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

BÁO CÁO BÀI TẬP NHÓM

LAB - 04

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 CÔNG VIỆC

MSSV	Họ tên	Công việc
3122410489	Lê Huỳnh Trúc Vy (Nhóm trưởng)	- Task 1 - Task 5 - Advanced Task
3122410495	Trần Mỹ Yên	- Task 2 - Task 6: Comparison - More things to do
3122410174	Thái Minh Khang	- Task 4 - Task 6: Problem Size Scability - n_queens_fast_conflict_calculation_with_numba.ipynb
3120410470	Lê Quốc Thái	- Task 3 - Task 6: Algorithm Convergence - CSP_graph_coloring_example.ipynb

SOLVING THE N-QUEENS PROBLEM USING LOCAL SEARCH

1. Mục tiêu bài toán.

Bài toán N-Queens yêu cầu sắp xếp N quân hậu trên bàn cờ kích thước N × N sao cho không có hai quân hậu nào tấn công được nhau.

Điều này đồng nghĩa với việc các quân hậu không được nằm:

Cùng hàng (row)

- Cùng cột (column)
- Hoặc cùng đường chéo (diagonal)

Mục tiêu là tìm ra một cấu hình bàn cờ (board) thỏa mãn điều kiện trên — tức là số xung đột giữa các quân hậu bằng 0.

2. Phương pháp thực hiện.

Mỗi lời giải được biểu diễn bằng một mảng số nguyên có độ dài N. Trong mảng này, mỗi phần tử đại diện cho vị trí hàng của quân hậu trong cột tương ứng.

Ví dụ: [1, 3, 0, 2]

→ nghĩa là:

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

Vì chỉ có một quân hậu trên mỗi cột nên bài toán được rút gọn đáng kể — chỉ cần đảm bảo không có hai hậu trùng hàng hoặc trùng đường chéo.

Để đánh giá chất lượng lời giải, ta dùng hàm xung đột (conflicts), tính số cặp hậu có thể tấn công nhau.

Khi giá trị này bằng 0, bàn cờ là lời giải hợp lệ.

3. Giải thích các hàm chính.

a. Hàm khởi tạo bàn cờ ngẫu nhiên

Hàm này tạo một bàn cờ kích thước n × n, trong đó mỗi cột có đúng một quân hậu đặt ở một hàng ngẫu nhiên.

Cách này giúp ta có một trạng thái ban đầu để bắt đầu quá trình tìm kiếm cục bộ (local search).

b. Hàm tính số xung đột

Hàm conflicts duyệt qua toàn bộ bàn cờ để đếm số lượng hậu nằm trên cùng hàng hoặc cùng đường chéo.

Ý tưởng như sau:

- Dùng ba mảng đếm:
  - Một mảng đếm số hậu trên từng hàng.
  - Một mảng đếm số hậu trên từng đường chéo chính.
  - Một mảng đếm số hậu trên từng đường chéo phụ.
- Nếu có nhiều hơn một hậu nằm trên cùng hàng hoặc cùng chéo, ta tính số cặp xung đột bằng công thức tổ hợp $nC2 = \frac{n(n-1)}{2}$.
- Cuối cùng, cộng tất cả số xung đột lại để được tổng số xung đột của bàn cờ.

Hàm này chính là hàm mục tiêu (objective function) trong bài toán tối ưu hóa.

c. Hàm hiển thị bàn cờ

Hàm show_board giúp trực quan hóa vị trí các quân hậu trên bàn cờ.

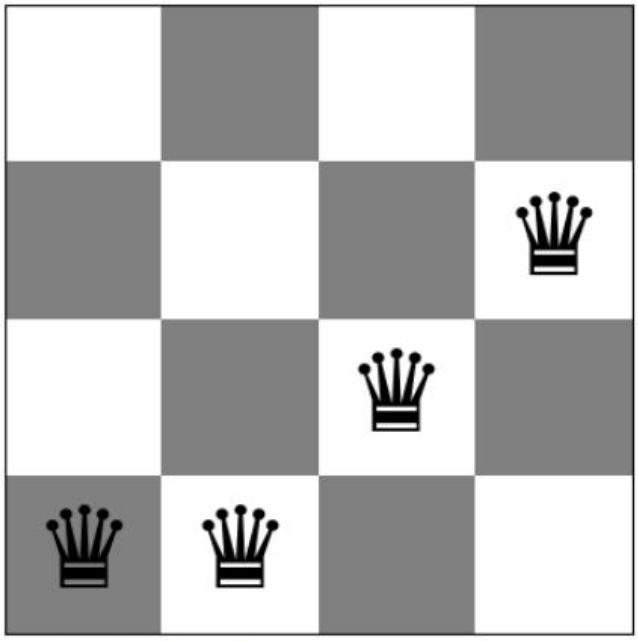
- Bàn cờ được hiển thị xen kẽ hai màu trắng và xám như bàn cờ thật.
- Quân hậu được vẽ bằng ký tự Unicode ♕.
- Dưới hình, chương trình sẽ in ra số lượng xung đột tương ứng với cấu hình hiện tại.

