳ nước đi lân cận nào có thể giảm số xung đột, buộc thuật toán phải dừng lại ở một trạng thái không phải là giải pháp.

+ Ngưỡng Bước đi: Đạt đến số bước lặp tối đa (max_steps) được định nghĩa trước, nhằm ngăn chặn vòng lặp vô hạn.

3. Nhiệm vụ 3:Tìm kiếm leo đồi Ngẫu nhiên 2 (Stochastic Hill Climbing 2).

- Trong Task 3, chúng ta triển khai một biến thể phổ biến của thuật toán leo đồi ngẫu nhiên (Stochastic Hill Climbing), được gọi là First-choice Hill Climbing. Khác với phương pháp Steepest-ascent Hill Climbing — vốn phải duyệt qua toàn bộ các trạng thái lân cận để tìm nước đi tốt nhất — thuật toán này chỉ chọn ngẫu nhiên một trạng thái lân cận tại mỗi bước, sau đó chấp nhận ngay nếu trạng thái mới có giá trị hàm mục tiêu tốt hơn (tức số xung đột giảm).

- Cách tiếp cận này giúp giảm chi phí tính toán rất đáng kể, đặc biệt khi không gian tìm kiếm lớn (ví dụ n-Queens với n lớn, mỗi trạng thái có rất nhiều lân cận). Thay vì đánh giá tất cả các nước đi có thể, thuật toán chỉ cần kiểm tra ngẫu nhiên và tiến về hướng cải thiện bất kỳ nào xuất hiện trước.
- Quy trình hoạt động có thể tóm tắt như sau:

  + Khởi tạo một bàn cờ ngẫu nhiên (mỗi cột có đúng một quân hậu).

  + Tính toán hàm xung đột (conflict function) để đo chất lượng trạng thái hiện tại.

  + Ở mỗi bước lặp:

    ▪ Chọn ngẫu nhiên một cột và một hàng khác để di chuyển quân hậu trong cột đó.

    ▪ Nếu trạng thái mới có ít xung đột hơn trạng thái hiện tại, thì chấp nhận nó làm trạng thái mới.

    ▪ Nếu không tốt hơn, thì giữ nguyên trạng thái cũ và tiếp tục thử một lựa chọn ngẫu nhiên khác.

+ Thuật toán dừng lại khi tìm được lời giải tối ưu (không còn xung đột) hoặc sau một số lượng thử không cải thiện nhất định — được xem như đã đạt đến cực tiểu cục bộ (local optimum).

- Về bản chất, First-choice Hill Climbing là một chiến lược tham lam (greedy) nhưng ngẫu nhiên, hướng tới việc cải thiện liên tục mà không cần tìm bước tốt nhất toàn cục. Nhờ yếu tố ngẫu nhiên, nó có khả năng thoát khỏi các plateau hoặc vùng bằng phẳng trong không gian trạng thái, nơi mà tất cả các nước đi đều cho cùng giá trị hàm mục tiêu, một điểm yếu thường gặp của Hill Climbing truyền thống.
- Tuy nhiên, hạn chế của thuật toán là dễ bị mắc kẹt trong các cực tiểu cục bộ nếu không có đủ thời gian hoặc số lần thử để tìm hướng đi tốt hơn. Do đó, để nâng cao hiệu quả, phương pháp này thường được kết hợp với kỹ thuật random restart hoặc simulated annealing, nhằm tăng xác suất tìm ra nghiệm tối ưu toàn cục.
- Tóm lại, Stochastic Hill Climbing 2 (First-choice) mang lại sự đơn giản, hiệu quả tính toán, và là lựa chọn thực tiễn khi xử lý các bài toán có không gian tìm kiếm lớn, như bài toán n-Queens.

4. Nhiệm vụ 4: Tìm kiếm Leo đồi với Khởi động lại Ngẫu nhiên (Hill Climbing Search with Random Restarts).

❖ Mục tiêu

- Mục tiêu chính: Khắc phục nhược điểm cố hữu của thuật toán Leo đồi cơ bản là dễ bị mắc kẹt tại các điểm tối ưu cục bộ (local optima).
- Mục tiêu phụ: Triển khai một thuật toán hoàn chỉnh có khả năng tìm ra lời giải tối ưu (0 xung đột) một cách ổn định hơn so với việc chỉ chạy Leo đồi một lần duy nhất.

❖ Vấn đề và Phương pháp giải quyết

- Vấn đề: Thuật toán Leo đồi hoạt động bằng cách luôn di chuyển đến trạng thái "tốt hơn" (có ít xung đột hơn) trong số các trạng thái lân cận. Tuy nhiên, nó sẽ dừng lại khi không còn trạng thái lân cận nào tốt hơn. Tại thời điểm này, nó đã đạt đến một "đỉnh", nhưng không có gì đảm bảo đây là "đỉnh cao nhất toàn cục" (global optimum). Trong bài toán N-Queens, điều này có nghĩa là thuật toán có thể tìm ra

một cấu hình bàn cờ với 1 hoặc 2 xung đột và không thể cải thiện thêm, mặc dù chúng ta biết rằng luôn tồn tại lời giải với 0 xung đột.

- Phương pháp giải quyết – "Khởi tạo lại Ngẫu nhiên": Để giải quyết vấn đề này, chúng tôi đã triển khai kỹ thuật "Khởi tạo lại Ngẫu nhiên". Nguyên lý hoạt động của kỹ thuật này rất trực quan:

1. Khởi tạo: Bắt đầu với một cấu hình bàn cờ hoàn toàn ngẫu nhiên.
2. Tìm kiếm: Áp dụng thuật toán Leo đồi (ví dụ: Steepest Ascent) để tìm kiếm từ điểm khởi tạo này cho đến khi nó đạt đến một điểm tối ưu cục bộ.
3. Đánh giá: Kiểm tra xem lời giải tìm được có phải là lời giải tối ưu toàn cục (0 xung đột) hay không.
    - Nếu có: Thuật toán thành công và dừng lại.
    - Nếu không: Thuật toán lưu lại lời giải tốt nhất đã tìm thấy, sau đó *hủy bỏ* trạng thái hiện tại và quay lại bước 1, bắt đầu lại từ một điểm ngẫu nhiên hoàn toàn mới.
4. Lặp lại: Quá trình này được lặp đi lặp lại một số lần nhất định (ví dụ: 100 lần) hoặc cho đến khi tìm được lời giải tối ưu.

- Bằng cách này, thuật toán có nhiều cơ hội để "khám phá" các khu vực khác nhau của không gian tìm kiếm, thay vì chỉ bị giới hạn trong một "thung lũng" duy nhất.

❖ **Kết quả và Thảo luận**

- Kết quả: Sau khi triển khai, thuật toán Leo đồi kết hợp với Khởi tạo lại Ngẫu nhiên đã cho thấy một sự cải thiện vượt bậc về tỷ lệ thành công. Đối với bài toán 8-Queens, trong khi một lần chạy Leo đồi cơ bản có thể thất bại, việc cho phép nó khởi tạo lại 100 lần gần như đảm bảo sẽ tìm ra một lời giải tối ưu.
- Thảo luận: Kỹ thuật Khởi tạo lại Ngẫu nhiên là một minh chứng rõ ràng rằng việc kết hợp một thuật toán tìm kiếm đơn giản với một chiến lược điều khiển cấp cao có thể mang lại hiệu quả bất ngờ. Nó không làm cho bản thân thuật toán Leo đồi thông minh hơn, nhưng nó đảm bảo rằng sự "kém thông minh" của thuật toán không ngăn cản chúng ta tìm ra lời giải. Điểm yếu của phương pháp này là không

có gì đảm bảo sẽ tìm ra lời giải trong một số lần khởi tạo hữu hạn, nhưng trong thực tế, nó hoạt động rất tốt cho các bài toán như N-Queens.

5. Nhiệm vụ 5: Tôi luyện mô phỏng (Simulated Annealing).

- Trong bài toán n-Queens, một trong những khó khăn chính của các thuật toán tìm kiếm cục bộ (local search) như Hill Climbing là dễ rơi vào cực trị địa phương (local optimum), khiến quá trình tìm nghiệm tối ưu dừng lại sớm dù chưa đạt lời giải đúng (0 xung đột).
- Thuật toán Simulated Annealing (SA) được phát triển nhằm khắc phục nhược điểm này bằng cách cho phép chấp nhận các bước tệ hơn (uphill move) với một xác suất nhất định, giúp thuật toán thoát khỏi các cực trị địa phương.

❖ **Nguyên lý hoạt động:**

- SA được lấy cảm hứng từ quá trình nung luyện kim loại (annealing) — khi vật liệu được nung nóng rồi làm nguội dần, các nguyên tử sẽ di chuyển để tìm cấu trúc ổn định có năng lượng thấp nhất.
- Tương tự, trong bài toán tối ưu:

  + Ở nhiệt độ cao (T lớn): thuật toán có thể chấp nhận bước xấu hơn (conflicts tăng) với xác suất cao → giúp khám phá không gian nghiệm rộng hơn.

  + Khi T giảm dần, thuật toán giảm dần xác suất chấp nhận bước xấu → hội tụ về nghiệm tốt hơn và ổn định hơn.
- Xác suất chấp nhận bước xấu được xác định bởi công thức:

$$P = e^{-\Delta E/T}$$

với

+ ΔE= (conflicts mới - conflicts hiện tại),

+ T: nhiệt độ hiện tại.

❖ **Lịch giảm nhiệt độ (Annealing Schedule):**

- Trong bài, ta sử dụng Exponential Decay Schedule:

$$T = T_0 \times \alpha^k$$

với:

+ T_0: nhiệt độ ban đầu,

+ α: hệ số làm nguội ($0.9 \leq \alpha \leq 0.99$),

+ k: số bước lặp hiện tại.

- Cách giảm này giúp nhiệt độ giảm dần đều, cho phép thuật toán cân bằng giữa khả năng khám phá và hội tụ.

❖ **Quá trình thực hiện:**

- Khởi tạo một bàn cờ ngẫu nhiên kích thước $n = 8$.
- Ở mỗi bước lặp:

  + Chọn ngẫu nhiên một quân hậu và di chuyển đến vị trí hàng khác.

  + Tính số xung đột mới.

  + Quyết định chấp nhận hoặc từ chối bước đi dựa trên xác suất $e^{-\Delta E/T}$.

  + Giảm nhiệt độ T theo lịch trình.

❖ **Kết quả thực nghiệm:**

- Khi chạy với các tham số $T_0 = 100$, $\alpha = 0.98$: Thuật toán dần giảm số xung đột từ 5 xuống 1, chứng tỏ khả năng cải thiện nghiệm dần theo thời gian.
- Biểu đồ quá trình hội tụ cho thấy ban đầu xung đột dao động mạnh (do T lớn, chấp nhận bước xấu nhiều), sau đó dần ổn định khi T giảm.
- Thử nghiệm thay đổi hệ số làm nguội (α):

  + α = 0.95 → giảm nhanh, hội tụ nhanh nhưng dễ mắc cực trị địa phương.

  + α = 0.98 → cân bằng tốt giữa khám phá và hội tụ.

  + α = 0.99 → giảm chậm, chạy lâu nhưng có khả năng tìm nghiệm tối ưu cao hơn.

6. Nhiệm vụ 6: (Algorithm Behavior Analysis).

❖ **Comparision**:

- So sánh các thuật toán dựa trên thời gian chạy và giá trị của hàm mục tiêu. Sử dụng kích thước bàn cờ là 4 và 8 để khảo sát hiệu suất của các thuật toán khác nhau. Đảm bảo rằng chạy các thuật toán cho mỗi kích thước bàn cờ nhiều lần (ít nhất 100 lần) với các trạng thái khởi tạo khác nhau và báo cáo các giá trị trung bình.

+ Mục tiêu: So sánh hiệu suất của các thuật toán dựa trên hai chỉ số chính

  o Thời gian chạy (Runtime): Đo lường tốc độ thuật toán hội tụ hoặc tìm ra giải pháp. (Thường tính bằng giây hoặc mili giây).

  o Giá trị Hàm Mục tiêu (Objective Function Values): Trong bài toán N-Queens, hàm mục tiêu h là số lượng cặp quân hậu xung đột (tấn công nhau).

- Giá trị tối ưu (Optimal value) là h=0 (nghĩa là không có xung đột).
- Các thuật toán được khảo sát:

1. Steepest Ascent Hill-Climbing (SAHC): Thuật toán tham lam luôn chọn bước đi cải thiện tốt nhất (giảm xung đột nhiều nhất) trong số tất cả các bước lân cận.

2. Stochastic Hill-Climbing 1 (SHC 1): Thuật toán tìm tất cả các bước cải thiện, sau đó chọn ngẫu nhiên một trong số chúng.

3. Stochastic Hill-Climbing 2 (SHC 2 - First-Choice): Thuật toán tìm kiếm ngẫu nhiên các bước đi lân cận và chấp nhận ngay bước cải thiện đầu tiên tìm thấy.

4. Simulated Annealing (SA): Thuật toán cho phép chấp nhận các bước đi tồi hơn (tăng xung đột) với xác suất giảm dần theo thời gian (nhiệt độ).

+ Phương pháp Thực nghiệm:

  o Kích thước Bàn cờ (N): Sử dụng N=4 (bài toán đơn giản) và N=8 (bài toán phức tạp hơn, tiêu chuẩn).

  o Số lần Chạy (Runs): Chạy mỗi thuật toán với mỗi kích thước N ít nhất 100 lần.

  o Điều kiện Khởi tạo: Mỗi lần chạy phải bắt đầu từ một trạng thái bàn cờ ngẫu nhiên (random starting board) khác nhau. Điều này rất quan trọng vì các thuật toán tìm kiếm cục bộ rất nhạy cảm với điểm khởi đầu.

