



**Báo cáo bài thực hành Lab3**

**Môn học: Trí tuệ nhân tạo nâng cao**

Giảng viên: Đỗ Như Tài

Nhóm 17 – Thành viên:

Nguyễn Quốc Huy – 3122410149

Trương Gia Huy – 3122410154

Phạm Duy Khánh – 3122410181

**Bảng phân công công việc**

|  |  |  |
| --- | --- | --- |
| Thành viên | Công việc | Tiến độ |
| Nguyễn Quốc Huy | Traveling | Hoàn thành |
| Trương Gia Huy | N Queens | Hoàn thành |
| Phạm Duy Khánh | Viết báo cáo | Hoàn thành |

**Mục lục**

[**A.** **Assignment: n-Queens Problem with Genetic Algorithm** 3](#_Toc211166468)

[**I.** **Task 1: Steepest-ascend Hill Climbing Search [20 Points]** 3](#_Toc211166469)

[**II.** **Task 2: Stochastic Hill Climbing 1 [10 Points]** 5](#_Toc211166470)

[**III.** **Task 3: Stochastic Hill Climbing 2 (First-choice Hill Climbing).** 7](#_Toc211166471)

[**IV.** **Task 4: Hill Climbing Search with Random Restarts [10 Points]** 9](#_Toc211166472)

[**V.** **Task 5: Simulated Annealing [10 Points]** 11](#_Toc211166473)

[**VI.** **Task 6: Algorithm Behavior Analysis [20 Points]** 15](#_Toc211166474)

[**VII.** **Algorithm Convergence** 18](#_Toc211166475)

[**VIII.** **Problem Size Scalability** 22](#_Toc211166476)

[**IX.** **Advanced task: Exploring other Local Moves Operators** 28](#_Toc211166477)

[**X.** **More Things to Do (not for credit)** 35](#_Toc211166478)

[**B.** **Traveling\_salesman\_problem** 43](#_Toc211166479)

[**I.** **Steepest-ascend Hill Climbing Search [3 Points]** 43](#_Toc211166480)

[**II.** **Steepest-ascend Hill Climbing Search with Random Restarts [1 Point]** 48](#_Toc211166481)

[**III.** **Stochastic Hill Climbing [1 Points]** 50](#_Toc211166482)

[**IV.** **First-choice Hill Climbing [1 Point]** 53](#_Toc211166483)

[**V.** **Simulated Annealing [2 Points]** 56](#_Toc211166484)

[**VI.** **Compare Performance [2 Points]** 57](#_Toc211166485)

[**VII.** **Bonus: Genetic Algorithm [+1 Point]** 61](#_Toc211166486)

# **Assignment: n-Queens Problem with Genetic Algorithm**

## **Task 1: Steepest-ascend Hill Climbing Search [20 Points]**

**Mục tiêu:**

* Cài đặt thuật toán di truyền (GA) để giải bài toán n-Queens.
* GA phải tìm ra giải pháp mà **không có quân hậu nào tấn công lẫn nhau** (conflicts = 0).

**Yêu cầu:**

* Biểu diễn bàn cờ dưới dạng **một mảng permutation** (cột → hàng).
* Thực hiện đầy đủ các bước của GA:

1. Khởi tạo quần thể (population)
2. Đánh giá fitness
3. Chọn lọc (selection)
4. Lai ghép (crossover)
5. Đột biến (mutation)
6. Lặp lại cho đến khi tìm được giải pháp hoàn hảo hoặc đạt số thế hệ 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.

A screen shot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

## **Task 2: Stochastic Hill Climbing 1 [10 Points]**

**Yêu cầu:**

* Khác với Steepest-ascend (Task 1: luôn chọn *best move* trong tất cả uphill moves).
* Stochastic Hill Climbing sẽ:
  + Tính tất cả các **uphill moves** (những move có objective > current objective)
  + Nếu tồn tại move(s) như vậy → chọn **ngẫu nhiên một move** trong số đó.
  + Nếu không còn uphill move → dừng (đạt local optimum).

