**Giải bài toán TSP bằng local search**

**Giới thiệu bài toán người du lịch (Traveling Salesman Problem – TSP)**

Mục tiêu của bài toán là tìm hành trình ngắn nhất đi qua tất cả nnn thành phố, mỗi thành phố đúng một lần và quay trở lại thành phố xuất phát. Cho trước khoảng cách giữa các cặp thành phố, trong đó di,j là khoảng cách từ thành phố i đến thành phố j.

Không gian trạng thái (State space) của bài toán được xác định bởi tất cả các hành trình có thể có. Mỗi trạng thái biểu diễn một hành trình (tour), trong đó các thành phố được đánh số thứ tự, và một hành trình có thể được biểu diễn bằng một vector hoán vị π\piπ cho biết thứ tự ghé thăm các thành phố. Cụ thể, π(1) là chỉ số của thành phố đầu tiên được ghé thăm, π(2) là thành phố thứ hai, và tiếp tục cho đến π(n) là thành phố cuối cùng trong hành trình.

Hàm mục tiêu của bài toán là tối thiểu hóa tổng độ dài của hành trình, được tính theo công thức:

A black background with white text

AI-generated content may be incorrect.

với điều kiện π là một vector hoán vị hợp lệ, tức là mỗi thành phố chỉ xuất hiện đúng một lần trong hành trình.

Các phép biến đổi cục bộ (local moves) được sử dụng trong các thuật toán tìm kiếm là việc hoán đổi vị trí của hai thành phố trong hành trình nhằm tạo ra một trạng thái lân cận mới để cải thiện lời giải hiện tại.

1. **Các hàm hỗ trợ**

Đầu tiên, ta tiến hành khai báo các thư viện cần thiết như NumPy, Pandas, Matplotlib, Math, và Random nhằm phục vụ cho việc sinh dữ liệu ngẫu nhiên, xử lý ma trận khoảng cách, và trực quan hóa kết quả. Hàm random.seed(1234) được sử dụng để đảm bảo các kết quả ngẫu nhiên có thể được tái lập khi chạy lại chương trình.

**1. Hàm random\_tour(n)**

Hàm này có nhiệm vụ tạo ra một hành trình (tour) ngẫu nhiên gồm n thành phố.  
Các thành phố được biểu diễn bằng các số nguyên từ 0 đến n-1, sau đó được xáo trộn ngẫu nhiên bằng random.shuffle() để tạo thành một thứ tự ghé thăm ngẫu nhiên.

**2. Hàm random\_tsp(n)**

Hàm này sinh ra một bài toán TSP ngẫu nhiên gồm n thành phố.  
Mỗi thành phố được gán một tọa độ ngẫu nhiên (x,y) trong khoảng [0,1).   
Sau đó, ta sử dụng hàm pdist và squareform từ thư viện SciPy để tính ma trận khoảng cách Euclidean giữa tất cả các cặp thành phố.  
Kết quả trả về là một dictionary gồm:

* pos: vị trí tọa độ của các thành phố.
* dist: ma trận khoảng cách giữa các thành phố.

**3. Hàm tour\_length(tsp, tour)**

Đây là hàm đánh giá hàm mục tiêu (objective function) của bài toán.  
Hàm này nhận vào:

* tsp: chứa thông tin về ma trận khoảng cách.
* tour: thứ tự ghé thăm các thành phố.

Hàm sẽ tính tổng khoảng cách giữa các thành phố liên tiếp trong tour, đồng thời cộng thêm khoảng cách từ thành phố cuối cùng quay về thành phố đầu tiên để tạo thành một vòng khép kín.

**4. Hàm show\_tsp(tsp, tour=None)**

Hàm này dùng để trực quan hóa bài toán TSP và hành trình tìm được.  
Nếu tham số tour được truyền vào, hàm sẽ:

* In ra tổng độ dài hành trình (tour length).
* Sắp xếp lại vị trí các thành phố theo thứ tự trong tour.
* Vẽ đường đi qua các thành phố theo thứ tự đó, đồng thời nối thành phố cuối cùng trở về điểm xuất phát.