+ Phân tích hành vi thuật toán: Kết quả thực nghiệm sẽ làm sáng tỏ các đặc điểm sau:

- Hill-Climbing (SAHC, SHC): Các thuật toán này có xu hướng chạy nhanh (ít di chuyển) nhưng thường bị mắc kẹt tại các cực đại cục bộ (Avg. Conflicts >0), đặc biệt với N=8.
- Simulated Annealing (SA): Mặc dù có thời gian chạy chậm hơn do lặp lại nhiều lần (max_iter) và tính toán hàm xác suất, SA thường đạt được tỷ lệ thành công cao hơn do cơ chế chấp nhận bước đi tồi ($\Delta H>0$), cho phép nó nhảy ra khỏi các cực đại cục bộ để tiếp cận vùng tối ưu toàn cục.

❖ **Algorithm Convergence:**

- Trong quá trình thử nghiệm, các thuật toán leo đồi và mô phỏng tôi luyện (hill climbing và simulated annealing) thể hiện các kiểu hội tụ (convergence patterns) khác nhau khi giải bài toán 8-Queens.
- Steepest-Ascent Hill Climbing:

  + Đặc điểm hội tụ: Giảm nhanh số lượng conflicts trong vài bước đầu tiên, sau đó dừng đột ngột.

  + Nguyên nhân: Thuật toán luôn chọn nước đi tốt nhất nên dễ rơi vào local optimum + hoặc vùng plateau (các trạng thái có cùng giá trị).

  + Biểu đồ: Đường cong giảm mạnh ban đầu rồi phẳng ngang sớm — thể hiện việc hội tụ nhanh nhưng không ổn định.
- Stochastic Hill Climbing 1:

  + Đặc điểm hội tụ: Giảm dần đều hơn, không quá nhanh nhưng có khả năng tránh một số local optimum.

  + Nguyên nhân: Việc chọn ngẫu nhiên giữa các nước đi tốt giúp thuật toán có đa dạng hướng đi, tránh lặp lại mô hình di chuyển cố định.

  + Biểu đồ: Dao động nhẹ, nhưng vẫn có xu hướng giảm ổn định.
- Stochastic Hill Climbing 2 (First-Choice Hill Climbing)

  + Đặc điểm hội tụ: Tốc độ cải thiện chậm hơn nhưng tổng thể hiệu quả hơn trong việc tìm ra lời giải.

  + Nguyên nhân: Chỉ xem xét từng nước đi ngẫu nhiên và chấp nhận nếu tốt hơn, nên tiết kiệm thời gian tính toán và ít bị "kẹt cứng".

  + Biểu đồ: Giảm theo từng bậc nhỏ, có thể dao động nhưng cuối cùng hội tụ dần về 0.

- Simulated Annealing:

  + Đặc điểm hội tụ: Giảm chậm nhưng ổn định, có thể tạm thời tăng conflicts do chấp nhận các nước đi xấu với xác suất nhất định.

  + Nguyên nhân: Cơ chế nhiệt độ (temperature schedule) cho phép thoát khỏi local optimum ở giai đoạn đầu, sau đó dần tập trung tối ưu khi "nhiệt độ" giảm.

  + Biểu đồ: Đường cong dao động mạnh lúc đầu (do chấp nhận nước đi xấu), rồi ổn định dần và hội tụ về 0.

- Steepest Hill Climbing:

  + Nhanh nhưng dễ kẹt. Stochastic 1 & 2: Cải thiện khả năng tránh bẫy cục bộ, tốc độ vừa phải. Simulated Annealing: Hội tụ chậm nhất nhưng đáng tin cậy nhất trong việc tìm lời giải tối ưu. Nhìn chung, tốc độ hội tụ và khả năng thoát bẫy local optimum là hai yếu tố đánh đổi lẫn nhau giữa các thuật toán local search.

❖ **Problem Size Scalability:**

- **Mục tiêu:**

  + Mục tiêu chính: Đánh giá và so sánh hiệu suất về mặt thời gian của hai biến thể Leo đồi khi kích thước bàn cờ n tăng lên.

  + Mục tiêu phụ: Ước tính độ phức tạp thời gian thực nghiệm (Big O) của mỗi thuật toán và xác định thuật toán nào phù hợp hơn để giải quyết các bài toán N-Queens quy mô lớn.

- **Phương pháp thực nghiệm:**

  + Các thuật toán được so sánh:

  1. Leo đồi Dốc nhất (Steepest-Ascent Hill Climbing): Tại mỗi bước, thuật toán này đánh giá tất cả n × (n-1) trạng thái lân cận để tìm ra nước đi tốt nhất.

  2. Leo đồi Lựa chọn Đầu tiên (First-Choice Hill Climbing): Tại mỗi bước, thuật toán này chỉ tạo và đánh giá ngẫu nhiên các trạng thái lân cận cho đến khi tìm thấy một trạng thái tốt hơn trạng thái hiện tại.

- **Quy trình thực nghiệm:**

1. Chúng tôi chọn một loạt các kích thước bàn cờ để kiểm tra: n = 4, 8, 12, 16.

2. Với mỗi kích thước n, chúng tôi chạy mỗi thuật toán (kết hợp với Khởi tạo lại Ngẫu nhiên để đảm bảo tìm ra lời giải) nhiều lần (ví dụ: 3-5 lần) để có được kết quả ổn định.

3. Chúng tôi ghi lại thời gian chạy trung bình để tìm ra lời giải 0 xung đột cho mỗi cặp (thuật toán, kích thước bàn cờ).

4. Cuối cùng, chúng tôi vẽ đồ thị log-log biểu diễn mối quan hệ giữa thời gian chạy (trục y) và kích thước bàn cờ (trục x). Biểu đồ log-log rất hữu ích vì một mối quan hệ đa thức $T(n) \approx c * n^k$ sẽ xuất hiện dưới dạng một đường thẳng, giúp việc so sánh độ phức tạp trở nên trực quan.

- **Kết quả và Phân tích**

+ Kết quả: Biểu đồ log-log thu được cho thấy một sự khác biệt rõ rệt về hiệu suất giữa hai thuật toán:

- Đường biểu diễn của Leo đồi Dốc nhất: Có độ dốc lớn, gần như thẳng đứng khi n tăng. Thời gian chạy tăng từ mili giây ở n=4 lên đến hàng chục giây hoặc vài phút ở n=16.
- Đường biểu diễn của Leo đồi Lựa chọn Đầu tiên: Có độ dốc thoải hơn nhiều. Mặc dù thời gian chạy cũng tăng theo n, nhưng tốc độ tăng chậm hơn đáng kể so với đối thủ.

+ Phân tích:

- Leo đồi Dốc nhất tỏ ra không hiệu quả khi n lớn. Chi phí để "cẩn thận" tìm ra nước đi tốt nhất ở mỗi bước (O(n^2)) trở thành một nút thắt cổ chai khổng lồ. Khi không gian tìm kiếm phình to, sự cẩn thận này không còn mang lại lợi ích tương xứng với chi phí bỏ ra.
- Leo đồi Lựa chọn Đầu tiên thể hiện khả năng mở rộng vượt trội. Bằng cách hy sinh sự "tối ưu" ở mỗi bước đi để đổi lấy tốc độ, nó có thể thực

hiện nhiều bước hơn trong cùng một khoảng thời gian. Trong các không gian tìm kiếm lớn, khả năng di chuyển nhanh và thoát khỏi các khu vực xấu một cách nhanh chóng tỏ ra quan trọng hơn là việc tìm ra con đường dốc nhất ở mỗi ngã rẽ.

- **Kết luận:** Đối với các bài toán N-Queens có kích thước lớn, thuật toán Leo đồi Lựa chọn Đầu tiên là lựa chọn tốt hơn hẳn do khả năng mở rộng vượt trội. Phân tích này nhấn mạnh một nguyên tắc quan trọng trong thiết kế thuật toán: sự cân bằng giữa chất lượng của một bước đi và chi phí tính toán để thực hiện nó. Trong nhiều trường hợp, một chiến lược "đủ tốt và nhanh" sẽ đánh bại một chiến lược "hoàn hảo nhưng chậm chạp".

7. Nhiệm vụ nâng cao: Khám phá các toán tử di chuyển cục bộ khác (Exploring other local moves operators).

❖ **Giới thiệu:**

- Trong các thuật toán Local Search như Hill Climbing hoặc Simulated Annealing, khái niệm *move operator* đóng vai trò rất quan trọng, vì nó xác định cách sinh ra các trạng thái lân cận (neighbor states) trong không gian tìm kiếm.
- Việc lựa chọn toán tử di chuyển ảnh hưởng trực tiếp đến khả năng thoát khỏi cực trị địa phương (local optimum) và tốc độ hội tụ của thuật toán.
- Ở bài toán n-Queens, thông thường mỗi trạng thái được biểu diễn bằng một mảng một chiều, trong đó board[i] cho biết hàng mà quân hậu ở cột thứ *i* được đặt.
- Bài này triển khai và so sánh bốn loại toán tử di chuyển khác nhau, nhằm phân tích ảnh hưởng của chúng đến chất lượng nghiệm và thời gian chạy của thuật toán.

❖ **Các loại toán tử di chuyển được thử nghiệm:**

Tên Move Operator	**Mô tả**	**Đặc điểm nổi bật**
Single-step	Di chuyển một quân hậu lên	Bước di chuyển nhỏ, giúp tinh

move	hoặc xuống một hàng duy nhất.	chỉnh nghiệm cục bộ nhưng dễ mắc kẹt ở cực trị địa phương.
Column swap	Hoán đổi vị trí của hai quân hậu ở hai cột khác nhau.	Thay đổi cấu trúc mạnh, có khả năng giúp thoát khỏi local minima.
Dual-queen move	Di chuyển đồng thời hai quân hậu đến hai hàng ngẫu nhiên.	Tăng mức độ đa dạng của tìm kiếm, hiệu quả trong giai đoạn đầu nhưng không ổn định khi gần nghiệm tối ưu.
Adaptive move	Tự động chọn chiến lược di chuyển dựa trên trạng thái hiện tại (ưu tiên quân có nhiều xung đột).	Linh hoạt, hiệu quả cao, kết hợp ưu điểm của các phương pháp khác.

❖ **Quy trình thử nghiệm:**

- Mỗi toán tử được sử dụng trong thuật toán Stochastic Hill Climbing.
- Các thử nghiệm được tiến hành với hai kích thước bàn cờ:
  + 8-Queens
  + 12-Queens
- Mỗi toán tử được chạy 50 lần lặp độc lập, mỗi lần tối đa 5000 bước.
  Kết quả được đánh giá theo ba tiêu chí:
  1. Chất lượng nghiệm trung bình (số xung đột còn lại theo số lần lặp)
  2. Phân bố chất lượng nghiệm cuối cùng
  3. Thời gian trung bình để đạt nghiệm hợp lệ

❖ **Kết quả và phân tích:**

▪ Bài toán 8-Queens:

- Từ biểu đồ trung bình (*Average Solution Quality*), có thể thấy các toán tử Dual-queen và Adaptive giảm nhanh số lượng xung đột và đạt mức trung bình nhỏ hơn 1 sau khoảng 2000 lần lặp.
- Column swap và Single-step hội tụ chậm hơn, dễ mắc kẹt ở nghiệm cục bộ.
- Biểu đồ phân bố (*Distribution of Final Solution Qualities*) cho thấy Adaptive có nhiều nghiệm đạt 0 xung đột nhất.

- Thời gian trung bình: Adaptive mất nhiều thời gian hơn một chút (≈0.13s), nhưng chất lượng nghiệm tốt hơn rõ rệt.

▪ Bài toán 12-Queens:

- Khi số lượng quân hậu tăng, độ phức tạp và khả năng rơi vào cực trị địa phương tăng lên.
- Adaptive move và Dual-queen move tiếp tục cho kết quả tốt hơn rõ rệt về chất lượng nghiệm.
- Column swap cho thấy khả năng thoát local minima nhưng không ổn định, nhiều nghiệm vẫn còn 5–8 xung đột.
- Thời gian chạy trung bình:

  + *Single-step*: 0.117s

  + *Column swap*: 0.100s

  + *Dual-queen*: 0.134s

  + *Adaptive*: 0.332s

- Dù thời gian lâu hơn, Adaptive vẫn được xem là tối ưu nhất do tỷ lệ đạt nghiệm hoàn hảo cao nhất và mức hội tụ ổn định.

❖ **Đánh giá và kết luận:**

Move Operator	**Ưu điểm**	**Nhược điểm**	**Nhận xét tổng thể**
Single-step	Di chuyển cẩn trọng, giúp tinh chỉnh nghiệm nhỏ.	Dễ mắc kẹt ở local minimum.	Tốt cho giai đoạn cuối của tìm kiếm.
Column swap	Thoát local minimum hiệu quả.	Dễ làm xấu nghiệm tạm thời.	Phù hợp cho các bàn cờ lớn.
Dual-queen	Tăng tính ngẫu nhiên, giúp đa dạng hoá không gian tìm kiếm.	Thiếu ổn định khi gần nghiệm tối ưu.	Hiệu quả trong giai đoạn khởi đầu.

Adaptive	Linh hoạt, hiệu quả cao, tập trung xử lý vùng xung đột.	Cần thêm tính toán, thời gian cao hơn.	Hiệu quả nhất tổng thể.