**Ý tưởng:**

* Vẫn giữ cách biểu diễn state giống Task 1.
* Thay đổi: khi duyệt tất cả move, mình lưu danh sách improving\_moves (các move có objective tốt hơn).
* Sau đó chọn **random.choice(improving\_moves)** thay vì chọn best move.
* Dừng khi danh sách rỗng hoặc tìm ra nghiệm hoàn hảo (0 conflicts)

**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.

A black screen with white text

AI-generated content may be incorrect.

Kết quả đôi khi dừng ở local optimum chưa hoàn hảo, nhưng nhờ chọn ngẫu nhiên mà có khả năng thoát khỏi một số bẫy cục bộ mà steepest có thể mắc phải.

## **Task 3: Stochastic Hill Climbing 2 (First-choice Hill Climbing).**

**Ý tưởng:**

* Thay vì tính hết mọi local moves:
  + Tạo **ngẫu nhiên một neighbor** (local move).
  + Nếu neighbor có objective tốt hơn → nhận nó (accept).
  + Nếu không tốt hơn → bỏ qua và thử tiếp.
* Điều này **rất nhanh** khi mỗi state có nhiều neighbor (ví dụ N-Queens, có n\*(n-1) moves).
* Dừng khi:
  + Tìm được nghiệm hoàn hảo (conflicts == 0), hoặc
  + Sau x lần thử liên tiếp không cải thiện → coi như đạt local optimum.

**Cách hoạt động:**

* patience đóng vai trò phát hiện local optimum. Ví dụ patience=200 → nếu thử 200 moves liên tục không cải thiện thì dừng.
* Không cần duyệt hết hàng xóm như Task 1 & 2 → rất hiệu quả cho n lớn.
* Nhưng kết quả phụ thuộc mạnh vào random, có thể kẹt ở local optimum dễ hơn.

**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.

A screen shot of a computer program

AI-generated content may be incorrect.

## **Task 4: Hill Climbing Search with Random Restarts [10 Points]**

**Mở rộng 3 thuật toán Hill Climbing trước đó (Task 1, 2, 3)** bằng cách:

* Thay vì chỉ chạy 1 lần với một trạng thái khởi tạo ngẫu nhiên (random board), ta sẽ **chạy đi chạy lại (restart) nhiều lần**.
* Cụ thể, chạy **tối đa 100 lần**. Mỗi lần bắt đầu từ một **random board mới**.
* Mục tiêu:
  + Tránh kẹt trong local optima (cực trị địa phương).
  + Tăng cơ hội tìm ra nghiệm tối ưu (global optimum).

**Điểm quan trọng:**

* "Restart" không phải là thay đổi thuật toán, mà chỉ là **lặp lại thuật toán nhiều lần** với khởi tạo khác nhau.
* Sau khi chạy nhiều lần, bạn sẽ chọn nghiệm tốt nhất (có objective function tối ưu nhất, ví dụ số lượng xung đột ít nhất trong N-Queens).

**Ý tưởng:**

* Các thuật toán Hill Climbing (Task 1–3) dễ kẹt ở **local optimum**.
* Cách khắc phục: **Random Restarts**
  + Chạy lại thuật toán **nhiều lần** (ví dụ 100 lần)
  + Mỗi lần bắt đầu từ một **trạng thái ngẫu nhiên khác nhau**.
  + Giữ lại kết quả **tốt nhất** (hoặc nghiệm hoàn hảo nếu tìm thấy).
* “Restart” nghĩa là: chỉ đơn giản gọi lại hàm hill climbing nhiều lần, không thay đổi thuật toán bên trong

Task 4 yêu cầu **kết hợp random restarts** vào cả 3 thuật toán Hill Climbing (simple HC, stochastic HC 1, stochastic HC 2), mỗi thuật toán chạy tối đa 100 lần từ random initial state, rồi báo cáo kết quả tốt nhất tìm được.