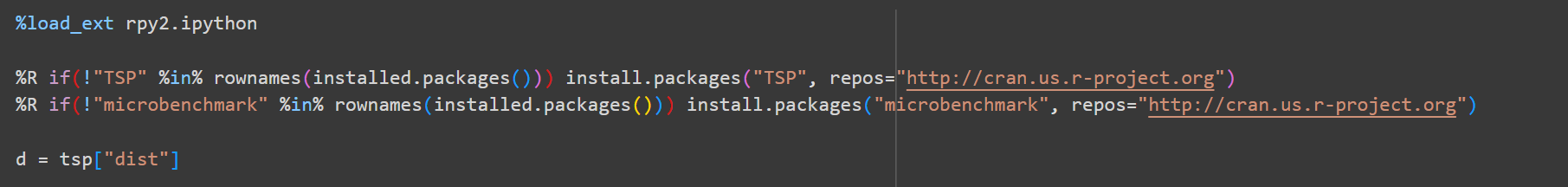
Kết quả hiển thị là biểu đồ scatter plot thể hiện vị trí các thành phố và đường đi tương ứng của hành trình.

1. **Sử dụng R để tìm lời giải cho bài toán TSP**

Để giải bài toán Người du lịch (Traveling Salesman Problem) một cách hiệu quả, ta có thể tận dụng gói TSP trong ngôn ngữ R, được thiết kế chuyên biệt cho các bài toán tối ưu hành trình.

Trước tiên, ta nạp phần mở rộng rpy2.ipython trong môi trường Python, cho phép tương tác trực tiếp giữa python và R.  
Sau đó, tiến hành kiểm tra và cài đặt các gói R cần thiết:

* TSP: dùng để định nghĩa và giải bài toán người du lịch.
* microbenchmark: dùng để đo thời gian thực thi của đoạn mã.



Sau khi chuẩn bị ma trận khoảng cách (distance matrix) giữa các thành phố, ta chuyển dữ liệu sang R để giải bài toán.

Giải bài toán TSP trong R

Sử dụng gói TSP, ta tạo một đối tượng tsp đại diện cho bài toán.  
Tiếp đó, ta dùng hàm solve\_TSP() để tìm hành trình ngắn nhất.  
Mặc định, phương pháp giải được sử dụng là 2-opt, đây là một thuật toán Steepest Ascent Hill Climbing, trong đó hai thành phố trong hành trình được hoán đổi để giảm độ dài tour.

Tham số rep = 100 được sử dụng để thực hiện 100 lần khởi tạo ngẫu nhiên (random restarts), nhằm tránh rơi vào nghiệm cục bộ và cải thiện kết quả tối ưu.

A screenshot of a computer program

AI-generated content may be incorrect.

Sau khi có kết quả hành trình tối ưu, ta trực quan hóa kết quả bằng hàm show\_tsp(tsp, tour) trong Python để vẽ đường đi qua các thành phố và hiển thị tổng chiều dài hành trình.

Tour length: 2.76A diagram of a constellation

AI-generated content may be incorrect.

**Đo thời gian thực thi thuật toán**

Để đánh giá hiệu năng của quá trình giải TSP, ta sử dụng gói microbenchmark trong R để đo thời gian thực thi của quá trình xây dựng đối tượng TSP và thực thi thuật toán với 100 lần khởi tạo ngẫu nhiên.

A screenshot of a computer

AI-generated content may be incorrect.

1. **Thuật toán Leo Núi Hướng Dốc Nhất (Steepest-Ascent Hill Climbing)**

Thuật toán Steepest-Ascent Hill Climbing là một phương pháp tìm kiếm cục bộ (local search) thường được sử dụng để giải bài toán TSP (Travelling Salesman Problem).  
Mục tiêu là liên tục cải thiện lời giải hiện tại bằng cách xem xét tất cả các hoán vị cục bộ có thể xảy ra, sau đó chọn lời giải tốt nhất trong số đó cho đến khi không còn cải thiện nào nữa.