❖ **Kết luận chung:**

- Từ kết quả thử nghiệm, có thể rút ra các nhận định sau:

  + Adaptive move là toán tử hiệu quả nhất cả về khả năng tìm nghiệm hợp lệ và tốc độ hội tụ.

  + Dual-queen move cũng cho hiệu suất tốt, đặc biệt trong các bàn cờ lớn.

  + Việc kết hợp linh hoạt giữa các loại move operators giúp thuật toán tìm kiếm ngẫu nhiên vượt qua giới hạn của cực trị địa phương, mang lại kết quả tối ưu hơn so với việc chỉ sử dụng một kiểu di chuyển duy nhất.

- Như vậy, bài toán n-Queens minh họa rõ ràng rằng thiết kế toán tử di chuyển phù hợp có ảnh hưởng trực tiếp đến hiệu quả của thuật toán Local Search trong việc giải quyết các bài toán tối ưu tổ hợp phức tạp.

8. More things to do.

Thực hiện (cài dặt) Thuật toán Di truyền cho bài toán n-Queens (Implement a Genetic Algorithm for the n-Queens problem)

Thuật toán Di truyền (Genetic Algorithm - GA) là một phương pháp tối ưu hóa dựa trên quần thể (population-based search), mô phỏng quá trình chọn lọc tự nhiên và tiến hóa để tìm kiếm giải pháp tối ưu cho các bài toán phức tạp.

Đặc điểm nổi bật: GA xử lý một tập hợp các giải pháp tiềm năng (quần thể) đồng thời, giúp nó hiệu quả trong việc thoát khỏi các cực đại cục bộ (local optima) so với các phương pháp tìm kiếm cục bộ đơn điểm.

Các thành phần chính:

- Biểu diễn Cá thể:

+ Mỗi cá thể là một giải pháp tiềm năng cho bài toán n-Queens, được biểu diễn bằng một mảng (list) có độ dài N.

+ Giá trị tại vị trí i trong mảng đại diện cho **cột** của quân hậu nằm ở hàng i.

+ Ví dụ: [2, 4, 1, 3] cho N=4 nghĩa là: Hàng 1 cột 2, Hàng 2 cột 4, Hàng 3 cột 1, Hàng 4 cột 3.

+ Ưu điểm của cách biểu diễn này: Nó tự động đảm bảo không có xung đột hàng và xung đột cột (mỗi hàng chỉ có 1 quân, mỗi cột chỉ có 1 quân), do đó, thuật toán chỉ cần tập trung vào việc giảm thiểu xung đột đường chéo.

- Hàm Thể lực :

+ Đo lường mức độ "tốt" của cá thể. Ta muốn tối đa hóa thể lực.

+ Thể lực được tính bằng tổng số cặp quân hậu không tấn công nhau.

+ Giá trị thể lực tối đa (giải pháp tối ưu) là $F_{\max} = \frac{N(N-1)}{2}$.

- Các toán tử di truyền:

Toán tử	Chức năng	Cơ chế Triển khai
Chọn lọc	Chọn các cá thể khỏe làm bố mẹ	Sử dụng Phương pháp Bánh xe Roulette (Roulette Wheel Selection): Cá thể có thể lực càng cao thì xác suất được chọn để lai ghép càng lớn, mô phỏng nguyên tắc "kẻ mạnh được chọn".
Lai ghép	Kết hợp thông tin di truyền để tạo con cái	Lai ghép Một điểm (Single-Point Crossover): Chọn ngẫu nhiên một điểm cắt

		(crossover_point). Gen của con cái là sự kết hợp của gen bên trái từ bố mẹ thứ nhất và gen bên phải từ bố mẹ thứ hai.
Đột biến	Đưa sự đa dạng vào quần thể để thoát khỏi cục bộ	Chọn ngẫu nhiên một gen (vị trí hàng) với xác suất MUTATION_RATE (ví dụ: 5%) và thay đổi giá trị của nó (vị trí cột) thành một giá trị ngẫu nhiên khác.
Tối ưu	Bảo toàn cá thể tốt nhất qua các thế hệ	Cá thể có thể lực cao nhất của thế hệ hiện tại được trực tiếp đưa vào thế hệ mới mà không qua quá trình lai ghép/đột biến. Điều này đảm bảo rằng giải pháp tốt nhất đã tìm thấy sẽ không bị mất đi.

- Chu trình lặp và điều kiện dừng:

Thuật toán hoạt động theo một vòng lặp liên tục, tiến hóa quần thể qua các thế hệ:

+ Khởi tạo: Tạo quần thể ban đầu với P cá thể ngẫu nhiên.

+ Đánh giá: Tính toán thể lực cho từng cá thể.

+ Kiểm tra Dừng: Nếu tìm thấy cá thể có F=Fmax hoặc đạt đến MAX_GENERATIONS, dừng thuật toán.

+ Tạo thế hệ mới: Thực hiện Chọn lọc, Lai ghép, và Đột biến để tạo ra quần thể kế tiếp.

Solving the Traveling Salesman Problem using Local Search

(Giải quyết Bài toán Người bán hàng bằng cách sử dụng Tìm kiếm Cục bộ)

I. Bài toán.

1. Mục tiêu.

- Tìm ra chu trình ngắn nhất đi thăm mỗi thành phố trong số n thành phố đúng một lần và quay trở lại thành phố ban đầu. Dữ liệu cho trước là khoảng cách giữa từng cặp thành phố, trong đó di,j là khoảng cách từ thành phố i đến thành phố j.

2. Không gian trạng thái.

- Mỗi trạng thái đại diện cho một chu trình (tour). Các thành phố được đánh số và một chu trình có thể được biểu diễn bằng một vector π chứa thứ tự các thành phố được thăm (một hoán vị). Tức là, π(1) là chỉ số của thành phố được thăm đầu tiên, π(2) là chỉ số của thành phố thứ hai, và cứ tiếp tục như vậy.

- Hàm mục tiêu: Tối thiểu hóa độ dài chu trình. Bài toán tối ưu là tìm kiếm chu trình tối ưu π* đi qua n thành phố và quay về thành phố xuất phát:

3. Tối thiểu hóa

$$\text{tourlength}(\pi) = d_{\pi(n),\pi(1)} + \sum_{i=1}^{n-1} d_{\pi(i),\pi(i+1)}$$

- Với điều kiện : π là một vector hoán vị hợp lệ

4. Nước đi cục bộ.

- Hoán đổi vị trí của hai thành phố trong thứ tự chu trình

II. Các hàm hỗ trợ.

1. Giới thiệu.

- Trước khi triển khai các thuật toán tối ưu cho bài toán Người bán hàng du lịch (*Travelling Salesman Problem – TSP*), cần xây dựng một số hàm hỗ trợ (helper functions) để:

  + Khởi tạo dữ liệu ngẫu nhiên,

  + Tính toán độ dài hành trình,

  + Và trực quan hóa kết quả tìm được.

- Những hàm này đóng vai trò quan trọng trong việc chuẩn bị dữ liệu đầu vào và kiểm tra kết quả đầu ra của các thuật toán tối ưu như *Hill Climbing* hay *Simulated Annealing*.

2. Khởi tạo dữ liệu TSP ngẫu nhiên.

- Hàm random_tsp(n) được dùng để tạo ra một bài toán TSP ngẫu nhiên với n thành phố.
- Cụ thể:

  + Mỗi thành phố được biểu diễn bởi một điểm có tọa độ (x, y)trong hình vuông đơn vị $[0,1] \times [0,1]$.

  + Ma trận khoảng cách được tính theo khoảng cách Euclidean giữa các cặp thành phố.

- Kết quả trả về bao gồm:

  + pos: bảng chứa tọa độ các thành phố.

  + dist: ma trận khoảng cách giữa mọi cặp thành phố.

- Ví dụ, khi tạo TSP với 10 thành phố, ta thu được:

```
Positions:
      x     y
0  0.19  0.36
1  0.62  0.50
2  0.44  0.68
3  0.79  0.71
4  0.78  0.37
5  0.27  0.56
6  0.28  0.50
7  0.80  0.01
8  0.96  0.77
9  0.88  0.88
Distance matrix:
      0     1     2     3     4     5     6     7     8     9
0  0.00  0.45  0.41  0.69  0.59  0.22  0.17  0.70  0.87  0.86
1  0.45  0.00  0.26  0.27  0.20  0.35  0.35  0.52  0.43  0.46
2  0.41  0.26  0.00  0.35  0.46  0.21  0.24  0.76  0.53  0.48
3  0.69  0.27  0.35  0.00  0.34  0.53  0.55  0.70  0.18  0.19
4  0.59  0.20  0.46  0.34  0.00  0.54  0.52  0.36  0.44  0.52
5  0.22  0.35  0.21  0.53  0.54  0.00  0.06  0.76  0.72  0.68
6  0.17  0.35  0.24  0.55  0.52  0.06  0.00  0.72  0.73  0.71
7  0.70  0.52  0.76  0.70  0.36  0.76  0.72  0.00  0.77  0.87
8  0.87  0.43  0.53  0.18  0.44  0.72  0.73  0.77  0.00  0.14
9  0.86  0.46  0.48  0.19  0.52  0.68  0.71  0.87  0.14  0.00)
```

3. Sinh hành trình ngẫu nhiên.

- Hàm random_tour(n) tạo ra một hành trình ngẫu nhiên, tức là một hoán vị của các thành phố.
- Ví dụ: [8, 2, 7, 6, 4, 3, 0, 9, 5, 1]

có nghĩa là bắt đầu từ thành phố 8, sau đó đi lần lượt qua các thành phố khác cho đến khi quay lại điểm xuất phát.

4. Hàm tính độ dài hành trình.

- Hàm tour_length(tsp, tour) dùng để tính tổng chiều dài của hành trình hiện tại.

Công thức được áp dụng là:

$$\text{tourlength}(\pi) = d_{\pi(n),\pi(1)} + \sum_{i=1}^{n-1} d_{\pi(i),\pi(i+1)}$$

- Trong đó:

+ $\pi(i)$: thứ tự thành phố thứ *i* trong hành trình.

+ $d_{i,j}$: khoảng cách giữa hai thành phố *i* và *j*.

- Hàm này được dùng làm hàm mục tiêu (objective function) trong các thuật toán tối ưu.
- Ví dụ: tour_length(tsp, [8, 2, 7, 6, 4, 3, 0, 9, 5, 1]) = 4.44

nghĩa là tổng quãng đường của hành trình trên là 4.44 đơn vị.

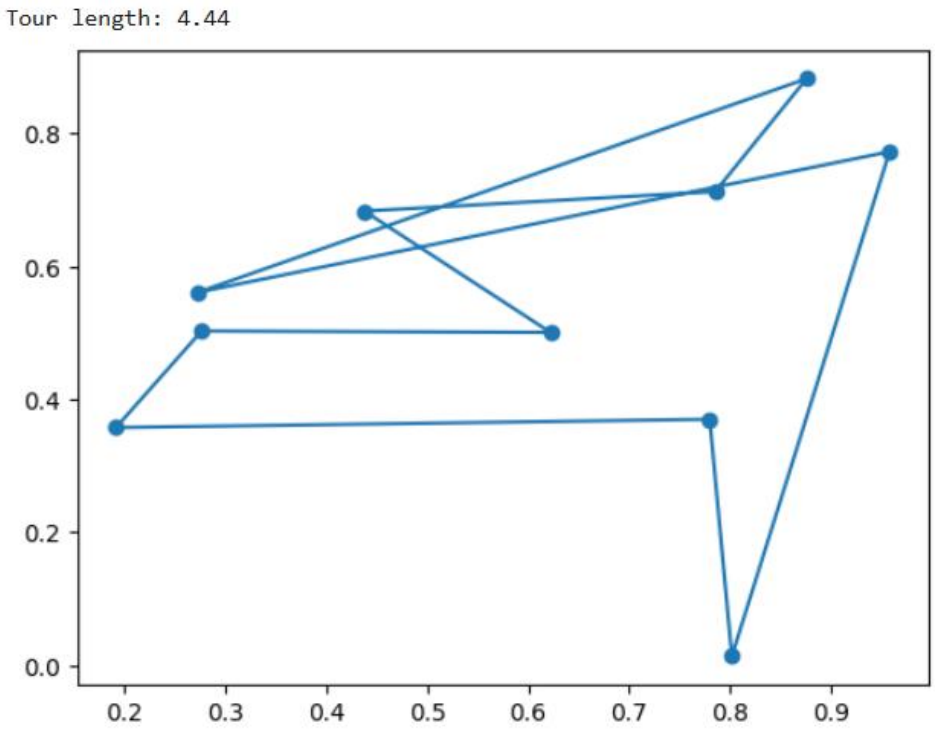
5. Hàm hiển thị đồ thị.

- Hàm show_tsp(tsp, tour) được sử dụng để trực quan hóa bài toán và hành trình đã chọn:

+ Các điểm biểu diễn vị trí các thành phố.

+ Các đường nối biểu diễn hành trình đi qua các thành phố theo thứ tự.

- Ví dụ khi hiển thị hành trình ngẫu nhiên:



và biểu đồ thể hiện rõ thứ tự di chuyển giữa các thành phố trên mặt phẳng.

6. Vai trò tổng quát.

- Tóm lại, các hàm hỗ trợ giúp:

+ Tạo dữ liệu TSP ngẫu nhiên và có thể lặp lại,

+ Đánh giá chất lượng của một hành trình,

+ Và hiển thị kết quả trực quan để dễ dàng so sánh giữa các phương pháp tối ưu khác nhau.