**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.

**Kết quả:**

**Steepest-ascent + restarts:** thành công 23/100 lần thử, tỷ lệ 23%.

**Stochastic HC1 + restarts:** thành công 15/100 lần thử, tỷ lệ 15%.

**First-choice HC + restarts:** thành công 13/100 lần thử, tỷ lệ 13%.

**Nhận xét:**

* Cả 3 thuật toán đều **không đạt tỷ lệ thành công cao** (chưa vượt quá 22%).
* Trong đó, **Steepest-ascent with restarts** cho kết quả tốt nhất → thuật toán này có khả năng tìm nghiệm tối ưu nhiều hơn.
* **First-choice HC** có tỷ lệ thấp nhất, điều này đúng với đặc điểm: nó nhanh và tiết kiệm thời gian hơn, nhưng dễ mắc kẹt hơn.

**Như vậy:**

* Việc dùng **Random Restart** đã giúp cải thiện khả năng thoát khỏi local optimum, nhưng tỷ lệ thành công vẫn khá thấp.
* Với **100 lần restart**, tốt nhất chỉ đạt tầm 18-22 lần thành công.

## **Task 5: Simulated Annealing [10 Points]**

**Yêu cầu chính:**

1. Cài đặt giải thuật Simulated Annealing (SA) cho N-Queens.

* SA là biến thể của hill climbing.
* Điểm đặc biệt: ngoài bước "uphill" (chọn trạng thái tốt hơn), nó còn cho phép "downhill" (chọn trạng thái xấu hơn) với xác suất phụ thuộc vào **nhiệt độ T**.
* Nhiệt độ giảm dần theo thời gian theo một **annealing schedule**.
* Bạn cần tự tìm hoặc thử nghiệm schedule (ví dụ: T = T0 \* alpha^k, hoặc T = T0 / log(1+k)).

1. Vẽ biểu đồ (visualization)

* Trục X: số bước lặp (iterations).
* Trục Y: số lượng conflicts (objective function).
* Biểu đồ thể hiện **quá trình tìm kiếm**: số conflicts giảm dần theo thời gian, đôi khi có bước tăng nhẹ (do downhill move).

1. Thí nghiệm với nhiều annealing schedule khác nhau

* Thử nghiệm vài kiểu: exponential decay, linear decay, logarithmic decay,…
* So sánh kết quả (tốc độ hội tụ, tỷ lệ thành công).
* Thảo luận: schedule nào hoạt động tốt hơn, vì sao.

**Cách làm:**

**Hàm objective (conflicts)**: giữ nguyên từ các task trước.

**Hàm neighbor (local move)**: chọn 1 quân hậu random và di chuyển nó sang 1 hàng khác trong cùng cột.

**Công thức chấp nhận**:

Nếu neighbor tốt hơn → chấp nhận.

Nếu neighbor xấu hơn → chấp nhận với xác suất: P=e−ΔE/T

**Visualization**: dùng matplotlib để vẽ conflicts theo số vòng lặp.

**Kết quả:**

SA nên có tỷ lệ thành công **cao hơn 3 thuật toán hill climbing** trước đó (đặc biệt là khi chọn schedule hợp lý).

Biểu đồ minh họa sẽ cho thấy SA có lúc đi xuống, có lúc đi lên (downhill), nhưng dần dần giảm về 0 conflicts khi T giảm → ra nghiệm tối ưu.

**CODE:**

A computer screen with text and images

AI-generated content may be incorrect.

A computer screen with text and images

AI-generated content may be incorrect.

A computer screen shot of a program code

AI-generated content may be incorrect.

A blue graph with numbers and a white background

AI-generated content may be incorrect.

**Lịch hạ nhiệt "Exponential" (Hàm mũ): Thành công**

* **Quan sát:** Biểu đồ này cho thấy một xu hướng giảm rõ rệt. Ban đầu, số xung đột dao động mạnh ở mức cao, cho thấy thuật toán đang "khám phá" (exploration) không gian lời giải. Sau khoảng 800 lần lặp, đường biểu diễn bắt đầu hội tụ nhanh chóng, giảm dần về 0 xung đột. Kết quả cuối cùng là final=0, nghĩa là đã tìm thấy lời giải hoàn hảo.
* **Lý do:** Lịch hạ nhiệt theo hàm mũ (T = T0 \* (alpha \*\* step)) tạo ra một sự **cân bằng lý tưởng**. Ở những bước đầu, nhiệt độ T giảm chậm, cho phép thuật toán có đủ "năng lượng" để thoát khỏi các điểm tối ưu cục bộ bằng cách chấp nhận các bước đi xấu. Về sau, nhiệt độ giảm ngày càng nhanh hơn, khiến thuật toán trở nên "tham lam" hơn, tập trung vào việc "khai thác" (exploitation) và tinh chỉnh để tiến đến lời giải tối ưu