Ý tưởng thuật toán

1. Bắt đầu từ một hành trình ngẫu nhiên (initial tour).
2. Sinh ra tất cả các lời giải lân cận (neighbors) bằng cách hoán đổi vị trí hai thành phố trong hành trình hiện tại.
3. Tính hàm mục tiêu (tour length) cho từng lời giải lân cận.
4. Chọn lời giải tốt nhất (có tổng quãng đường ngắn nhất) trong số các neighbors.
5. Nếu lời giải tốt nhất tốt hơn lời giải hiện tại, cập nhật và lặp lại bước 2.
6. Nếu không còn cải thiện nào, thuật toán dừng tại cực trị cục bộ (local optimum).

Cài đặt thuật toán:

A computer screen shot of a program

AI-generated content may be incorrect.

A computer screen shot of a program

AI-generated content may be incorrect.

Kết quả sau chạy thuật toán Steepest-Ascent Hill Climbing:

A graph with blue lines and numbers

AI-generated content may be incorrect.

Nhận xét:

Thuật toán Steepest-Ascent Hill Climbing cho thấy khả năng cải thiện hành trình rõ rệt khi chiều dài đường đi giảm đáng kể so với ban đầu, lộ trình thu được ít giao cắt và hợp lý hơn, ưu điểm của thuật toán là đơn giản, dễ triển khai, hiệu quả với dữ liệu nhỏ và luôn chọn bước đi tốt nhất, tuy nhiên nó dễ mắc kẹt tại cực trị cục bộ, tốn thời gian khi số thành phố lớn và phụ thuộc vào trạng thái khởi tạo ban đầu; nhìn chung đây là phương pháp tìm kiếm cục bộ hiệu quả cho bài toán TSP nhỏ

1. **Thuật toán Stochastic Hill Climbing**

Thuật toán này là một biến thể ngẫu nhiên của Hill Climbing, hoạt động theo các bước sau:

1. Bắt đầu với một tour ban đầu (initial\_tour).
2. Sinh tất cả các hoán đổi 2 thành phố (neighbor).
3. Lọc ra các neighbor có độ dài ngắn hơn tour hiện tại (tức là tốt hơn).
4. Chọn ngẫu nhiên một trong số các neighbor tốt hơn này để di chuyển đến, thay vì chọn cái tốt nhất như trong Steepest-Ascent.
5. Lặp lại tối đa max\_iter lần hoặc cho đến khi không có neighbor nào tốt hơn.

Cài đặt thuật toán:

A screen shot of a computer code

AI-generated content may be incorrect.

Kết quả sau khi chạy thuật toán:

A screenshot of a graph

AI-generated content may be incorrect.

Nhận xét:

Thuật toán Stochastic Hill Climbing đã tìm được lộ trình tối ưu cục bộ với độ dài tour 2.88, cho thấy sự cải thiện so với ban đầu và đường đi hợp lý giữa các thành phố. Khác với Steepest-Ascent luôn chọn hướng cải thiện tốt nhất, Stochastic Hill Climbing chọn ngẫu nhiên trong các hướng cải thiện, giúp tránh mắc kẹt tại cực trị cục bộ nhưng kết quả có thể thay đổi giữa các lần chạy.

1. **First-choice Hill Climbing**

Thuật toán First-Choice Hill Climbing là một biến thể của Hill Climbing trong nhóm local search (tìm kiếm cục bộ). Mục tiêu của thuật toán là cải thiện lời giải hiện tại bằng cách di chuyển sang các trạng thái lân cận có giá trị hàm mục tiêu tốt hơn, tuy nhiên thay vì xét toàn bộ các lân cận, thuật toán này chọn ngẫu nhiên một lân cận tại mỗi bước và chấp nhận ngay nếu nó tốt hơn. Cách tiếp cận này giúp giảm đáng kể thời gian tính toán nhưng vẫn duy trì khả năng cải thiện lời giải.

Nguyên lý hoạt động:

1. Bắt đầu với một hành trình ban đầu (tour ngẫu nhiên).
2. Tạo một cặp thành phố ngẫu nhiên và hoán đổi vị trí của chúng để tạo ra một láng giềng.
3. Nếu hành trình mới ngắn hơn hành trình hiện tại (tốt hơn), thì cập nhật lời giải.
4. Lặp lại quá trình cho đến khi đạt số lần lặp tối đa hoặc không còn cải thiện.

