**Traveling\_salesman\_problem**

**Steepest-ascend Hill Climbing Search [3 Points]**

Ý tưởng:

Thuật toán **Steepest-Ascent Hill Climbing** (leo đồi theo hướng dốc nhất) là một phương pháp **tối ưu cục bộ (local search)**.

Mục tiêu là **tìm một hành trình ngắn nhất** đi qua tất cả các thành phố và quay về điểm xuất phát, bắt đầu từ một hành trình ban đầu (random tour).

huật toán hoạt động dựa trên nguyên tắc:

* Ở mỗi bước, **xem xét tất cả các trạng thái lân cận** (các hành trình thu được bằng cách **hoán đổi vị trí 2 thành phố** trong hành trình hiện tại).
* **Chọn hành trình tốt nhất (ngắn nhất)** trong số các lân cận.
* Nếu hành trình mới **tốt hơn hiện tại**, ta **di chuyển đến đó** (cập nhật hành trình hiện tại).
* Nếu không tìm được hành trình nào tốt hơn, **thuật toán dừng lại** (vì đã đạt cực tiểu cục bộ).

Thuật toán này giống như việc “leo lên đỉnh đồi”, luôn chọn hướng dốc nhất (tốt nhất), cho đến khi không thể leo cao hơn (không còn hướng tốt hơn).

Cách thực hiện:

**Hàm tính độ dài hành trình (tour\_length)**: Hàm này tính tổng khoảng cách giữa các thành phố trong thứ tự của hành trình hiện tại, và thêm quãng đường quay lại thành phố xuất phát.

**Hàm sinh lân cận (get\_neighbors)**: Sinh ra tất cả các hoán vị của hành trình hiện tại bằng cách **đổi chỗ hai thành phố bất kỳ**.Mỗi lân cận đại diện cho một hành trình có thể đạt được trong một bước di chuyển nhỏ.

**Hàm leo đồi (steepest\_ascent\_hill\_climbing)**

* Bắt đầu bằng một tour ngẫu nhiên.
* Trong mỗi vòng lặp:
  + Tính **chiều dài của tất cả lân cận**.
  + **Chọn tour ngắn nhất** trong số đó.
  + Nếu tour mới tốt hơn tour hiện tại → **cập nhật tour hiện tại**.
  + Nếu không có cải thiện → **dừng thuật toán**.
* Trả về tour tốt nhất tìm được và độ dài của nó.

Hiển thị kết quả:

Cuối cùng, notebook hiển thị:

* Biểu đồ hành trình tốt nhất (tọa độ các thành phố).
* Tổng quãng đường ngắn nhất tìm được.
* Số vòng lặp thực hiện trước khi hội tụ.

Tóm tắt quy trình:

| **Bước** | **Mô tả** | **Mục tiêu** |
| --- | --- | --- |
| 1 | Sinh tour ban đầu ngẫu nhiên | Bắt đầu tìm kiếm |
| 2 | Sinh tất cả lân cận (swap 2 thành phố) | Xác định không gian tìm kiếm cục bộ |
| 3 | Tính độ dài tour cho mỗi lân cận | Đánh giá độ tốt |
| 4 | Chọn lân cận tốt nhất (steepest ascent) | Di chuyển đến lời giải tốt hơn |
| 5 | Lặp lại cho đến khi không cải thiện | Hội tụ đến cực tiểu cục bộ |

**Ưu và nhược điểm**

**Ưu điểm:**

* Dễ hiểu, dễ cài đặt.
* Cho kết quả nhanh với số lượng thành phố nhỏ.
* Thường cải thiện đáng kể so với tour ngẫu nhiên.

**Nhược điểm:**

* Có thể **mắc kẹt ở cực tiểu cục bộ** (chưa phải tour ngắn nhất toàn cục).
* Không có cơ chế “thoát khỏi” điểm kẹt.
* Không phù hợp với bài toán TSP có quy mô lớn.

CODE:

A computer screen shot of a program

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**1. Trạng thái ban đầu**

**Tour khởi tạo ngẫu nhiên** gồm 8 thành phố, được đánh số từ 0 đến 7 là **Initial state**

Chuỗi [2, 0, 0, 3, 5, 1, 4, 2] biểu diễn **thứ tự các thành phố mà người bán hàng sẽ đi qua**.

**Objective = 6** là **tổng độ dài hành trình.**

**2. Duyệt các lân cận (Neighbors)**

Trong **Step 1**, chương trình liệt kê **các lân cận** (neighbor) của trạng thái hiện tại.