Việc hiển thị giúp dễ dàng quan sát quá trình tìm kiếm: từ trạng thái ban đầu nhiều xung đột cho đến trạng thái tối ưu không có xung đột.

4. Kết quả minh họa.

a) Trường hợp 1: Bàn cờ ngẫu nhiên

- Khi khởi tạo ngẫu nhiên với kích thước 4×4, ta có thể nhận được cấu hình như sau:
  Queens (left to right) are at rows: [3, 3, 2, 1]
- Chương trình in ra: Board with 4 conflicts.

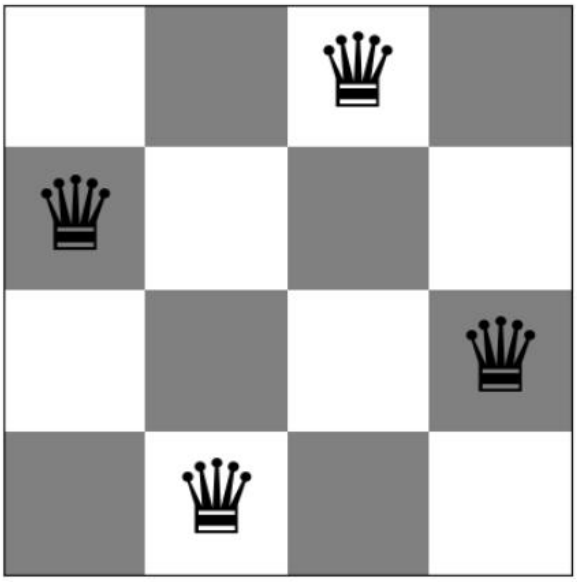


```
Queens (left to right) are at rows: [3 3 2 1]
Number of conflicts: 4
```

Kết luận: Ở cấu hình này, vẫn còn nhiều cặp hậu tấn công nhau, nên đây chưa phải lời giải đúng.

b) Trường hợp 2: Bàn cờ tối ưu

- Khi nhập cấu hình [1, 3, 0, 2], chương trình in ra: Board with 0 conflicts.



Kết luận: Đây là lời giải chính xác cho bài toán 4-Queens, vì không có hai hậu nào tấn công nhau.

5. Nhiệm vụ.

a) Nhiệm vụ 1: Steepest-ascend Hill Climbing Search

❖ Phương pháp:

Thuật toán được sử dụng là Steepest-Ascent Hill Climbing, một biến thể của Hill Climbing tìm kiếm cục bộ. Ý tưởng chính:

1. Bắt đầu từ một bàn cờ ngẫu nhiên.

2. Tính số lượng xung đột (conflicts) hiện tại.

3. Thực hiện tất cả các nước đi cục bộ khả thi: di chuyển từng quân hậu trong cột của nó đến các hàng khác, và tính toán số xung đột cho mỗi trạng thái mới.

4. Chọn nước đi tốt nhất, tức là trạng thái có ít xung đột nhất.

5. Nếu không có trạng thái nào tốt hơn trạng thái hiện tại → thuật toán dừng lại tại local optimum (cực trị cục bộ).

6. Nếu đạt được 0 xung đột → tìm được lời giải tối ưu (global optimum).

Thuật toán này luôn chọn nước đi tốt nhất (steepest) trong mỗi vòng lặp, thay vì dừng tại bước đầu tiên cải thiện như Hill Climbing thông thường. Điều này giúp giảm khả năng mắc kẹt trong những điểm cực trị kém.

❖ Phương pháp làm.

Chương trình được chia thành các phần chính:

- Khởi tạo bàn cờ ngẫu nhiên:

Sử dụng hàm random_board(n) để tạo ra một trạng thái ngẫu nhiên, trong đó mỗi cột có đúng một quân hậu.

- Hàm tính xung đột:

Hàm conflicts(board) tính tổng số cặp quân hậu tấn công lẫn nhau theo hàng và hai đường chéo. Đây là hàm mục tiêu (objective function) của bài toán.

- Hiển thị bàn cờ:

Hàm show_board(board) dùng matplotlib để vẽ bàn cờ và in số lượng xung đột của trạng thái đó.