**Ưu điểm:**

Tốc độ nhanh hơn vì không cần duyệt toàn bộ không gian láng giềng. Có khả năng thoát khỏi cực trị cục bộ nếu có yếu tố ngẫu nhiên tốt.

**Nhược điểm:**

Dễ bỏ lỡ hướng cải thiện tốt nhất. Kết quả có thể không ổn định, phụ thuộc vào chuỗi ngẫu nhiên sinh ra.

Cài đặt thuật toán:

A screen shot of a computer program

AI-generated content may be incorrect.

Kết quả chạy thuật toán:

A graph with blue lines and dots

AI-generated content may be incorrect.

Nhận xét kết quả chạy thuật toán:

Thuật toán First-Choice Hill Climbing cho kết quả độ dài hành trình tốt hơn hai thuật toán trước (≈ 2.79 so với ≈ 2.88), cho thấy hiệu quả cải thiện đáng kể dù chỉ kiểm tra ngẫu nhiên một lân cận tại mỗi bước. Do chọn lân cận ngẫu nhiên và cập nhật ngay khi tốt hơn, thuật toán chạy nhanh hơn Steepest-Ascent và Stochastic Hill Climbing, đồng thời có khả năng thoát khỏi cực trị cục bộ tốt hơn Steepest-Ascent. Tuy nhiên, do tính ngẫu nhiên cao nên kết quả có thể khác nhau giữa các lần chạy, không ổn định bằng Steepest-Ascent.

1. **Thuật toán Simulated Annealing**

Simulated Annealing (SA) là một biến thể nâng cao của Hill Climbing, mô phỏng quá trình “tôi luyện kim loại” trong vật lý. Thuật toán bắt đầu với nhiệt độ cao và dần làm nguội theo thời gian. Ở giai đoạn đầu, thuật toán cho phép chấp nhận những lời giải kém hơn (tức có giá trị mục tiêu lớn hơn) với một xác suất nhất định, giúp nó có khả năng thoát khỏi các cực trị cục bộ mà Hill Climbing thường mắc phải. Khi nhiệt độ giảm dần, thuật toán trở nên “bảo thủ” hơn, chỉ chấp nhận những lời giải tốt hơn dần hội tụ về nghiệm tối ưu cục bộ hoặc gần tối ưu toàn cục.

Nguyên lí hoạt động:

* Khởi tạo tour ban đầu và nhiệt độ Tstart ​.
* Ở mỗi vòng lặp, sinh một lân cận ngẫu nhiên bằng cách hoán đổi hai thành phố.
* Nếu lân cận tốt hơn, chấp nhận ngay.
* Nếu kém hơn, vẫn có thể chấp nhận với xác suất e^−Δ/T.
* Giảm dần nhiệt độ theo hệ số làm nguội alpha cho đến khi nhỏ hơn Tend ​ hoặc đạt số vòng lặp tối đa.
* Ghi nhận tour có độ dài ngắn nhất trong toàn bộ quá trình.

**Ưu điểm:**  
 Thuật toán Simulated Annealing vượt trội hơn so với các phương pháp Hill Climbing thuần túy, vì có khả năng vượt qua các điểm cực trị cục bộ, giúp tìm ra lời giải tốt hơn gần với tối ưu toàn cục. Ngoài ra, nó đơn giản, dễ triển khai và không cần khảo sát toàn bộ lân cận, nên tốc độ vẫn khả thi với các bài toán kích thước vừa.

**Nhược điểm:**  
 Tốc độ hội tụ phụ thuộc mạnh vào các tham số Tstart ​, alpha và số vòng lặp. Nếu giảm nhiệt quá nhanh, dễ dừng sớm ở nghiệm chưa tốt; nếu giảm quá chậm, thời gian chạy tăng đáng kể. Ngoài ra, kết quả vẫn mang tính ngẫu nhiên và có thể thay đổi giữa các lần chạy.