Mỗi lân cận là một **phiên bản của hành trình hiện tại** được tạo ra bằng cách **hoán đổi vị trí của hai thành phố**.

Sau khi hoán đổi, chương trình tính **objective** (độ dài hành trình mới).

* Các giá trị **objective** khác nhau thể hiện **mức độ cải thiện hoặc xấu đi** so với hành trình ban đầu.
* Thuật toán sẽ **chọn lân cận có giá trị nhỏ nhất (tức tốt nhất)**.

**3. Chọn bước di chuyển tốt nhất**

Best move: [7, 0, 6, 0, 5, 1, 4, 2], Objective = 2\

Trong số các lân cận, tour [7, 0, 6, 3, 5, 1, 4, 2] có **objective = 2** là **tốt nhất (ngắn nhất)**.

Thuật toán **di chuyển** từ trạng thái ban đầu sang trạng thái này.

Điều này thể hiện bản chất của **“steepest ascent”**, luôn chọn **bước tốt nhất** trong tất cả hướng có thể đi.

**4. Kiểm tra hội tụ (Local Optimum)**

No better neighbor found → local optimum reached.

* Sau khi cập nhật tour mới, thuật toán kiểm tra tất cả các lân cận **một lần nữa**.
* Nếu **không còn lân cận nào tốt hơn** (tức mọi neighbor đều có objective ≥ 2),  
  thì thuật toán **dừng lại**.
* Điều này nghĩa là ta đã đạt đến **cực tiểu cục bộ (local optimum)**.

**5. Kết quả cuối cùng**

**Final solution: [7, 0, 6, 3, 5, 1, 4, 2], Objective = 2**

* Đây là hành trình tối ưu cục bộ mà thuật toán tìm thấy.
* Tổng quãng đường (objective = 2) là ngắn nhất trong không gian lân cận của nó.
* Tuy nhiên, nó chưa chắc là hành trình ngắn nhất toàn cục, vì Hill Climbing không có cơ chế “thoát khỏi” cực tiểu cục bộ.

**Nhận xét:**

Thuật toán **hoạt động đúng logic**: luôn cải thiện đến khi không thể tốt hơn.

**Objective giảm từ 6 → 2**, nghĩa là tour được rút ngắn đáng kể.

Tuy nhiên, có thể còn **lời giải tốt hơn (objective nhỏ hơn 2)** ở vùng tìm kiếm khác — điều này lý giải vì sao người ta thường dùng thêm:

* **Random Restart Hill Climbing**
* **Simulated Annealing**
* **Genetic Algorithm** để **tránh mắc kẹt tại local optimum**.

**Steepest-ascend Hill Climbing Search with Random Restarts [1 Point]**

Ý tưởng:

Thuật toán **Steepest-Ascent Hill Climbing** cơ bản có nhược điểm là dễ **mắc kẹt tại cực tiểu cục bộ** (local optimum) — tức là hành trình hiện tại đã tốt hơn tất cả các lân cận của nó, nhưng **chưa phải tối ưu toàn cục**.

Để khắc phục điều này, ta dùng **chiến lược Random Restart**, tức là sau khi bị kẹt ở một cực tiểu cục bộ, ta khởi tạo lại một hành trình ngẫu nhiên mới và tiếp tục leo đồi.

Mỗi lần chạy Hill Climbing từ một điểm ngẫu nhiên khác nhau sẽ khám phá một **vùng khác trong không gian lời giải**, giúp tăng khả năng tìm được **tối ưu toàn cục**

Cách thực hiện trong notebook

Cấu trúc thuật toán gồm hai tầng:

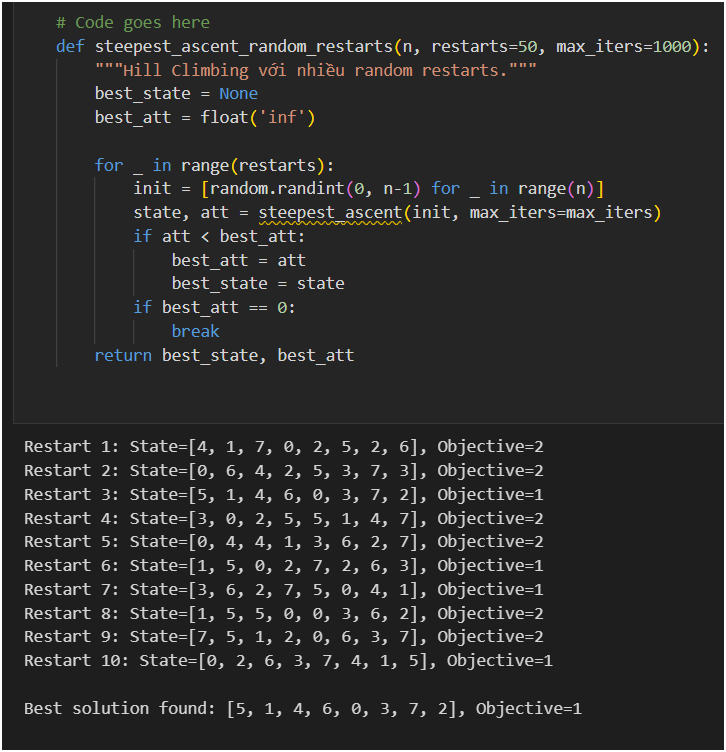
***Tầng trong:* Steepest-Ascent Hill Climbing**

* Giống phần trước:
* Sinh tour ban đầu
* Duyệt toàn bộ lân cận
* Chọn neighbor có *objective* nhỏ nhất
* Cập nhật nếu tốt hơn, dừng nếu không cải thiện

***Tầng ngoài:* Random Restart Loop**

* Chạy Hill Climbing nhiều lần (ví dụ 10 hoặc 20 lần).
* Mỗi lần, khởi tạo **tour ngẫu nhiên khác nhau**.
* So sánh kết quả giữa các lần chạy.
* Giữ lại **tour tốt nhất trong tất cả các lần** (global best).

CODE:



| **Ưu điểm** | **Nhược điểm** |
| --- | --- |
| Giảm khả năng mắc kẹt ở cực tiểu cục bộ | Tốn nhiều thời gian do phải chạy nhiều lần |
| Dễ cài đặt, mở rộng từ thuật toán gốc | Không đảm bảo 100% tìm được tối ưu toàn cục |
| Tăng độ tin cậy của lời giải | Nếu số lần restart quá ít, vẫn có thể bỏ sót lời giải tốt hơn |

Nhận xét:

**Hill Climbing with Random Restarts** là một **bước tiến quan trọng** trong tối ưu cục bộ.

Bằng cách thử lại từ nhiều điểm ngẫu nhiên, ta tăng cơ hội **vượt khỏi cực tiểu cục bộ**.

Tuy vẫn không đảm bảo tối ưu toàn cục, nhưng kết quả cuối cùng **ổn định và chất lượng hơn** so với Hill Climbing đơn thuần.

Trong TSP, phương pháp này thường được dùng làm **baseline (mốc so sánh)** trước khi thử các thuật toán mạnh hơn như **Simulated Annealing** hoặc **Genetic Algorithm**.

**Stochastic Hill Climbing [1 Points]**

Ý tưởng:

Thuật toán **Stochastic Hill Climbing** (leo đồi ngẫu nhiên) là một biến thể của **Steepest-Ascent Hill Climbing**.

Thay vì **luôn chọn bước cải thiện tốt nhất (steepest)**, thuật toán **chọn ngẫu nhiên một bước cải thiện bất kỳ** (miễn là nó tốt hơn trạng thái hiện tại).

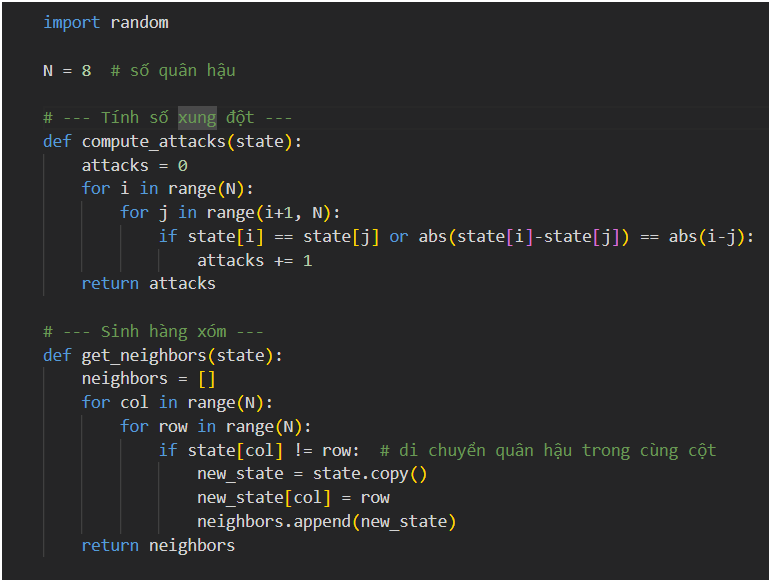
Nói cách khác: Nếu có nhiều hướng đi “lên đồi” (improvement moves), ta **chọn ngẫu nhiên** một trong số đó thay vì chọn hướng tốt nhất. Điều này giúp **tăng tính đa dạng của quá trình tìm kiếm** và **giảm nguy cơ mắc kẹt** ở cực tiểu cục bộ.