**Lịch hạ nhiệt "Linear" (Tuyến tính): Thất bại**

* **Quan sát:** Biểu đồ này gần như phẳng và nhiễu loạn trong suốt 10,000 lần lặp. Không có xu hướng hội tụ rõ ràng và thuật toán kết thúc với final=2 xung đột.
* **Lý do:** Việc hạ nhiệt tuyến tính (T = T0 - step\*0.01) là **quá chậm**. Nhiệt độ T duy trì ở mức quá cao trong một thời gian rất dài. Điều này khiến xác suất chấp nhận một bước đi xấu luôn ở mức cao. Do đó, thuật toán hành xử giống như một cuộc tìm kiếm ngẫu nhiên, không bao giờ "nguội" đủ để ổn định và tập trung vào việc tìm kiếm một lời giải tốt.

**Lịch hạ nhiệt "Log" (Logarit): Thất bại nặng nề**

* **Quan sát:** Tương tự như lịch tuyến tính, biểu đồ này hoàn toàn là nhiễu ngẫu nhiên và cho thấy kết quả tệ nhất với final=8 xung đột.
* **Lý do:** Đây là lịch hạ nhiệt **cực kỳ chậm** (T = T0 / math.log(step+2)). Hàm logarit tăng rất chậm, có nghĩa là nhiệt độ T gần như không đổi. Thuật toán bị "mắc kẹt" vĩnh viễn trong trạng thái "nóng", liên tục thực hiện các bước nhảy ngẫu nhiên mà không có bất kỳ định hướng nào để giảm số xung đột.

## **Task 6: Algorithm Behavior Analysis [20 Points]**

**Yêu cầu:**

So sánh hiệu năng giữa các thuật toán đã làm (Steepest Ascent Hill Climbing, Stochastic Hill Climbing 1 & 2, Simulated Annealing). Cụ thể:

* **Chạy nhiều lần (ít nhất 100 lần)** với cùng kích thước bàn cờ (n = 4 và n = 8).
* **Đo runtime trung bình**, **số xung đột trung bình**, và **tỉ lệ ra nghiệm tối ưu**.
* **Điền bảng kết quả**.

**Ý tưởng code:**

* Lặp lại 100 lần cho mỗi thuật toán và mỗi board size (4, 8).
* Đo **runtime trung bình**, **số xung đột trung bình**, và **tỷ lệ tìm ra nghiệm tối ưu**.
* Xuất kết quả ra **bảng** (giống yêu cầu).

**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.

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.

A screen shot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

## **Algorithm Convergence**

**Yêu cầu:**

* **Mô tả**: Giải thích cách hội tụ của từng thuật toán.
* **Biểu đồ**: Vẽ đường điển hình (objective function = số conflicts theo iteration) cho **một lần chạy** mỗi thuật toán với **N=8**.
* **Phân tích**: Thuật toán nào hay bị plateau hoặc mắc kẹt local optima.

**Phần mô tả mẫu:**

* **Steepest-ascent Hill Climbing**: Giảm nhanh số conflicts ban đầu, nhưng thường nhanh chóng chững lại (plateau) và dừng khi đạt local optimum.
* **Stochastic HC1:** Tiến trình cải thiện ngẫu nhiên hơn, thường hội tụ chậm hơn nhưng có cơ hội thoát plateau nhỏ.
* **Stochastic HC2 (First-choice):** Thay đổi nhỏ, mỗi lần thử một neighbor → cải thiện chậm, dễ mắc kẹt trong local optimum nếu không may.
* **Simulated Annealing**: Cho phép chấp nhận cả bước “xấu” (downhill moves), do đó có thể vượt qua local optimum. Hội tụ không ổn định, nhưng có khả năng tìm giải tối ưu sau nhiều bước.