Cài đặt thuật toán:

A screen shot of a computer program

AI-generated content may be incorrect.

Kết quả sau khi chạy thuật toán:

A screen shot of a graph

AI-generated content may be incorrect.

Nhận xét:

Thuật toán Simulated Annealing đã cải thiện đáng kể hành trình so với ban đầu, giảm tổng độ dài tour từ 4.79 xuống còn 3.05, cho thấy khả năng tối ưu tốt. Tuy nhiên, kết quả cuối cùng vẫn dài hơn một chút so với các thuật toán Hill Climbing trước, do đặc tính ngẫu nhiên và quá trình làm nguội ảnh hưởng đến khả năng hội tụ. Dù vậy, SA có ưu điểm nổi bật là không bị mắc kẹt tại cực trị cục bộ, giúp tìm được lời giải ổn định và có tính tổng quát cao hơn các thuật toán leo đồi truyền thống.

1. **So sánh hiệu năng của các thuật toán**

**1. Mục tiêu**

So sánh bốn thuật toán Heuristic:

* Steepest-Ascent Hill Climbing
* Stochastic Hill Climbing
* First-Choice Hill Climbing
* Simulated Annealing

**Dựa trên ba tiêu chí:**

1. Runtime (ms) — thời gian chạy trung bình.
2. Scalability — khả năng mở rộng khi tăng số lượng thành phố.
3. Best Length — giá trị hàm mục tiêu nhỏ nhất (tổng quãng đường ngắn nhất).

Cài đặt:

A screenshot of a computer

AI-generated content may be incorrect.

Kết quả sau khi cài đặt:

A screenshot of a computer

AI-generated content may be incorrect.

Nhận xét:

Stochastic Hill Climbing cho kết quả tốt nhất ở bài toán nhỏ, Simulated Annealing ổn định và tránh kẹt cực trị khi bài toán lớn, First-Choice nhanh nhất nhưng ít chính xác, còn Steepest-Ascent ổn định, tốn thời gian khi số thành phố tăng.

1. **Genetic algorithm**

**Mục tiêu:**  
 Tìm đường đi ngắn nhất qua tất cả các thành phố bằng cách mô phỏng quá trình tiến hóa tự nhiên (chọn lọc, lai ghép, đột biến).

**Nguyên lí hoạt động:**

1. Khởi tạo quần thể: Sinh ngẫu nhiên nhiều tour (nghiệm).
2. Đánh giá độ thích nghi (fitness): Dựa trên nghịch đảo độ dài đường đi (1 / tour\_length).
3. Chọn lọc (Selection): Dùng Roulette Wheel để chọn các cá thể tốt hơn làm “bố mẹ”.
4. Lai ghép (Crossover): Kết hợp hai cá thể bằng Ordered Crossover (OX) để tạo con.
5. Đột biến (Mutation): Đảo vị trí hai thành phố trong tour với xác suất nhỏ.
6. Cập nhật thế hệ: Thay quần thể cũ bằng quần thể con và lưu nghiệm tốt nhất.

Cài đặt thuật toán:

A screen shot of a computer

AI-generated content may be incorrect.A screen shot of a computer

AI-generated content may be incorrect.

Kết quả sau khi cài đặt thuật toán:

A screen shot of a graph

AI-generated content may be incorrect.

Nhận xét:

Thuật toán di truyền (GA) cho kết quả tour tối ưu có độ dài 2.98, thể hiện khả năng tìm nghiệm tốt nhờ cơ chế lai ghép (crossover) và đột biến (mutation) giúp khám phá không gian nghiệm hiệu quả, tránh được việc mắc kẹt ở cực trị cục bộ; tuy nhiên quá trình tiến hóa qua nhiều thế hệ nên thời gian chạy chậm hơn các thuật toán leo đồi. So với Steepest-Ascent, Stochastic, First-Choice và Simulated Annealing, thuật toán Genetic Algorithm cho kết quả tốt và ổn định hơn, đặc biệt ở các bài toán có số thành phố lớn; tuy nhiên nó tốn tài nguyên và thời gian hơn do cần xử lý quần thể lớn và nhiều vòng lặp tiến hóa.