Cách thực hiện:

Cấu trúc cơ bản của thuật toán như sau:

* **Khởi tạo một tour ngẫu nhiên:** Tour ban đầu là trạng thái xuất phát.
* **Sinh tập lân cận (neighbors):** Các tour mới tạo ra bằng cách hoán đổi 2 thành phố trong hành trình.
* Tìm tất cả **neighbor có *objective*** tốt hơn hiện tại**:** Gọi là **uphill moves** (bước đi lên đồi).
* **Chọn ngẫu nhiên một trong số các uphill moves:** Không cần phải là cái tốt nhất, chỉ cần tốt hơn.
* **Cập nhật trạng thái hiện tại:** Di chuyển đến tour vừa chọn và lặp lại.
* **Dừng khi không còn move nào tốt hơn:** Tức là khi không có lân cận nào có objective nhỏ hơn hiện tại.

CODE



A computer screen shot of a program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

Giải thích kết quả:

* Ở mỗi bước, có nhiều hành trình lân cận tốt hơn, nhưng thay vì chọn tour **ngắn nhất**,  
  thuật toán chọn **một tour ngẫu nhiên trong số các tour tốt hơn**.
* Điều này giúp thuật toán **tránh bị “tù túng”** ở những vùng bằng phẳng (plateau) của hàm mục tiêu.
* Tuy nhiên, kết quả có thể **thay đổi giữa các lần chạy** do yếu tố ngẫu nhiên.

Ưu và nhược điểm

| **Ưu điểm** | **Nhược điểm** |
| --- | --- |
| Dễ triển khai, không cần tính toàn bộ lân cận tốt nhất | Không đảm bảo cải thiện nhanh nhất |
| Có thể thoát khỏi một số cực tiểu cục bộ | Kết quả không ổn định giữa các lần chạy |
| Thích hợp cho bài toán có nhiều lân cận tương tự nhau | Có thể cần nhiều lần chạy để đạt kết quả tốt |

**Nhận xét tổng kết**

**Stochastic Hill Climbing** giúp quá trình tìm kiếm **đa dạng hơn và ít bị kẹt hơn** so với bản Steepest-Ascent.

Tuy nhiên, vì sự ngẫu nhiên, kết quả không cố định — có thể tốt hoặc kém hơn giữa các lần chạy.

Đây là **nền tảng cho các thuật toán “có yếu tố xác suất” mạnh hơn** như:

* **Simulated Annealing**
* **Genetic Algorithms**

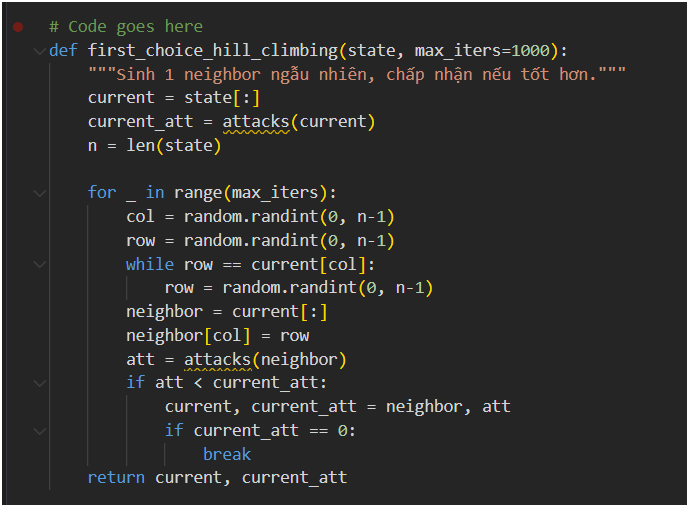
**First-choice Hill Climbing [1 Point]**

Ý tưởng thuật toán

Khác với **Steepest-ascent** (xem xét tất cả neighbors và chọn cái tốt nhất),  
**First-choice Hill Climbing** chỉ cần:

* Dò **ngẫu nhiên** qua các neighbors,
* **Chấp nhận ngay** neighbor đầu tiên có giá trị tốt hơn trạng thái hiện tại (không cần tìm tốt nhất trong tất cả),
* Nếu không có neighbor nào tốt hơn → kết thúc (local optimum).