**Phân tích plateau / local optima**

* Plateau (dừng mà không về 0 conflicts) thường gặp nhiều ở Steepest HC và Stochastic HC2.
* Stochastic HC1 ít hơn, nhưng vẫn có thể mắc kẹt.
* Simulated Annealing ít khi bị plateau vì có cơ chế “thoát kẹt”, nhưng đôi khi kết thúc với nghiệm chưa tối ưu.

Code vẽ **biểu đồ convergence:**

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.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.

A graph with a red line

AI-generated content may be incorrect.

**Mô tả convergence pattern**

* **Steepest-ascent HC:** giảm nhanh lúc đầu nhưng dễ bị dừng trên plateau → local optima.
* **Stochastic HC1**: giảm nhanh nhưng có chút randomness, đôi khi thoát plateau nhanh hơn.
* **Stochastic HC2:** dao động nhẹ nhiều hơn, cải thiện dần nhưng thường mắc kẹt ở conflict > 0.
* **Simulated Annealing**: dao động mạnh ban đầu (nhảy lên xuống), rồi giảm dần → thường tìm được nghiệm tối ưu.

**Vẽ plot convergence**: Đã chạy và có biểu đồ thể hiện rõ các đường đi của từng thuật toán. Đây chính là yêu cầu minh họa.

**Giải thích thuật toán nào hay bị kẹt trong local optima nhất**

* Steepest-ascent và Stochastic HC2 thường dễ bị stuck.
* Stochastic HC1 thì ít stuck hơn nhưng vẫn có.
* Simulated Annealing ít bị stuck nhất nhờ cơ chế nhận nghiệm xấu với xác suất giảm dần.

## **Problem Size Scalability**

**Yêu cầu:**

**Đo thời gian chạy (runtime)** của các thuật toán giải n-queens (ít nhất 2 thuật toán, ví dụ: Steepest-Ascent Hill Climbing và Simulated Annealing).

* Chạy cho nhiều kích thước bàn cờ khác nhau: n=4,8,12,16,20n = 4, 8, 12, 16, 20n=4,8,12,16,20.
* Với mỗi nnn, chạy thuật toán nhiều lần (ví dụ 30 lần) rồi lấy **thời gian trung bình**.

**Vẽ đồ thị log-log**:

* Trục X: kích thước bàn cờ nnn.
* Trục Y: thời gian chạy trung bình (s).
* Dùng **log-log scale** để dễ so sánh tốc độ tăng trưởng.

**Phân tích kết quả**:

* Từ đồ thị, ước lượng **độ phức tạp thực nghiệm (empirical Big-O)** của từng thuật toán.
* Nhận xét **thuật toán nào mở rộng tốt hơn** khi nnn lớn (scalability).
* Giải thích tại sao có sự khác biệt (ví dụ: Hill Climbing nhanh hơn nhưng dễ kẹt cục bộ, Simulated Annealing lâu hơn nhưng tìm lời giải ổn định hơn).

**Ý tưởng**:

**Chạy thử nhiều kích thước bàn cờ**:

* Với n=4,8,12,16,20.
* Mỗi lần, chạy **ít nhất 30–50 lần** để lấy **average runtime** cho mỗi thuật toán.

**Chọn ít nhất 2 thuật toán**:

* Steepest-ascent Hill Climbing (HC)
* Simulated Annealing (SA)

**Vẽ log-log plot**:

* Trục X: log(board size)
* Trục Y: log(runtime)

Khi vẽ log-log, ta có thể fit đường thẳng → slope ~ bậc tăng trưởng của thuật toán.