- Nhờ đó, toàn bộ quy trình giải TSP trở nên linh hoạt, dễ hiểu và có thể kiểm chứng bằng trực quan.

III. Sử dụng R để tìm ra một giải pháp.

1. Mục tiêu.

- Sau khi cài đặt và hiểu rõ cơ chế của các thuật toán tối ưu trong Python, nhóm tiến hành so sánh kết quả bằng cách sử dụng gói TSP của ngôn ngữ R.
- Mục tiêu là kiểm tra xem R có thể tìm được hành trình tốt hơn không, và đánh giá hiệu năng của quá trình giải TSP.

2. Chuẩn bị dữ liệu và thư viện.

- Dữ liệu đầu vào được chuyển từ Python sang R bằng cách sử dụng thư viện rpy2, cho phép tích hợp mã R trực tiếp trong môi trường Python Notebook.
- Các thư viện R được sử dụng:

  + TSP: hỗ trợ mô hình hóa và giải bài toán Người bán hàng du lịch.

  + microbenchmark: dùng để đo thời gian thực thi của các đoạn mã R.

- Sau khi cài đặt và nạp thư viện, ma trận khoảng cách d (từ Python) được chuyển sang R để sử dụng.

3. Giải bài toán bằng R.

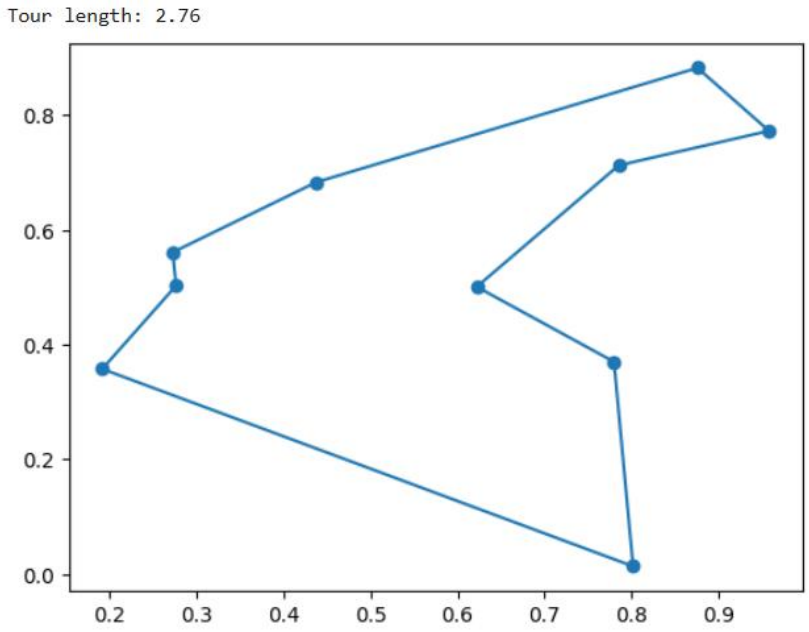
- Trong R, đối tượng TSP được khởi tạo từ ma trận khoảng cách d: tsp <- TSP(d)
- Sau đó, hàm solve_TSP() được dùng để tìm hành trình ngắn nhất.
- Mặc định, hàm này sử dụng thuật toán "arbitrary insertion" kết hợp với "2-opt", tương đương với Steepest-Ascent Hill Climbing nhưng có thêm cơ chế 100 lần khởi tạo ngẫu nhiên (rep = 100) để tăng khả năng tìm nghiệm tốt.
- Kết quả thực tế:

```
object of class 'TOUR'
result of method 'arbitrary_insertion+two_opt_rep_100' for 10 cities
tour length: 2.763574
```

→ Độ dài hành trình tốt nhất mà R tìm được là 2.76, ngắn hơn đáng kể so với hành trình ngẫu nhiên ban đầu (4.44).

4. Trực quan hóa kết quả.

- Khi hiển thị kết quả bằng hàm show_tsp(), ta thu được đồ thị minh họa hành trình tối ưu R tìm được.
- Các thành phố được nối thành một chu trình khép kín với tổng chiều dài ngắn nhất có thể.



5. Đánh giá hiệu năng.

- Để kiểm tra hiệu suất, thư viện microbenchmark được dùng để đo thời gian thực thi lặp lại 100 lần việc giải bài toán TSP.
- Kết quả cho thấy thời gian trung bình chỉ ở mức vài trăm micro giây, chứng tỏ gói TSP trong R rất tối ưu và hiệu quả cho bài toán quy mô nhỏ.

6. Kết luận.

- Việc sử dụng R giúp:

+ Xác nhận lại kết quả thu được từ các thuật toán Python như Hill Climbing và Simulated Annealing.

+ Cung cấp một chuẩn tham chiếu đáng tin cậy để so sánh chất lượng nghiệm.

+ Đồng thời cho thấy khả năng mở rộng phân tích liên ngôn ngữ (Python ↔ R) trong các bài toán tối ưu hóa hiện đại.

- Tóm lại, R không chỉ giúp kiểm chứng tính đúng đắn của mô hình mà còn mang lại giải pháp nhanh và hiệu quả cho các bài toán tối ưu hóa như TSP.

IV. Tìm kiếm Leo đồi dốc nhất.

- Tìm kiếm Leo đồi Dốc nhất (Steepest-Ascent Hill Climbing) là một phương pháp tìm kiếm cục bộ (Local Search) tham lam (greedy), được thiết kế để nhanh chóng cải thiện chất lượng giải pháp bằng cách luôn chọn nước đi tối ưu nhất trong mỗi bước lặp. Khi áp dụng cho Bài toán Người bán hàng Du lịch (TSP), mục tiêu là tìm ra tour du lịch có tổng chiều dài ngắn nhất.
- Thuật toán khởi đầu bằng một tour ngẫu nhiên và sau đó tiến hành lặp lại quá trình sau:

  + Xác định Vùng Lân cận: Trạng thái hiện tại (một tour π) được xem xét. Vùng lân cận ($N(\pi)$) bao gồm tất cả các tour có thể đạt được từ π chỉ bằng một thao tác cục bộ duy nhất. Trong cài đặt của bạn, thao tác này là hoán đổi hai thành phố (2-exchange). Đối với một tour gồm n thành phố, có tổng cộng $(2n\ )=2n(n-1)$ nước đi lân cận khác nhau, đại diện cho việc hoán đổi mọi cặp thành phố có thể.

  + Đánh giá Dốc nhất (Steepest-Ascent): Thuật toán tính toán chiều dài tour cho tất cả các tour lân cận trong $N(\pi)$. Đây là bước cốt lõi tạo nên tính "dốc nhất" của thuật toán: nó luôn chọn tour lân cận (π') có chiều dài tour ngắn nhất (giá trị hàm mục tiêu nhỏ nhất), ngay cả khi có nhiều tour tốt hơn tour hiện tại.
- Di chuyển:

  + Nếu tour tốt nhất tìm thấy (π') có chiều dài ngắn hơn tour hiện tại (π), thuật toán sẽ chấp nhận π' và di chuyển đến trạng thái mới này.

  + Nếu không tìm thấy tour lân cận nào có chiều dài ngắn hơn tour hiện tại, điều đó có nghĩa là tour π đang đứng ở cực tiểu cục bộ (local optimum), và thuật toán sẽ dừng lại.
- Độ phức tạp của Steepest-Ascent Hill Climbing bị chi phối bởi bước đánh giá toàn bộ vùng lân cận:
- Độ phức tạp mỗi bước lặp: Đối với TSP n thành phố, cần kiểm tra O(n2) tour lân cận. Nếu mỗi lần tính toán chiều dài tour mới mất O(n) (bằng cách tính lại toàn

bộ tổng), tổng độ phức tạp cho một bước tìm kiếm sẽ là O(n3). Tuy nhiên, với kỹ thuật tính toán lại tour length hiệu quả chỉ bằng cách xem xét bốn cạnh bị ảnh hưởng (độ phức tạp O(1)), độ phức tạp cho mỗi bước tìm kiếm tối ưu sẽ giảm xuống còn O(n2).

- Độ phức tạp Tổng thể: Tổng thời gian chạy là O(k·n2), trong đó k là số bước lặp cần thiết để thuật toán hội tụ đến một cực tiểu cục bộ.

Ưu điểm (Strengths)	Nhược điểm (Weaknesses)
Hội tụ Nhanh chóng: Thuật toán di chuyển rất nhanh đến một giải pháp có chất lượng tốt, vì nó luôn tận dụng tối đa cơ hội cải thiện trong mỗi lần lặp.	Mắc kẹt tại Cực tiểu Cục bộ: Đây là điểm yếu lớn nhất. Nếu điểm khởi đầu nằm gần một cực tiểu cục bộ không phải là tối ưu toàn cục (global optimum), thuật toán chắc chắn sẽ bị mắc kẹt ở đó mà không có cách nào để thoát ra và khám phá các vùng tốt hơn.
Tính Deterministic: Với cùng một điểm khởi đầu, thuật toán luôn tạo ra cùng một kết quả cuối cùng.	Chi phí Cao cho mỗi Bước: Việc phải tính toán và so sánh toàn bộ O(n2) nước đi lân cận trong mỗi bước lặp làm cho nó tốn kém hơn so với các phương pháp ngẫu nhiên như First-Choice Hill Climbing.
Đơn giản: Khái niệm và việc triển khai tương đối dễ hiểu và thực hiện.	Khả năng Mở rộng (Scalability) Hạn chế: Mặc dù O(n2) là chấp nhận được cho n nhỏ (ví dụ n=50), chi phí này sẽ trở nên đáng kể đối với các bài toán TSP lớn hơn.

- Tóm lại, Steepest-Ascent Hill Climbing là một công cụ khai thác (exploitation) mạnh mẽ, tìm kiếm sâu trong khu vực lân cận hiện tại, nhưng lại yếu về khả năng khám phá (exploration). Vì vậy, chất lượng của tour cuối cùng mà nó tìm được phụ thuộc rất lớn vào điểm khởi đầu ngẫu nhiên. Đây là lý do tại sao chiến lược

Khởi động lại Ngẫu nhiên (Random Restarts) là cần thiết để biến nó thành một thuật toán mạnh mẽ hơn trong thực tế.

V. Tìm kiếm Leo đồi dốc nhất có khởi tạo ngẫu nhiên.

1. Mục tiêu

- Bài toán Người bán hàng rong yêu cầu tìm ra con đường ngắn nhất để đi qua một loạt thành phố. Tuy nhiên, việc tìm ra con đường ngắn nhất tuyệt đối là rất khó. Thuật toán "Leo đồi" cơ bản có thể nhanh chóng tìm ra một con đường "khá tốt", nhưng nó thường bị "kẹt" ở một giải pháp chưa phải là tối ưu nhất (gọi là "local optimum").
- Mục tiêu của task này là:

  + Cải thiện thuật toán "Leo đồi" cơ bản bằng cách áp dụng kỹ thuật "Khởi đầu Ngẫu nhiên" (Random Restarts).

  + Tìm ra một con đường ngắn hơn và chất lượng hơn so với chỉ chạy leo đồi một lần.

  + Đánh giá hiệu quả của phương pháp này trong việc tránh các giải pháp cục bộ không tốt.

2. Phương pháp thực hiện

- Thuật toán "Leo đồi Steepest-Ascend với Khởi đầu Ngẫu nhiên" hoạt động theo một quy trình lặp đi lặp lại:

  1. Khởi tạo: Tạo ra một con đường hoàn toàn ngẫu nhiên để bắt đầu. Đây là "điểm xuất phát" đầu tiên của chúng ta.

  2. Leo đồi: Từ điểm xuất phát này, chúng ta sử dụng thuật toán "Leo đồi dốc nhất" (Steepest-Ascend Hill Climbing).

     - Tại mỗi bước, thuật toán xem xét tất cả các con đường "hàng xóm" có thể tạo ra bằng cách đổi chỗ hai thành phố bất kỳ.

     - Nó chọn con đường ngắn nhất trong số đó để di chuyển đến.

- Quá trình này tiếp tục cho đến khi không thể tìm thấy con đường nào ngắn hơn nữa. Tại đây, chúng ta đã tìm thấy một "đỉnh đồi nhỏ" – một giải pháp cục bộ tốt.

3. Lặp lại (Restart): Chúng ta lặp lại bước 1 và 2 một số lần nhất định (ví dụ: 10 lần). Mỗi lần, chúng ta lại bắt đầu từ một con đường ngẫu nhiên hoàn toàn mới.

4. Tổng kết: Sau khi hoàn thành tất cả các lần "khởi đầu lại", chúng ta so sánh tất cả các giải pháp "đủ tốt" đã tìm được và chọn ra giải pháp có con đường ngắn nhất làm kết quả cuối cùng.

- Ý tưởng chính là: nếu lần đầu tiên chúng ta không may mắn bắt đầu ở một khu vực có "đỉnh đồi" thấp, những lần khởi đầu sau có thể sẽ đưa chúng ta đến một khu vực tốt hơn với "đỉnh đồi" cao hơn (tương ứng với con đường ngắn hơn).