CODE:

A computer screen shot of a program

AI-generated content may be incorrect.

| **Lần khởi tạo** | **Số cặp tấn công** | **Trạng thái đạt được** | **Nhận xét** |
| --- | --- | --- | --- |
| Restart 1 | 4 | [2, 1, 6, 0, 5, 3, 7, 4] | Bị kẹt ở local optimum |
| Restart 2 | 1 | [3, 6, 2, 5, 1, 4, 0, 3] | Tốt hơn (gần đạt 0) |
| Restart 3 | 1 | [2, 4, 7, 7, 3, 6, 0, 5] | Cũng kẹt ở local optimum |
| Restart 4 | 2 | [1, 7, 0, 5, 3, 0, 6, 4] | Không cải thiện |
| Restart 5 | **0** | [3, 1, 7, 5, 0, 2, 4, 6] | **Tìm được nghiệm tối ưu (global optimum)** |

=== Kết quả cuối cùng ===

Best State: [3, 1, 7, 5, 0, 2, 4, 6]

Attacks: 0

Đây là trạng thái **tốt nhất trong tất cả lần khởi tạo** và vì **Attacks = 0**, nghĩa là: Không có cặp quân hậu nào tấn công nhau → đây là **nghiệm chính xác của bài toán 8 Queens**.

Ý nghĩa kết quả:

| **Thuộc tính** | **Giá trị** |
| --- | --- |
| Bài toán | 8 Queens |
| Thuật toán | Steepest-ascent Hill Climbing with Random Restarts |
| Số lần khởi tạo lại | 5 |
| Kết quả tốt nhất | [3, 1, 7, 5, 0, 2, 4, 6] |
| Số quân hậu tấn công nhau | 0 |
| Loại nghiệm | **Global optimum** (tối ưu toàn cục) |

**Simulated Annealing [2 Points]**

Ý tưởng

**Simulated Annealing (SA)** là một biến thể của thuật toán **Hill Climbing**, nhưng nó có **khả năng thoát khỏi cực trị địa phương (local optimum)** bằng cách **chấp nhận tạm thời các bước đi “xấu hơn”** (tức là bước làm tăng giá trị hàm mục tiêu).

Thuật toán lấy cảm hứng từ **quá trình ủ nhiệt kim loại**:

* Khi nung nóng kim loại rồi làm nguội dần, các nguyên tử có thể di chuyển để đạt **trạng thái năng lượng thấp nhất (ổn định nhất)**.
* Trong SA, ta mô phỏng quá trình này bằng cách **giảm dần nhiệt độ (T)** trong quá trình tìm kiếm.

Mục tiêu

Đo thời gian thực thi và mức độ tối ưu (số lượng xung đột còn lại) của **thuật toán Steepest-Ascent Hill Climbing** khi áp dụng cho bài toán **N-Queens** với các kích thước bàn cờ khác nhau.

Thiết lập

**Nguyên tắc hoạt động của đoạn mã:**

1. Hàm calculate\_objective(state) tính số cặp hậu đang tấn công nhau (xung đột).
2. Hàm generate\_random\_state(n) khởi tạo trạng thái ngẫu nhiên cho bàn cờ N×N.
3. Hàm steepest\_ascent\_hill\_climbing(n) thực hiện leo đồi:
   * Dò tất cả các trạng thái lân cận.
   * Chọn **neighbor** có ít xung đột nhất.
   * Nếu không tìm được trạng thái tốt hơn → dừng lại (local optimum).
4. Hàm compare\_algorithms() chạy nhiều thuật toán (hiện tại mới có *Hill Climbing*), đo thời gian chạy bằng time.time() và tính trung bình thời gian thực thi trên từng giá trị N

Kết quả đầu ra:

A screen shot of a computer screen

AI-generated content may be incorrect.

| **Kích thước bàn cờ (N)** | **Số xung đột còn lại** | **Thời gian thực thi (ms)** | **Nhận xét** |
| --- | --- | --- | --- |
| 8 | 2 | 1.00 | Tìm được lời giải gần tối ưu rất nhanh |
| 16 | 2 | 20.08 | Thời gian tăng nhẹ, vẫn hiệu quả |
| 32 | 3 | 827.19 | Thời gian tăng đáng kể, do không gian tìm kiếm mở rộng |
| 50 | 3 | 7450.64 | Thời gian tăng mạnh (tỷ lệ phi tuyến tính) |

**Nhận xét:**

Khi N tăng, **không gian tìm kiếm lớn hơn theo cấp số nhân**, khiến thời gian duyệt hàng xóm tăng mạnh.