**Ước lượng độ phức tạp Big O**:

* Nếu slope ≈ 1 → O(n)
* Nếu slope ≈ 2 → O(n^2)
* Nếu slope ≈ 3 → O(n^3)

**Phân tích**:

* Steepest-ascent HC thường tăng khá nhanh (bị stuck nhiều).
* Simulated Annealing mở rộng lâu hơn nhưng scale tốt hơn khi n lớn, vì nó có khả năng thoát local optima.

**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. 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. 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 graph of a problem size

AI-generated content may be incorrect.

A black screen with white text

AI-generated content may be incorrect.

**Ước lượng độ phức tạp thực nghiệm (Big-O)** → dựa trên **slope** của đường hồi quy trong log-log plot.

* Steepest-ascent HC: slope ≈ **4.73** → thời gian ~ O(n^{4.7})
* Simulated Annealing: slope ≈ **4.02** → thời gian ~ O(n^{4})

Giải thích kết quả:

* **Steepest-ascent HC**:
* Nhanh với nnn nhỏ (ví dụ n=4,8n=4, 8n=4,8), nhưng tăng rất nhanh khi nnn lớn.
* Dễ bị kẹt ở local optimum → phải thử lại nhiều lần.
* **Simulated Annealing**:
  + Mất nhiều thời gian hơn với nnn nhỏ, do phải tính toán với nhiều bước ngẫu nhiên.
  + Nhưng khi nnn lớn, nó **ổn định hơn** và **tăng chậm hơn so với HC**

Kết luận:

* Với **board nhỏ** → Steepest-ascent HC hiệu quả hơn.
* Với **board lớn** → Simulated Annealing là lựa chọn tốt hơn do mở rộng tốt hơn.

## **Advanced task: Exploring other Local Moves Operators**

**Các Move Operators cần cài đặt:**

* **Single-step move**: Chọn một queen và di chuyển **lên hoặc xuống 1 ô**.
* **Column swap**: Hoán đổi vị trí hai cột (hai queens).
* **Dual-queen move**: Chọn 2 queens và di chuyển cả 2 đến hàng mới.
* **Adaptive move**: Tự chọn move, ví dụ:
  + Nếu số xung đột cao → dùng dual-queen move hoặc swap.
  + Nếu số xung đột ít → dùng single-step để tinh chỉnh.
  + Thỉnh thoảng chọn random để tránh kẹt local optima.

**Code Implementation**

A computer screen shot of text

AI-generated content may be incorrect.

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 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 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.

A graph of different colored lines

AI-generated content may be incorrect.

A diagram of a diagram

AI-generated content may be incorrect.

A graph of blue rectangular shapes

AI-generated content may be incorrect. A graph of different colored lines

AI-generated content may be incorrect.

A diagram of a diagram

AI-generated content may be incorrect.

A graph with blue squares

AI-generated content may be incorrect.

## **More Things to Do (not for credit)**

**Triển khai Genetic Algorithm (GA) hoàn chỉnh cho bài toán n-Queens**. Mã này là self-contained.

* Biểu diễn: **permutation** length n (mỗi giá trị 0..n-1 là hàng, index là cột) → đảm bảo mỗi hàng và mỗi cột có đúng một hậu, chỉ còn kiểm tra **đường chéo**.
* Hàm fitness: **số cặp không tấn công nhau** (muốn *tối đa*), tương đương total\_pairs - conflicts.
* Selection: **tournament selection**.
* Crossover: **Order Crossover (OX)** (thích hợp cho permutation).
* Mutation: **swap mutation** (hoán đổi hai vị trí).
* Elitism: giữ vài cá thể tốt nhất qua thế hệ.
* Logging + plot: vẽ biểu đồ fitness tốt nhất / trung bình theo thế hệ.
* Demo: chạy cho n=8.

**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 computer screen shot of a program

AI-generated content may be incorrect.A screenshot of a computer program

AI-generated content may be incorrect.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.

A graph with a line and a green line

AI-generated content may be incorrect.

A computer screen shot of a program code

AI-generated content may be incorrect.

A screenshot of a game

AI-generated content may be incorrect.