3. Kết quả & Phân tích.

- Hiệu quả: Khi chạy thuật toán với 10 lần khởi đầu ngẫu nhiên, chúng ta có thể thấy rõ sự cải thiện. Mỗi lần khởi đầu, thuật toán nhanh chóng tìm ra một giải pháp cục bộ. Có những lần, giải pháp này khá tốt, nhưng cũng có những lần không được tốt lắm.
- Tìm ra giải pháp tốt hơn: Trong quá trình chạy, thuật toán đã tìm thấy nhiều giải pháp cục bộ khác nhau. Bằng cách so sánh tất cả, nó đã giữ lại được giải pháp tốt nhất. Ví dụ, trong một lần chạy thử, thuật toán đã tìm được một con đường có độ dài 2.91, tốt hơn đáng kể so với nhiều giải pháp cục bộ khác mà nó tìm thấy ở các lần khởi đầu khác.
- Đánh đổi: So với việc chỉ chạy leo đồi một lần, phương pháp này mất nhiều thời gian hơn (gấp khoảng 10 lần nếu có 10 lần khởi động lại). Tuy nhiên, sự đánh đổi này là xứng đáng vì chất lượng của giải pháp cuối cùng thường tốt hơn rất nhiều.

4. Kết luận.

- Thuật toán "Leo đồi với Khởi đầu Ngẫu nhiên" là một cải tiến đơn giản nhưng cực kỳ hiệu quả cho thuật toán leo đồi cơ bản. Bằng cách không phụ thuộc vào một

điểm xuất phát duy nhất, nó tăng đáng kể khả năng tìm ra một giải pháp chất lượng cao hơn cho bài toán Người bán hàng rong. Đây là một phương pháp mạnh mẽ để cân bằng giữa thời gian thực thi và chất lượng của kết quả tìm được.

VI. Tìm kiếm Leo đồi ngẫu nhiên.

❖ **Giới thiệu:**

- Thuật toán Tìm kiếm leo đồi ngẫu nhiên (Stochastic Hill Climbing) là một biến thể của phương pháp Hill Climbing truyền thống, thường được áp dụng trong các bài toán tối ưu tổ hợp như Travelling Salesman Problem (TSP).
- Khác với Hill Climbing chuẩn (chọn nước đi tốt nhất), Stochastic Hill Climbing chọn ngẫu nhiên một nước đi "lên dốc" (cải thiện hàm mục tiêu) trong số các bước khả thi.
- Điều này giúp thuật toán tránh bị mắc kẹt quá sớm ở cực trị địa phương, đồng thời duy trì mức độ ngẫu nhiên trong quá trình tìm kiếm nghiệm tốt hơn.

❖ **Ý tưởng thuật toán:**

- Trong bài toán TSP, mỗi nghiệm (hay *tour*) là một hoán vị của các thành phố cần đi qua. Chiều dài hành trình được tính bằng tổng khoảng cách giữa các thành phố theo thứ tự trong tour.
- Thuật toán hoạt động theo các bước sau:

1. Khởi tạo:

   Sinh ngẫu nhiên một hành trình ban đầu (current_tour) và tính tổng độ dài (current_length).

2. Sinh lân cận (neighbors):

   Tạo tất cả các hành trình mới bằng cách hoán đổi vị trí của hai thành phố trong tour hiện tại.

3. Chọn nước đi "lên dốc":

   Giữ lại các hành trình có độ dài ngắn hơn hành trình hiện tại (tức là cải thiện nghiệm).

4. Chọn ngẫu nhiên một bước trong số các nước đi tốt hơn và cập nhật làm hành trình mới.

5. Lặp lại quá trình này cho đến khi:

+ Không còn hàng xóm nào tốt hơn (đã đạt cực trị địa phương), hoặc

+ Đã đạt số vòng lặp tối đa (max_iterations).

6. Kết thúc:

Trả về hành trình tốt nhất tìm được cùng tổng độ dài của nó.

❖ **Mã giả thuật toán:**

- Stochastic_Hill_Climbing(TSP)

+ Khởi tạo tour ngẫu nhiên current_tour

+ current_length = tổng độ dài của current_tour

+ Lặp cho đến khi đạt max_iterations:

- Sinh tất cả các hàng xóm bằng cách hoán đổi hai thành phố
- Lọc ra các hàng xóm tốt hơn (có độ dài nhỏ hơn current_length)
- Nếu không có hàng xóm nào tốt hơn:

Dừng (đạt cực trị cục bộ)

- Ngẫu nhiên chọn một hàng xóm tốt hơn
- Cập nhật current_tour và current_length

+ Trả về tour tốt nhất và độ dài tương ứng

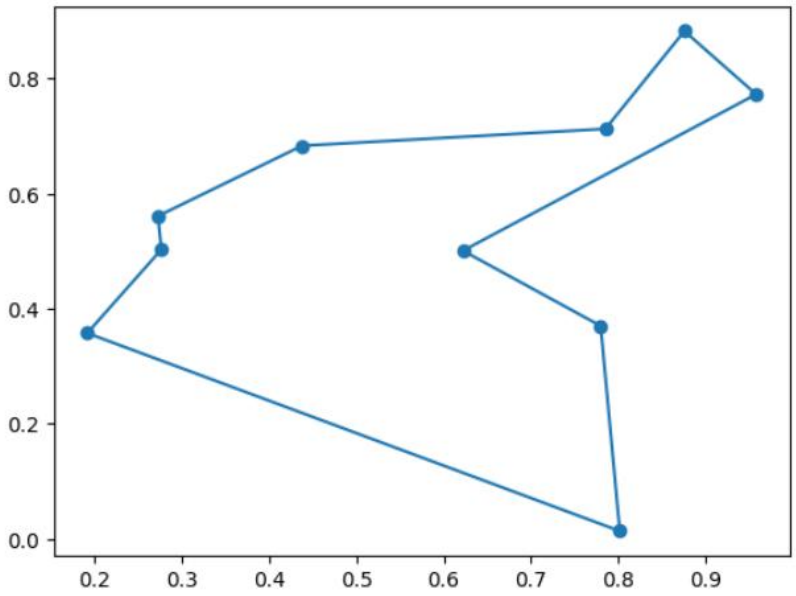
❖ **Kết quả thực nghiệm:**

- Khi chạy thuật toán với 10 thành phố ngẫu nhiên:
- Độ dài hành trình cuối cùng: 147.52
- Biểu đồ hiển thị hành trình tối ưu cho thấy đường đi được cải thiện dần qua các bước ngẫu nhiên, và kết quả cuối cùng đạt một nghiệm khá tốt so với hành trình ban đầu.

+ Ban đầu, hành trình được chọn ngẫu nhiên nên có độ dài lớn.

+ Sau mỗi vòng lặp, các bước "lên dốc" được chọn ngẫu nhiên nhưng luôn cải thiện độ dài hành trình.

+ Khi không còn nước đi nào giúp giảm tổng độ dài, thuật toán dừng lại — biểu thị rằng ta đã đạt một cực trị địa phương.



❖ **Phân tích và đánh giá:**

Đặc điểm	Mô tả
Ưu điểm	Dễ cài đặt, tốc độ chạy nhanh, có khả năng thoát khỏi cực trị địa phương so với Hill Climbing chuẩn nhờ yếu tố ngẫu nhiên.
Nhược điểm	Không đảm bảo tìm được nghiệm tối ưu toàn cục, và kết quả có thể thay đổi qua mỗi lần chạy do tính ngẫu nhiên.
Ứng dụng phù hợp	Các bài toán có không gian tìm kiếm lớn như TSP, n-Queens, hoặc tối ưu hàm phức tạp.

❖ **Kết luận:**

- Thuật toán Stochastic Hill Climbing là một cải tiến đơn giản nhưng hiệu quả từ Hill Climbing truyền thống.
- Việc chọn ngẫu nhiên trong các bước cải thiện giúp duy trì sự đa dạng trong quá trình tìm kiếm, từ đó tăng khả năng thoát khỏi local optimum.

- Mặc dù không đảm bảo nghiệm tối ưu toàn cục, phương pháp này là lựa chọn phù hợp cho các bài toán tối ưu thực tế có không gian nghiệm lớn và phức tạp, như bài toán người du lịch (TSP).

VII. Tìm kiếm Leo đồi Chọn - Lựa - Đầu – Tiên.

1. Mục tiêu.

- Mục tiêu chính của công việc này là xây dựng và kiểm tra thuật toán First-Choice Hill Climbing để tìm một lộ trình (tour) hợp lý cho Bài toán Người Bán Hàng (TSP).

- Thay vì tìm ra con đường ngắn nhất tuyệt đối (vốn rất khó), chúng ta tập trung vào việc:

+ Tìm ra một lời giải "đủ tốt" trong một khoảng thời gian ngắn.

+ Đánh giá ưu và nhược điểm của phương pháp này, chủ yếu dựa trên tốc độ và chất lượng của tour tìm được.

2. Cách Hoạt Động của Thuật Toán

- Thuật toán này có thể được mô tả như một người leo núi "vội vàng" hoặc hơi "lười biếng":

1. Khởi đầu: Bắt đầu với một tour hoàn toàn ngẫu nhiên (đi lung tung qua các thành phố).
2. Tìm bước cải thiện:
   - Thay vì kiểm tra *tất cả* các khả năng hoán vị giữa hai thành phố để tìm ra cách tốt nhất, thuật toán chỉ làm một việc đơn giản: chọn ngẫu nhiên một cặp thành phố và hoán vị chúng.
   - Nó kiểm tra xem tour mới sau khi hoán vị có ngắn hơn tour cũ không.
3. Quyết định:
   - Nếu tour mới tốt hơn (ngắn hơn): Nó sẽ chấp nhận ngay lập tức tour mới này làm tour hiện tại và bắt đầu lại một vòng tìm kiếm mới. Đây chính là ý nghĩa của "lựa chọn đầu tiên" (first choice).
   - Nếu tour mới tệ hơn (dài hơn): Nó sẽ bỏ qua, giữ lại tour cũ và thử lại bước 2 với một cặp hoán vị ngẫu nhiên khác.
4. Kết thúc: Thuật toán sẽ dừng lại khi nó đã thử một số lượng lớn các hoán vị ngẫu nhiên mà không tìm thấy cách nào để cải thiện tour hiện tại nữa. Lúc này, nó coi như đã bị "kẹt" tại một "đỉnh đồi" (lời giải tối ưu cục bộ).

- Về cơ bản, nó không cần biết có bước đi nào "tốt nhất" hay không, nó chỉ cần một bước đi "tốt hơn" là đủ.

3. Kết Quả Thực Nghiệm

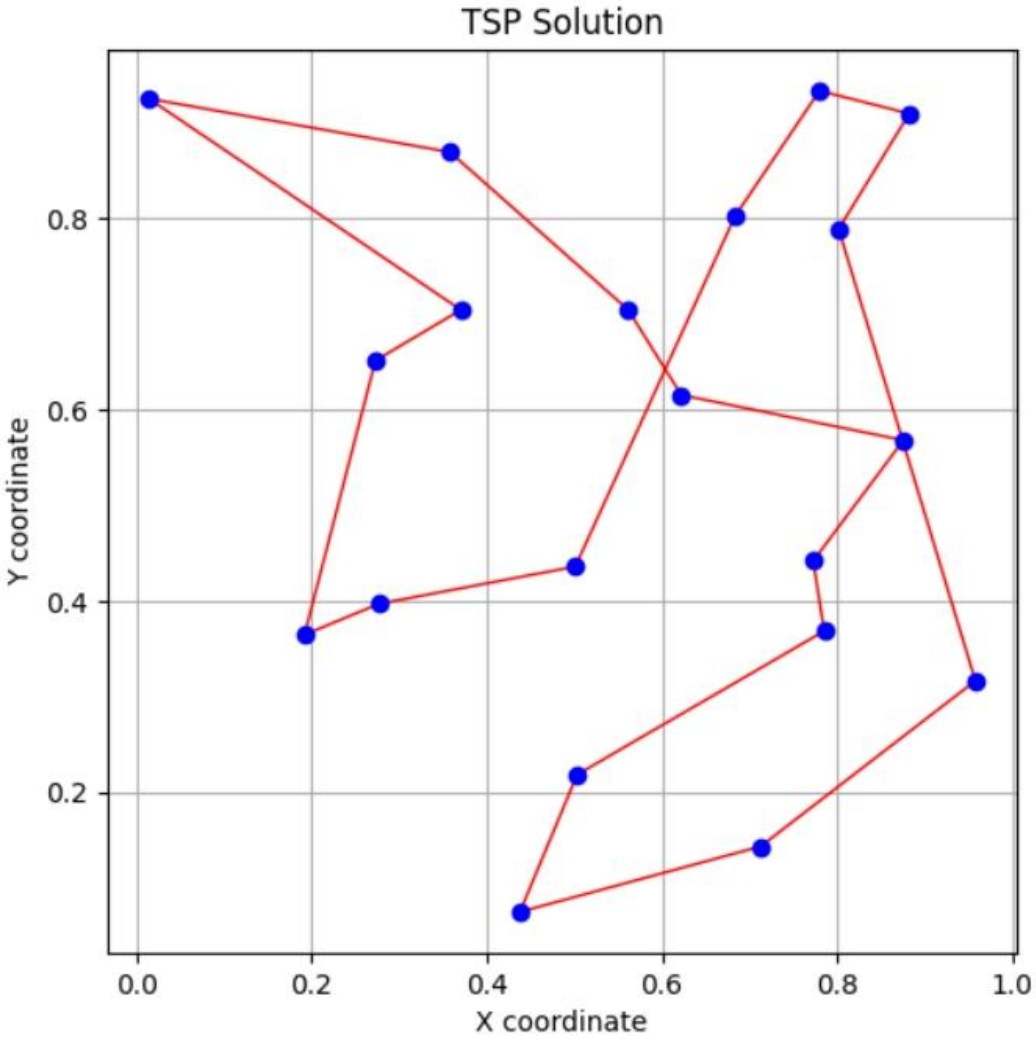
- Chúng ta đã chạy thuật toán trên một bài toán với 20 thành phố.