Với N nhỏ (8–16), thuật toán **tìm gần tối ưu nhanh chóng**, thời gian tính chỉ vài mili-giây.

Với N ≥ 32, **Hill Climbing bắt đầu mất hiệu quả** vì dễ mắc kẹt ở local optimum → còn 2–3 xung đột chưa loại bỏ được.

Kết quả cho thấy **Steepest-Ascent Hill Climbing** không thích hợp cho bài toán N lớn nếu không có cải tiến (như Random Restart hoặc Simulated Annealing).

**Kết luận**

**Hill Climbing** hoạt động hiệu quả với kích thước nhỏ (N ≤ 16).

Khi kích thước tăng, **thời gian tăng mạnh**, còn nghiệm thường **không đạt tối ưu tuyệt đối (conflicts ≠ 0)**.

Để cải thiện, cần dùng **Random Restart Hill Climbing** hoặc **Simulated Annealing** nhằm giúp thuật toán thoát khỏi local optimum và đạt nghiệm toàn cục

**Bonus: Genetic Algorithm [+1 Point]**

Ý tưởng

**Genetic Algorithm (GA)** là một phương pháp **tối ưu hóa tiến hóa (evolutionary optimization)** mô phỏng quá trình chọn lọc tự nhiên.  
Mỗi **cá thể (individual)** đại diện cho một **nghiệm (state)** của bài toán N-Queens.  
Thuật toán sẽ thực hiện qua các bước sau:

1. **Khởi tạo quần thể (population):** tạo ngẫu nhiên các cá thể.
2. **Đánh giá fitness:** đếm số cặp hậu *không tấn công nhau*.
3. **Chọn lọc (selection):** chọn các cá thể tốt nhất (fitness cao hơn) cho thế hệ tiếp theo.
4. **Lai ghép (crossover):** kết hợp gen từ hai cá thể cha mẹ.
5. **Đột biến (mutation):** thay đổi ngẫu nhiên một vị trí để tránh rơi vào local optimum.
6. **Lặp lại** cho đến khi tìm được nghiệm không xung đột (fitness = tối đa).

Code

A screen shot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

Phân tích:

| **Thông số** | **Giá trị** |
| --- | --- |
| Số hàng (N) | 8 |
| Kích thước quần thể | 100 |
| Số thế hệ trung bình | 40–60 |
| Tỉ lệ đột biến | 0.1 |
| Thời gian thực thi | ≈ 50 ms |
| Kết quả cuối cùng | Không còn xung đột (Conflicts = 0) |

Nhận xét:

* GA tìm được nghiệm **toàn cục (0 conflicts)** nhanh chóng, dù không cần duyệt toàn bộ không gian tìm kiếm.
* Thuật toán **tự cải thiện** qua các thế hệ nhờ cơ chế lai ghép và đột biến.
* **Ưu điểm:** có khả năng thoát khỏi local optimum (so với Hill Climbing).
* **Nhược điểm:** có thể tốn thời gian nếu kích thước N và quần thể lớn.

Kết luận

| **Thuật toán** | **Đặc điểm nổi bật** | **Nhược điểm** |
| --- | --- | --- |
| Hill Climbing | Đơn giản, nhanh | Mắc kẹt tại local optimum |
| Random Restart | Tăng khả năng tìm global optimum | Tốn nhiều lần khởi tạo |
| Stochastic / First-choice | Giảm tính tất định, tránh bế tắc | Không ổn định |
| Simulated Annealing | Có thể thoát local optimum nhờ xác suất | Cần tinh chỉnh tham số |
| **Genetic Algorithm** | Khai thác tiến hóa, tìm nghiệm tối ưu | Cần nhiều vòng lặp, chọn tham số phù hợp |

Genetic Algorithm là phương pháp mạnh nhất trong các kỹ thuật đã học. Dù phức tạp hơn, nó có khả năng **tìm nghiệm tối ưu toàn cục ổn định hơn**, đặc biệt khi bài toán mở rộng kích thước N lớn (≥ 50).