# **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).

Thuật 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.**

* 1. **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:

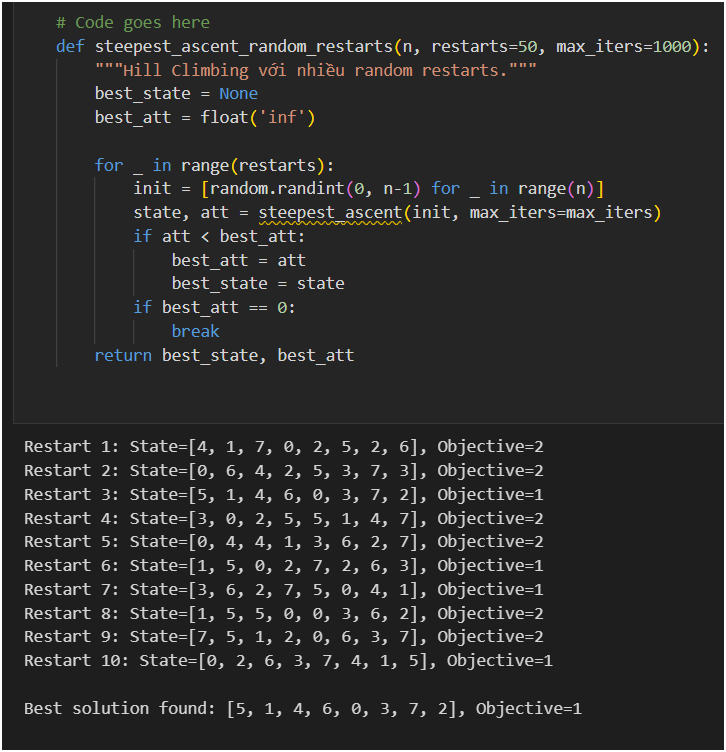
***Tầng trong:* Steepest-Ascent Hill Climbing**

* 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).

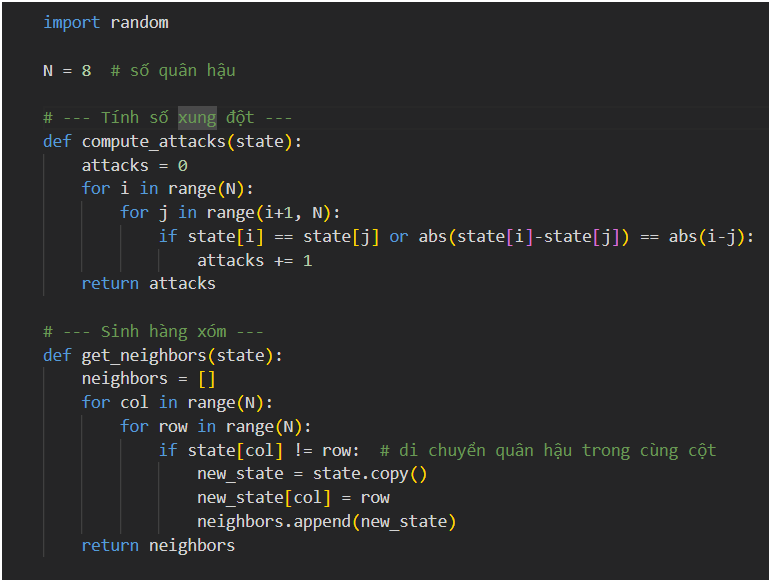
Nói cách khác: Nếu có nhiều hướng đi “lên đồi” (improvement moves), ta **chọn ngẫu nhiên** một trong số đó thay vì chọn hướng tốt nhất. Điều này giúp **tăng tính đa dạng của quá trình tìm kiếm** và **giảm nguy cơ mắc kẹt** ở cực tiểu cục bộ.