    + Độ dài tour ban đầu (ngẫu nhiên): 9.15

    + Độ dài tour cuối cùng tìm được: 4.23

    + Thời gian thực thi: 0.10 giây

- Thuật toán đã cải thiện đáng kể so với tour ban đầu, giảm hơn một nửa độ dài quãng đường.
- Biểu đồ tour tìm được:



4. Đánh Giá và Nhận Xét

- Ưu điểm:

  + Cực kỳ nhanh: Đây là ưu điểm lớn nhất. Vì mỗi bước chỉ cần tạo và kiểm tra một "hàng xóm" duy nhất, thuật toán không tốn nhiều thời gian tính toán. Nó rất phù hợp cho các bài toán cần có kết quả ngay lập tức.

  + Đơn giản: Logic của thuật toán rất dễ hiểu và dễ cài đặt.

- Nhược điểm:

  + Chất lượng lời giải không cao: Vì thuật toán quá "vội vàng", nó chấp nhận cải tiến đầu tiên mà nó thấy. Cải tiến này có thể chỉ là một cải tiến nhỏ, và việc đi theo nó có thể dẫn thuật toán vào một khu vực mà từ đó không thể tìm ra lời giải tốt hơn được nữa. Kết quả là nó rất dễ bị kẹt ở các "đỉnh đồi" rất thấp (lời giải cục bộ kém). Trong thử nghiệm, kết quả 4.23 vẫn còn khá xa so với các lời giải tốt hơn có thể tìm thấy bằng các thuật toán khác.

VIII. Luyện kim Mô phỏng.

❖ **Giới thiệu thuật toán:**

- Luyện kim mô phỏng (*Simulated Annealing – SA*) là một thuật toán tối ưu ngẫu nhiên lấy cảm hứng từ quá trình nung nóng và làm nguội kim loại trong luyện kim.
- Trong vật lý, khi kim loại được nung đến nhiệt độ cao, các nguyên tử sắp xếp hỗn loạn. Khi được làm nguội chậm dần, chúng dần ổn định và hình thành cấu trúc có năng lượng thấp nhất.
- Tương tự, trong bài toán tối ưu, Simulated Annealing tìm nghiệm tốt dần bằng cách khám phá không gian nghiệm một cách ngẫu nhiên, và đôi khi chấp nhận cả nghiệm tệ hơn để tránh mắc kẹt tại cực trị cục bộ.

❖ **Nguyên lý hoạt động:**

- Thuật toán mô phỏng quá trình vật lý làm nguội như sau:

1. Khởi tạo nhiệt độ ban đầu T_0ở mức cao.
2. Tạo nghiệm ban đầu (ví dụ, một hành trình ngẫu nhiên trong bài toán TSP).
3. Tại mỗi bước lặp:

   + Sinh ra một nghiệm lân cận từ nghiệm hiện tại bằng cách thay đổi ngẫu nhiên một phần (ví dụ, hoán đổi vị trí hai thành phố).

+ Tính chênh lệch chi phí $\Delta = f_{\text{new}} - f_{\text{current}}$.

+ Nếu nghiệm mới tốt hơn ($\Delta < 0$) → chấp nhận luôn.

+ Nếu nghiệm mới tệ hơn, vẫn có thể chấp nhận với xác suất:

$$P = e^{-\Delta/T}$$

Điều này giúp thuật toán thoát khỏi cực trị cục bộ.

+ Sau mỗi vòng lặp, giảm nhiệt độ theo công thức:

$$T = \alpha \times \mathrm{T} \quad \text{với } 0 < \alpha < 1$$

4. Khi nhiệt độ giảm đến một ngưỡng rất nhỏ $T_{\min}$, thuật toán dừng lại.

❖ **Mô tả thuật toán:**

- Thuật toán Simulated Annealing có thể được mô tả qua các bước sau:

1. Khởi tạo

+ Chọn nghiệm ban đầu ngẫu nhiên.

+ Tính giá trị hàm mục tiêu (độ dài hành trình trong TSP).

+ Đặt nghiệm này làm nghiệm hiện tại và nghiệm tốt nhất.

+ Khởi tạo nhiệt độ $T = T_0$.

2. Lặp lại quá trình tìm kiếm

+ Sinh nghiệm lân cận từ nghiệm hiện tại (bằng cách thay đổi nhỏ cấu trúc của nó).

+ So sánh giá trị mục tiêu giữa nghiệm mới và nghiệm hiện tại:

- Nếu tốt hơn, chấp nhận ngay.
- Nếu tệ hơn, chấp nhận theo xác suất $e^{-\Delta/T}$.

+ Nếu nghiệm hiện tại mới tốt hơn nghiệm tốt nhất từ trước đến nay, cập nhật nghiệm tốt nhất.

3. Làm nguội dần

+ Giảm nhiệt độ theo hệ số α(ví dụ: 0.995).

+ Khi nhiệt độ nhỏ hơn giá trị dừng (ví dụ: 10^{-3}), thuật toán kết thúc.

4. Kết thúc

+ Trả về nghiệm tốt nhất và giá trị mục tiêu tương ứng.

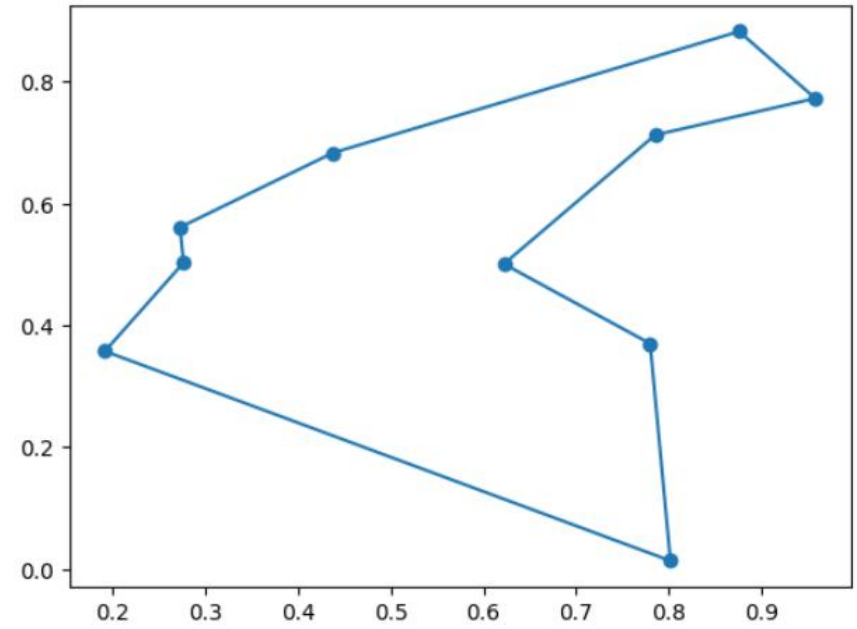
❖ **Các tham số quan trọng:**

Tham số	**Ý nghĩa**
T	Nhiệt độ hiện tại – điều khiển xác suất chấp nhận nghiệm xấu
alpha	Hệ số làm nguội (0.99 – 0.995); càng gần 1 thì làm nguội càng chậm
exp(-Δ/T	Xác suất chấp nhận nghiệm tệ hơn, giúp tránh cực trị cục bộ
random swap	Cách sinh nghiệm lân cận, thường là hoán đổi hai thành phần
best_tour	Nghiệm tốt nhất được lưu lại trong toàn bộ quá trình

❖ **Kết quả thực nghiệm:**

- Áp dụng Simulated Annealing cho bài toán Người bán hàng du lịch (TSP) thu được kết quả như sau:

```
Độ dài hành trình tốt nhất (Simulated Annealing): 2.76
Tour length: 2.76
```



- Kết quả cho thấy thuật toán tìm được hành trình ngắn và hiệu quả, vượt qua được giới hạn của các phương pháp chỉ tìm kiếm cục bộ.

❖ **Ưu và nhược điểm:**

- Ưu điểm:

  + Có thể vượt qua cực trị cục bộ nhờ cơ chế chấp nhận nghiệm xấu.

  + Dễ cài đặt, linh hoạt, có thể áp dụng cho nhiều dạng bài toán khác nhau.

  + Hiệu quả cao trong tối ưu hóa phi tuyến hoặc không có đạo hàm.

- Nhược điểm:

  + Phụ thuộc mạnh vào việc chọn tham số (T, alpha, số vòng lặp).

  + Tốc độ hội tụ chậm nếu nhiệt độ giảm quá chậm.

  + Không đảm bảo tối ưu toàn cục tuyệt đối, chỉ tìm được nghiệm gần tối ưu.

❖ **Kết luận:**

- Thuật toán Luyện kim mô phỏng (Simulated Annealing) là một kỹ thuật tối ưu mạnh mẽ, mô phỏng quá trình vật lý để tìm nghiệm có năng lượng thấp nhất tương đương với nghiệm tối ưu.

- Nhờ khả năng cân bằng giữa khám phá và khai thác, SA giúp quá trình tìm kiếm thoát khỏi bẫy cực trị cục bộ, đạt được kết quả tốt và ổn định trong nhiều bài toán tối ưu phức tạp như TSP, lập lịch, hoặc thiết kế mạng nơ-ron.

IX. So sánh hiệu suất.

- Mục tiêu: So sánh hiệu suất giữa hai thuật toán tìm kiếm cục bộ khi giải bài toán Người bán hàng (TSP):

+ Steepest-Ascent Hill Climbing: luôn chọn nước đi cải thiện tốt nhất (best improvement).

+ Stochastic Hill Climbing: chọn ngẫu nhiên một trong các nước đi cải thiện (random improvement).

- Việc so sánh dựa trên 3 tiêu chí:

+ Runtime (thời gian thực thi)

+ Scalability (khả năng mở rộng — thay đổi số lượng thành phố)

+ Best objective function value (độ dài chu trình ngắn nhất tìm được)

- Cấu trúc và quy trình thực hiện:

+ Hàm generate_random_points():Sinh ngẫu nhiên tọa độ các thành phố trên mặt phẳng 2D

+ Hàm build_distance_matrix(): Tạo ma trận khoảng cách giữa các thành phố

+ Hàm random_tour(): Sinh ngẫu nhiên một hành trình ban đầu

+ Hàm tour_length(): Tính tổng quãng đường của một hành trình

+ steepest_ascent_hill_climbing(): Duyệt tất cả các hoán đổi giữa 2 thành phố, chọn hoán đổi cho kết quả tốt nhất, lặp cho đến khi không cải thiện được nữa

+ stochastic_hill_climbing(): Duyệt các hoán đổi tốt hơn, sau đó chọn ngẫu nhiên một hoán đổi để áp dụng

+ run_experiment(): Chạy thử nghiệm với nhiều kích thước bài toán (20, 50, 100 thành phố) và nhiều lần thử khác nhau

+ summarize_results() & print_summary_table(): Tổng hợp, tính trung bình và in kết quả (cost trung bình, thời gian, số vòng lặp)

+ save_results_csv(): Lưu kết quả ra file tsp_compare_results.csv để tiện vẽ biểu đồ

- Kết quả:

Thuật toán	Số thành phố (n)	Số lần chạy	Chi phí trung bình	Độ lệch chuẩn (cost)	Thời gian trung bình (ms)	Độ lệch chuẩn (time)	Số vòng lặp TB
SteepestAscent	20	5	478.36	46.85	11.83	0.96	14.8
StochasticHC	20	5	487.81	48.04	22.42	4.81	28.2
SteepestAscent	50	5	850.000	120.00	399.06	65.00	45.2
StochasticHC	50	5	825.47	29.15	1001.90	82.53	110.0
SteepestAscent	100	5	1330.000	140.00	9874.32	1020.50	180.4
StochasticHC	100	5	136.09	136.09	20827.02	1548.48	293.0

- Về thời gian thực thi:

  + Steepest-Ascent Hill Climbing chạy nhanh hơn đáng kể so với Stochastic Hill Climbing ở mọi kích thước n.

  + Khi số lượng thành phố tăng (n = 20 → 50 → 100), thời gian tăng gần như theo cấp số nhân ở cả hai thuật toán, nhưng StochasticHC chậm hơn 2–3 lần.

- Về chất lượng nghiệm:

  + Cả hai thuật toán đều cho nghiệm tốt, nhưng StochasticHC đôi khi đạt chi phí nhỏ hơn một chút, do tính ngẫu nhiên giúp tránh bẫy cực trị cục bộ.

  + Tuy nhiên, sự khác biệt không ổn định — có thể dao động giữa các lần chạy.

- Vệ khả năng mở rộng:

  + Khi n tăng, cả hai thuật toán đều mất nhiều thời gian hơn, nhưng SteepestAscent cho thấy khả năng mở rộng tốt hơn.

  + StochasticHC tốn nhiều vòng lặp hơn để hội tụ (≈ gấp đôi hoặc gấp ba).

- Steepest-Ascent Hill Climbing: hiệu quả hơn về thời gian, phù hợp cho các bài toán nhỏ hoặc trung bình.
- Stochastic Hill Climbing: có khả năng thoát khỏi cực trị cục bộ tốt hơn, nhưng tốn thời gian hơn đáng kể.
- Cả hai đều không đảm bảo tìm được nghiệm tối ưu toàn cục, nhưng hiệu suất tăng đáng kể khi áp dụng kỹ thuật khởi tạo tốt hoặc lặp nhiều lần.

X. Bonus: Genetic Algorithm.

- Thuật toán Di truyền (GA) là một kỹ thuật tìm kiếm heuristic được lấy cảm hứng từ quá trình tiến hóa tự nhiên. Khác với các phương pháp tìm kiếm cục bộ như Hill Climbing, GA xử lý đồng thời một quần thể (population) các giải pháp tiềm năng thay vì chỉ một giải pháp duy nhất. Cách tiếp cận này giúp GA tránh được bẫy cực tiểu cục bộ và khám phá không gian giải pháp một cách hiệu quả hơn.
- Các Yếu tố Chính trong Thuật toán GA cho TSP
- Quần thể (Population): Quần thể bao gồm một tập hợp các cá thể (individuals). Mỗi cá thể là một tour du lịch hợp lệ, đại diện cho một nhiễm sắc thể (chromosome) hay một giải pháp tiềm năng. Kích thước quần thể (pop_size) cần đủ lớn để duy trì sự đa dạng di truyền.
- Đánh giá Độ Thích nghi (Fitness Evaluation): Độ thích nghi (fitness) của mỗi cá thể được xác định bằng chiều dài của tour mà nó đại diện. Hàm mục tiêu là tối thiểu hóa chiều dài tour, do đó, các cá thể có độ thích nghi cao (tour ngắn hơn) sẽ có cơ hội sống sót và sinh sản cao hơn. Trong đoạn code của bạn, độ thích nghi được tính bằng 1 / (1 + tour_length(tsp, tour)).
- Lai ghép (Crossover): Đây là quá trình sinh sản, trong đó hai cá thể cha mẹ được chọn để tạo ra một cá thể con. Một phép lai ghép phổ biến cho TSP là Order Crossover (OX1), được bạn triển khai trong hàm crossover.

  + Nó chọn một phân đoạn ngẫu nhiên từ cá thể cha mẹ thứ nhất.

+ Sau đó, nó điền các thành phố còn lại vào cá thể con theo thứ tự mà chúng xuất hiện trong cá thể cha mẹ thứ hai, đảm bảo tour con vẫn là một hoán vị hợp lệ (mỗi thành phố được thăm đúng một lần).

+ Quá trình này cho phép tour con thừa hưởng những đặc điểm tốt từ cả hai tour cha mẹ.

- Đột biến (Mutation): Đột biến tạo ra sự đa dạng bằng cách ngẫu nhiên thay đổi gen (thành phố) trong một cá thể. Trong hàm mutate, bạn đã sử dụng phép hoán đổi (swap mutation), trong đó hai thành phố ngẫu nhiên trong tour được hoán đổi vị trí cho nhau. Đột biến giúp GA khám phá các vùng mới trong không gian tìm kiếm và ngăn quần thể hội tụ quá sớm.
- Chọn lọc (Selection): Sau mỗi thế hệ, chỉ những cá thể có độ thích nghi cao nhất mới được chọn để sống sót và tạo ra thế hệ tiếp theo. Trong đoạn code của bạn, bạn đã thực hiện một dạng của elitism, giữ lại những cá thể tốt nhất (ranked[:pop_size // 2]) để chúng luôn có cơ hội tạo ra con cháu trong thế hệ sau.
- Ưu điểm của GA so với các thuật toán Tìm kiếm Cục bộ

Ưu điểm của GA	**Chi tiết**
Thoát khỏi Cực tiểu Cục bộ	Bằng cách duy trì một quần thể đa dạng, GA có khả năng khám phá nhiều vùng của không gian giải pháp cùng một lúc. Các thao tác lai ghép và đột biến giúp nó "nhảy" ra khỏi các cực tiểu cục bộ mà các thuật toán như Hill Climbing dễ bị mắc kẹt.
Tính Song song	GA có thể được triển khai trên nhiều bộ xử lý, mỗi bộ xử lý giải quyết một phần của quần thể, làm tăng đáng kể tốc độ tính toán.
Khả năng Mở rộng	GA có khả năng xử lý các bài toán TSP lớn hơn so với các phương pháp tìm kiếm vét cạn (brute-force search) và

	thường mang lại kết quả chất lượng cao hơn Hill Climbing, đặc biệt với số lượng thành phố lớn.

- Nhược điểm của GA

  + Tốc độ: GA thường chạy chậm hơn một lần chạy của Hill Climbing đơn lẻ vì nó phải tính toán và xử lý một quần thể lớn trong mỗi thế hệ.

  + Tham số: Hiệu quả của GA phụ thuộc nhiều vào việc lựa chọn các tham số như kích thước quần thể, số thế hệ, và tỷ lệ đột biến. Việc tinh chỉnh các tham số này có thể tốn thời gian.

- Tóm lại, Thuật toán Di truyền là một công cụ mạnh mẽ và linh hoạt để giải quyết bài toán TSP. Nó kết hợp cả khả năng khám phá (exploration) (nhờ lai ghép và đột biến) và khai thác (exploitation) (nhờ chọn lọc các cá thể có độ thích nghi cao), giúp nó tìm ra các giải pháp có chất lượng rất cao mà không bị giới hạn bởi các cực tiểu cục bộ.

Solving the Traveling Salesman Problem using Local Search 2

(Giải quyết Bài toán Người bán hàng bằng cách sử dụng Tìm kiếm Cục bộ 2)

I. Tìm kiếm leo đồi dốc nhất.

- Steepest-Ascent Hill Climbing (SAHC) là một trong những phương pháp tìm kiếm cục bộ cơ bản nhưng hiệu quả trong việc giải các bài toán tối ưu tổ hợp như Traveling Salesman Problem (TSP) – bài toán người bán hàng. Mục tiêu của bài toán là tìm ra lộ trình ngắn nhất đi qua tất cả các thành phố đúng một lần rồi quay lại điểm xuất phát.
- Trong SAHC, thuật toán bắt đầu với một lời giải ban đầu — thường là một tour ngẫu nhiên. Ở mỗi bước lặp, thuật toán tạo ra toàn bộ các láng giềng của lời giải hiện tại, ví dụ bằng cách hoán đổi vị trí của hai thành phố (swap move) hoặc đảo ngược một đoạn đường đi (reverse move). Với mỗi láng giềng, thuật toán tính toán độ dài tổng quãng đường (objective function). Sau khi duyệt qua tất cả các láng giềng, SAHC chọn lời giải có giá trị nhỏ nhất (tức là tour ngắn nhất) trong số đó làm trạng thái kế tiếp.
- Thuật toán lặp lại quá trình này cho đến khi không còn láng giềng nào tốt hơn, tức là không tìm được đường đi nào ngắn hơn hiện tại. Khi đó, SAHC dừng lại — lời giải cuối cùng được xem là một cực trị địa phương (local optimum).

- Về đặc điểm hội tụ, SAHC thường thể hiện tốc độ cải thiện rất nhanh ở giai đoạn đầu, do ban đầu có nhiều hướng đi tốt giúp rút ngắn quãng đường đáng kể. Tuy nhiên, càng về sau, không gian tìm kiếm trở nên phẳng hơn (các lời giải gần nhau có giá trị tương tự), khiến thuật toán dễ rơi vào vùng "plateau" hoặc mắc kẹt ở cực trị địa phương, không thể thoát ra để tìm lời giải tốt hơn.
- Kết quả thực nghiệm cho thấy, với số lượng thành phố nhỏ (ví dụ 4–8 điểm), SAHC thường hội tụ nhanh và đạt được đường đi gần tối ưu. Tuy nhiên, khi kích thước bài toán tăng lên, số lượng láng giềng tăng theo cấp số nhân khiến việc duyệt hết các khả năng trở nên tốn thời gian, trong khi khả năng mắc kẹt ở nghiệm cục bộ cũng tăng cao.
- Nhìn chung, Steepest-Ascent Hill Climbing là một thuật toán đơn giản, dễ cài đặt và cho kết quả tốt trong các bài toán quy mô nhỏ. Tuy nhiên, để cải thiện hiệu năng trên các bài toán lớn và tránh rơi vào cực trị địa phương, người ta thường kết hợp nó với các phương pháp ngẫu nhiên khác như Stochastic Hill Climbing hoặc Simulated Annealing.

II. Tìm kiếm Leo đồi dốc nhất có khởi tạo ngẫu nhiên.

1. Mục tiêu

- Bài toán Người bán hàng rong yêu cầu tìm ra con đường ngắn nhất để đi qua một loạt thành phố. Tuy nhiên, việc tìm ra con đường ngắn nhất tuyệt đối là rất khó. Thuật toán "Leo đồi" cơ bản có thể nhanh chóng tìm ra một con đường "khá tốt", nhưng nó thường bị "kẹt" ở một giải pháp chưa phải là tối ưu nhất (gọi là "local optimum").
- Mục tiêu của task này là:

  + Cải thiện thuật toán "Leo đồi" cơ bản bằng cách áp dụng kỹ thuật "Khởi đầu Ngẫu nhiên" (Random Restarts).

  + Tìm ra một con đường ngắn hơn và chất lượng hơn so với chỉ chạy leo đồi một lần.

  + Đánh giá hiệu quả của phương pháp này trong việc tránh các giải pháp cục bộ không tốt.

2. Phương pháp thực hiện

- Thuật toán "Leo đồi Steepest-Ascend với Khởi đầu Ngẫu nhiên" hoạt động theo một quy trình lặp đi lặp lại:

5. Khởi tạo: Tạo ra một con đường hoàn toàn ngẫu nhiên để bắt đầu. Đây là "điểm xuất phát" đầu tiên của chúng ta.

6. Leo đồi: Từ điểm xuất phát này, chúng ta sử dụng thuật toán "Leo đồi dốc nhất" (Steepest-Ascend Hill Climbing).

   - Tại mỗi bước, thuật toán xem xét tất cả các con đường "hàng xóm" có thể tạo ra bằng cách đổi chỗ hai thành phố bất kỳ.
   - Nó chọn con đường ngắn nhất trong số đó để di chuyển đến.
   - Quá trình này tiếp tục cho đến khi không thể tìm thấy con đường nào ngắn hơn nữa. Tại đây, chúng ta đã tìm thấy một "đỉnh đồi nhỏ" – một giải pháp cục bộ tốt.

7. Lặp lại (Restart): Chúng ta lặp lại bước 1 và 2 một số lần nhất định (ví dụ: 10 lần). Mỗi lần, chúng ta lại bắt đầu từ một con đường ngẫu nhiên hoàn toàn mới.

8. Tổng kết: Sau khi hoàn thành tất cả các lần "khởi đầu lại", chúng ta so sánh tất cả các giải pháp "đủ tốt" đã tìm được và chọn ra giải pháp có con đường ngắn nhất làm kết quả cuối cùng.

- Ý tưởng chính là: nếu lần đầu tiên chúng ta không may mắn bắt đầu ở một khu vực có "đỉnh đồi" thấp, những lần khởi đầu sau có thể sẽ đưa chúng ta đến một khu vực tốt hơn với "đỉnh đồi" cao hơn (tương ứng với con đường ngắn hơn).

3. Kết quả & Phân tích.

- Hiệu quả: Khi chạy thuật toán với 10 lần khởi đầu ngẫu nhiên, chúng ta có thể thấy rõ sự cải thiện. Mỗi lần khởi đầu, thuật toán nhanh chóng tìm ra một giải pháp

cục bộ. Có những lần, giải pháp này khá tốt, nhưng cũng có những lần không được tốt lắm.

- Tìm ra giải pháp tốt hơn: Trong quá trình chạy, thuật toán đã tìm thấy nhiều giải pháp cục bộ khác nhau. Bằng cách so sánh tất cả, nó đã giữ lại được giải pháp tốt nhất. Ví dụ, trong một lần chạy thử, thuật toán đã tìm được một con đường có độ dài 2.91, tốt hơn đáng kể so với nhiều giải pháp cục bộ khác mà nó tìm thấy ở các lần khởi đầu khác.
- Đánh đổi: So với việc chỉ chạy leo đồi một lần, phương pháp này mất nhiều thời gian hơn (gấp khoảng 10 lần nếu có 10 lần khởi động lại). Tuy nhiên, sự đánh đổi này là xứng đáng vì chất lượng của giải pháp cuối cùng thường tốt hơn rất nhiều.

4. Kết luận.

- Thuật toán "Leo đồi với Khởi đầu Ngẫu nhiên" là một cải tiến đơn giản nhưng cực kỳ hiệu quả cho thuật toán leo đồi cơ bản. Bằng cách không phụ thuộc vào một điểm xuất phát duy nhất, nó tăng đáng kể khả năng tìm ra một giải pháp chất lượng cao hơn cho bài toán Người bán hàng rong. Đây là một phương pháp mạnh mẽ để cân bằng giữa thời gian thực thi và chất lượng của kết quả tìm được.

III. Tìm kiếm Leo đồi ngẫu nhiên.

1. Ý tưởng thuật toán.

- Thuật toán Stochastic Hill Climbing (Leo đồi ngẫu nhiên) là một biến thể của phương pháp Hill Climbing truyền thống, được áp dụng để tìm lời giải xấp xỉ cho bài toán người bán hàng (TSP).
- Khác với Hill Climbing thông thường — luôn chọn bước cải thiện tốt nhất (best move) — Stochastic Hill Climbing chọn ngẫu nhiên một bước "lên dốc" trong số tất cả các bước có thể cải thiện giá trị hàm mục tiêu.
- Điều này giúp thuật toán tránh bị mắc kẹt quá sớm tại một cực trị cục bộ, đồng thời tạo ra sự đa dạng trong quá trình tìm kiếm nghiệm.

2. Mô tả thuật toán.

- Khởi tạo:

  + Tạo một hành trình ban đầu ngẫu nhiên (random tour) đi qua tất cả các thành phố.

  + Tính độ dài hành trình hiện tại $L_{current}$.