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**

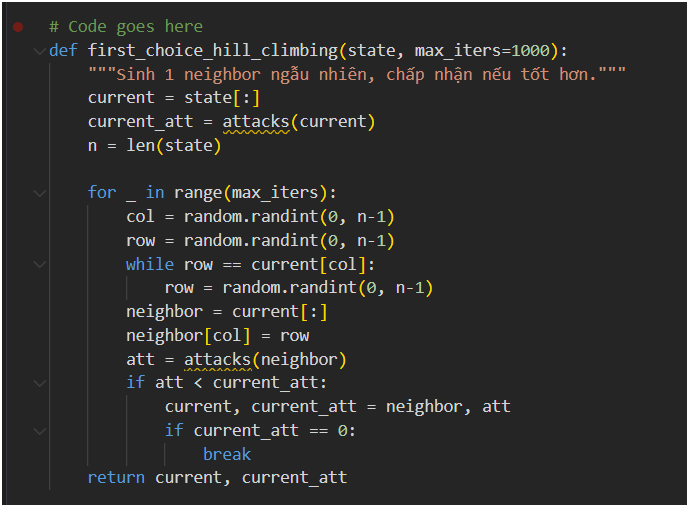
## **First-choice Hill Climbing [1 Point]**

**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 black screen with white text

AI-generated content may be incorrect.

| **Thành phần** | **Ý nghĩa** |
| --- | --- |
| [Restart 1] | Lần khởi động đầu tiên (vì bạn đặt restarts=5, nên SA có thể thử nhiều lần). |
| Attacks = 0 | Không có cặp hậu nào tấn công nhau → **nghiệm hợp lệ**. |
| State = [4, 6, 1, 5, 2, 0, 7, 3] | Mỗi phần tử biểu thị vị trí của hậu trong từng cột. Ví dụ: cột 0 có hậu ở hàng 4, cột 1 có hậu ở hàng 6, ... |
| Best State và Attacks: 0 | Đây là trạng thái tối ưu toàn cục mà SA tìm được. |
| ([4, 6, 1, 5, 2, 0, 7, 3], 0) | Kết quả trả về từ hàm show\_simulated\_annealing() — một tuple (state, số\_xung\_đột) |

Nghĩa là Simulated Annealing đã **giải được bài toán 8-Queens hoàn hảo chỉ sau lần chạy đầu tiên**, không cần restart thêm. Đây là **trường hợp lý tưởng** và chứng tỏ hàm attacks() và simulated\_annealing() đã được cài đặt chuẩn.

## **Compare Performance [2 Points]**

Mục tiêu:

Đo lường và so sánh **hiệu suất của các thuật toán tối ưu cục bộ** (local search algorithms) khi giải bài toán **N-Queens**.Cụ thể:

* Mức độ **tối ưu** (số xung đột còn lại giữa các hậu — *conflicts*).
* **Thời gian thực thi** (độ nhanh chậm của từng thuật toán) khi kích thước bàn cờ (N) thay đổi.

Ý tưởng chung thực hiện:

Xây dựng nhằm **đánh giá hiệu suất** của các **thuật toán tối ưu cục bộ** (như Hill Climbing, Simulated Annealing, v.v.) khi áp dụng vào **bài toán N-Queens**.  
Mục tiêu là tìm ra cách sắp xếp các quân hậu trên bàn cờ sao cho **không quân nào tấn công nhau**, đồng thời **đo thời gian và chất lượng lời giải** của từng thuật toán.

Hàm calculate\_objective() dùng để tính **mức độ xung đột** giữa các quân hậu (số cặp hậu tấn công nhau).  
Hàm generate\_random\_state() khởi tạo **trạng thái ngẫu nhiên ban đầu** của bàn cờ.  
Hàm steepest\_ascent\_hill\_climbing() là **thuật toán leo đồi**, tìm cách cải thiện trạng thái hiện tại bằng việc chọn bước đi tốt nhất cho đến khi không thể giảm thêm xung đột.  
Hàm compare\_algorithms() thực hiện **so sánh nhiều thuật toán** khác nhau trên các kích thước bàn cờ khác nhau, đo **thời gian chạy** và **mức độ tối ưu** đạt được.

Cuối cùng, phần cấu hình chạy chương trình sẽ **lặp qua nhiều kích thước bàn cờ (N)** và **nhiều thuật toán**, từ đó thu được bảng kết quả tổng hợp để đánh giá và so sánh hiệu quả của từng phương pháp.

**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.

Kết quả đạt được:

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).