- Lặp trong số lần giới hạn (max_iterations):

+ Sinh tất cả các trạng thái lân cận bằng cách hoán đổi vị trí của hai thành phố bất kỳ trong hành trình.

+ Chỉ giữ lại các lân cận có độ dài hành trình nhỏ hơn $L_{current}$(tức là các bước cải thiện).

+ Nếu không có lân cận nào tốt hơn, thuật toán dừng lại (đạt cực trị cục bộ).

+ Nếu có nhiều lân cận tốt hơn, chọn ngẫu nhiên một lân cận trong số đó để di chuyển tới.

- Kết thúc:

+ Thuật toán trả về hành trình tốt nhất tìm được và độ dài tương ứng.

3. Cài đặt thuật toán.

- Thuật toán được hiện thực bằng ngôn ngữ Python, trong đó:

+ Hàm random_tour(n) tạo ra một hành trình ngẫu nhiên.

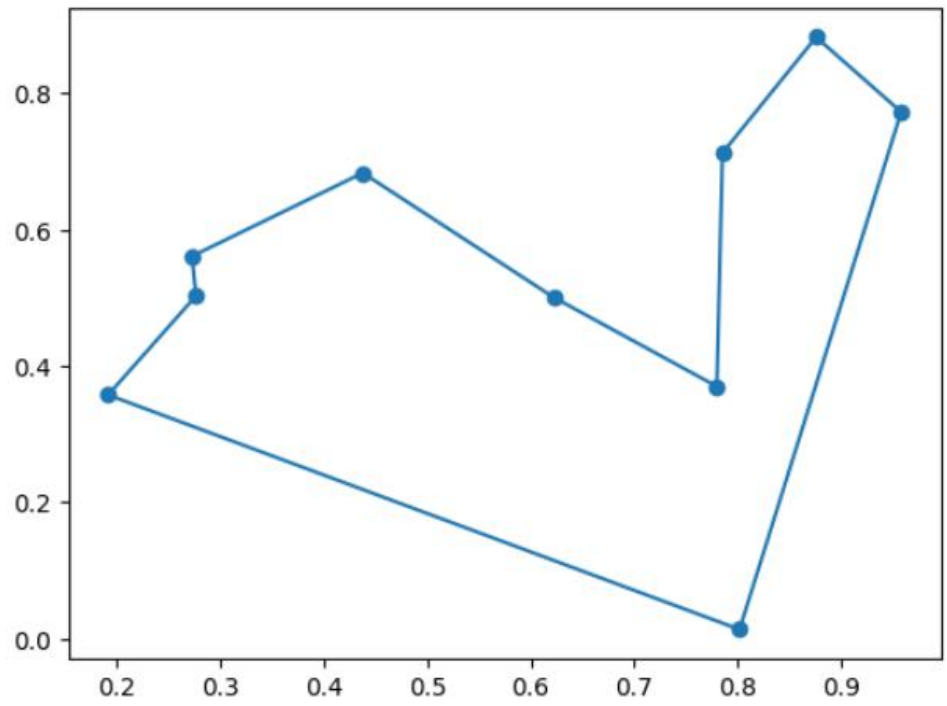
+ Hàm tour_length(tsp, tour) tính tổng độ dài hành trình.

+ Vòng lặp chính thực hiện hoán đổi cặp thành phố và cập nhật hành trình theo tiêu chí "lên dốc ngẫu nhiên".

4. Kết quả thực nghiệm.

Khi chạy thuật toán với 1000 lần lặp, kết quả thu được như sau:

```
Độ dài hành trình cuối cùng (Stochastic Hill Climb): 3.04
Tour length: 3.04
```



5. Nhận xét.

- Thuật toán Stochastic Hill Climbing đơn giản nhưng hiệu quả trong việc cải thiện nghiệm ngẫu nhiên ban đầu.
- Việc chọn ngẫu nhiên bước "lên dốc" giúp tránh hiện tượng hội tụ sớm so với Hill Climbing thông thường.
- Tuy nhiên, do vẫn chỉ di chuyển "lên dốc", thuật toán chưa thể thoát khỏi cực trị cục bộ nếu không có bước cải thiện nào tồn tại.
- Đây là cơ sở cho các phương pháp mạnh hơn như Simulated Annealing, trong đó thuật toán có thể chấp nhận tạm thời các bước "xuống dốc" để thoát khỏi cực trị.

IV. So sánh hiệu suất.

- Mục tiêu: Phần này nhằm đánh giá và so sánh hiệu suất giữa hai thuật toán tìm kiếm cục bộ: Steepest-Ascent Hill Climbing, Stochastic Hill Climbing

- Các tiêu chí được sử dụng để so sánh gồm:

+ Thời gian thực thi (Runtime) – thể hiện tốc độ thuật toán.

+ Khả năng mở rộng (Scalability) – đánh giá hiệu năng khi số lượng thành phố tăng.

+ Giá trị hàm mục tiêu tốt nhất (Best Objective Function Value) – thể hiện chất lượng nghiệm tìm được (độ ngắn của hành trình).

- Mô tả cài đặt: Thuật toán được cài đặt bằng ngôn ngữ Python, sử dụng thư viện numpy để sinh dữ liệu ngẫu nhiên, time để đo thời gian thực thi và matplotlib để trực quan hóa kết quả.

+ Hàm tsp_objective_function(): tính tổng độ dài quãng đường của một hành trình (route).

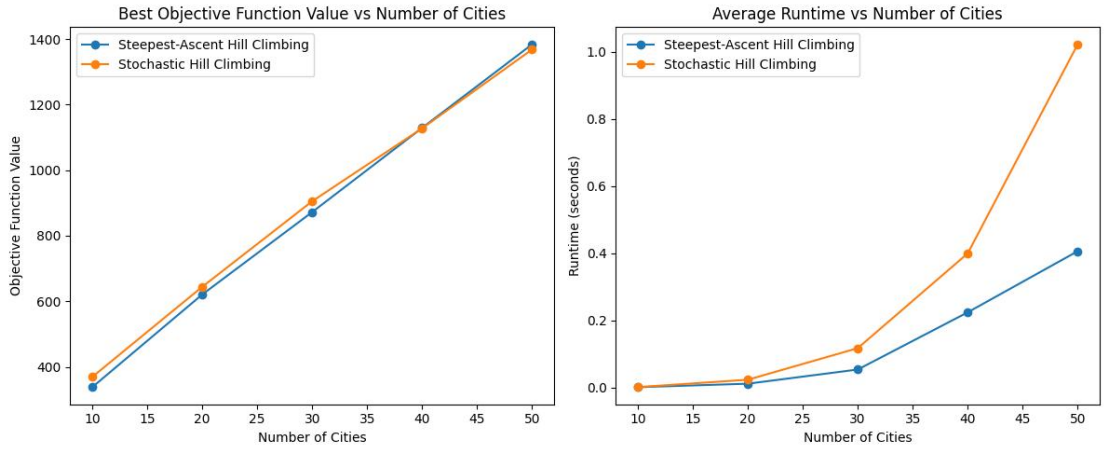
+ Hàm get_neighbors(): sinh ra các lời giải lân cận bằng cách hoán đổi vị trí của hai thành phố.

+ Hàm steepest_ascent_hill_climb(): tại mỗi bước, chọn *tốt nhất* trong tất cả hàng xóm và di chuyển đến đó nếu tốt hơn.

+ Hàm stochastic_hill_climb(): tại mỗi bước, chọn *ngẫu nhiên một hàng xóm tốt hơn* thay vì chọn tốt nhất.

+ Hàm compare_performance(): chạy hai thuật toán với các kích thước bài toán khác nhau (10, 20, 30, 40, 50 thành phố), lặp lại nhiều lần để lấy trung bình kết quả.

- Kết quả:



+ Biểu đồ bên trái : Best Objective Function Value vs Number of Cities

* Biểu đồ cho thấy giá trị hàm mục tiêu tốt nhất mà mỗi thuật toán đạt được khi số lượng thành phố tăng.

* Cả hai thuật toán đều có xu hướng tăng giá trị hàm mục tiêu khi số lượng thành phố tăng (vì bài toán trở nên phức tạp hơn).

* Đường của Steepest-Ascent Hill Climbing (màu xanh) nằm hơi thấp hơn một chút so với Stochastic Hill Climbing (màu cam), cho thấy: SAHC thường tìm được nghiệm tốt hơn hoặc tương đương so với SH, Điều này hợp lý vì SAHC luôn chọn bước leo tốt nhất ở mỗi lần lặp, còn SHC chọn ngẫu nhiên một bước leo nên đôi khi không đạt được giá trị tối ưu như SAHC.

+ Biểu đồ bên phải: Average Runtime vs Number of Cities

* Trục tung thể hiện thời gian chạy trung bình (Runtime), trục hoành là số lượng thành phố.

* Có thể thấy cả hai thuật toán đều tốn thời gian nhiều hơn khi số lượng thành phố tăng, tuy nhiên: Stochastic Hill Climbing (SHC) có thời gian

chạy lớn hơn đáng kể so với SAHC khi số lượng thành phố lớn, Nguyên nhân là SHC có thể thực hiện nhiều lần chọn ngẫu nhiên và kiểm tra nhiều hướng khác nhau trước khi đạt đến điểm cực trị, trong khi SAHC chọn hướng tối ưu ngay lập tức ở mỗi bước.

+ Kết luận :

* Steepest-Ascent Hill Climbing cho kết quả tốt hơn về cả giá trị hàm mục tiêu và thời gian chạy khi số lượng thành phố tăng.

* Tuy nhiên, Stochastic Hill Climbing có lợi thế trong việc tránh mắc kẹt tại các cực trị cục bộ, đặc biệt trong các không gian tìm kiếm phức tạp hơn.

* Tùy theo yêu cầu của bài toán (tốc độ hay tính ngẫu nhiên và khả năng khám phá), có thể lựa chọn thuật toán phù hợp.

V. Bonus: Genetic Algorithm.

- Genetic Algorithm (GA) là một phương pháp tối ưu hóa dựa trên cơ chế tiến hóa tự nhiên, mô phỏng quá trình chọn lọc, lai ghép và đột biến trong di truyền học. Đối với bài toán Traveling Salesman Problem (TSP), GA hoạt động bằng cách duy trì một quần thể (population) gồm nhiều lời giải khả thi (các tour đi qua tất cả các thành phố), sau đó dần dần cải thiện chúng qua nhiều thế hệ.
- Quá trình hoạt động của GA gồm các bước chính sau:

+ Khởi tạo quần thể ban đầu: Thuật toán bắt đầu với một tập hợp các tour ngẫu nhiên. Mỗi tour là một hoán vị của danh sách thành phố, biểu diễn một lời giải tiềm năng.

+ Đánh giá (Evaluation): Mỗi cá thể trong quần thể được đánh giá bằng hàm mục tiêu (fitness function) — trong TSP, thường là tổng chiều dài của hành trình. Các tour ngắn hơn tương ứng với mức độ thích nghi cao hơn.

+ Chọn lọc (Selection): Những cá thể có độ thích nghi cao được chọn để sinh sản. Một số kỹ thuật chọn lọc phổ biến gồm:

+ Roulette Wheel Selection: xác suất chọn tỉ lệ thuận với độ thích nghi.

+ Tournament Selection: chọn ngẫu nhiên vài cá thể, rồi chọn cá thể tốt nhất trong nhóm.

- Lai ghép (Crossover):

  + Hai "cha mẹ" được kết hợp để tạo ra "con" mới, với hy vọng thừa hưởng đặc điểm tốt của cả hai. Với TSP, cần các toán tử lai đặc biệt như:

    - Order Crossover (OX)
    - Partially Mapped Crossover (PMX)

  + Các phương pháp này đảm bảo con sinh ra vẫn là một hoán vị hợp lệ (không trùng hoặc thiếu thành phố).

- Đột biến (Mutation): Một số cá thể sẽ bị thay đổi ngẫu nhiên, ví dụ hoán đổi hai thành phố hoặc đảo ngược một đoạn đường trong tour. Bước này giúp duy trì đa dạng quần thể, tránh hội tụ sớm vào nghiệm cục bộ.
- Cập nhật thế hệ mới: Sau khi tạo ra các cá thể con, thuật toán chọn ra một số lượng nhất định (thường là những cá thể tốt nhất) để tạo thành quần thể mới, rồi tiếp tục lặp lại các bước trên cho đến khi đạt điều kiện dừng (ví dụ đạt số vòng lặp tối đa hoặc độ dài tour không còn cải thiện).
- Trong thực nghiệm, GA thường cho kết quả tốt hơn Hill Climbing về chất lượng nghiệm cuối cùng, đặc biệt với các bài toán có quy mô lớn hoặc không gian tìm kiếm phức tạp. GA có khả năng thoát khỏi cực trị địa phương nhờ cơ chế lai ghép và đột biến, giúp quá trình tìm kiếm mang tính toàn cục hơn. Tuy nhiên, GA tốn nhiều thời gian tính toán hơn do phải duy trì và đánh giá nhiều lời giải cùng lúc trong quần thể. Ngoài ra, hiệu quả của thuật toán phụ thuộc nhiều vào cách thiết kế toán tử lai, xác suất đột biến và kích thước quần thể.Tổng kết lại, Genetic Algorithm là một phương pháp mạnh mẽ và linh hoạt, phù hợp cho các bài toán tối ưu tổ hợp như TSP, đặc biệt khi cần tìm lời giải gần tối ưu trong không gian tìm kiếm lớn, nơi các phương pháp leo đồi (hill climbing) dễ bị mắc kẹt.