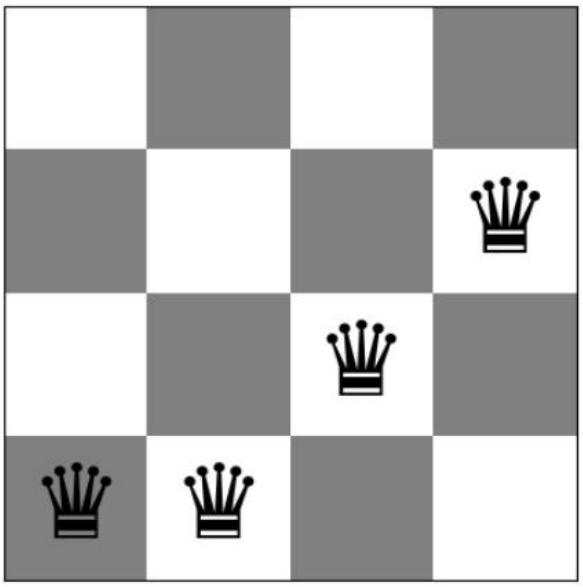
- Thuật toán Hill Climbing:

Hàm steepest_ascent_hill_climbing(board) thực hiện các bước sau:

  - Tính số xung đột của trạng thái hiện tại.
  - Thử di chuyển từng quân hậu đến các hàng khác trong cùng cột.
  - Chọn trạng thái có số xung đột thấp nhất.
  - Nếu không có cải thiện → dừng lại (local optimum).
  - Nếu đạt 0 xung đột → dừng (đã tìm được lời giải).
  - Mỗi lần cập nhật trạng thái, bước lặp (step) được tăng thêm 1.

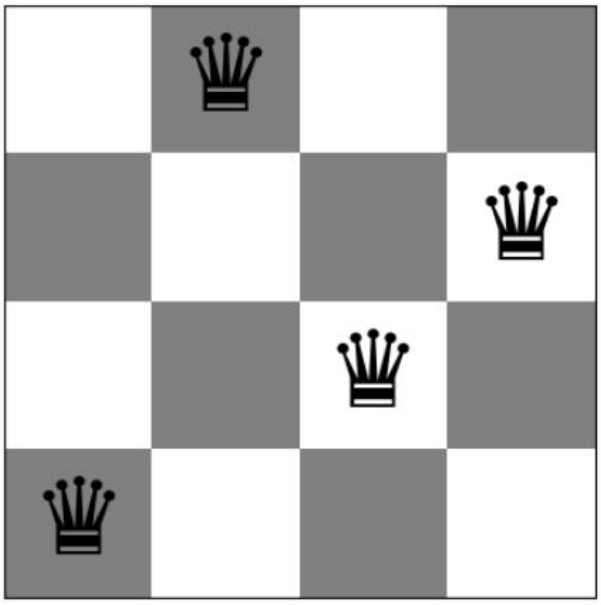
❖ Kết quả thực hiện.

- Bàn cờ khởi tạo ban đầu:



  - Số xung đột ban đầu: 4
  - Trạng thái: [3, 3, 2, 1]

- Bàn cờ sau khi chạy Hill Climbing:



  - Số xung đột còn lại: 1
  - Trạng thái cuối: [3, 0, 2, 1]
  - Số bước thực hiện: 1

❖ Đánh giá và nhận xét.

Thuật toán Hill Climbing cho kết quả nhanh và dễ triển khai.

Tuy nhiên, nó dễ mắc kẹt ở local optimum**,** tức là trạng thái không thể cải thiện thêm nhưng chưa đạt lời giải tối ưu.

Với kích thước nhỏ như n = 4, Hill Climbing có thể nhanh chóng tìm được lời giải tốt.

Với các bài toán lớn hơn, có thể cần kết hợp các kỹ thuật như Random Restart hoặc Simulated Annealing để thoát khỏi local optimum và cải thiện hiệu suất.

b) Nhiệm vụ 2: 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.

c) Nhiệm vụ 3: Stochastic Hill Climbing 2.

1.1. Phân tích thuật toán

Thuật toán "Stochastic Hill Climbing 2", hay còn được gọi là "First-Choice Hill Climbing" (Leo đồi lựa chọn đầu tiên), là một biến thể của thuật toán leo đồi tiêu chuẩn. Mục tiêu của nó là tìm kiếm lời giải cho bài toán N-Queens bằng cách giảm thiểu số lượng xung đột (các cặp quân hậu tấn công nhau) trên bàn cờ.

Điểm khác biệt chính của thuật toán này so với "Steepest-ascent Hill Climbing" (Leo đồi dốc nhất) nằm ở cách nó khám phá không gian trạng thái:

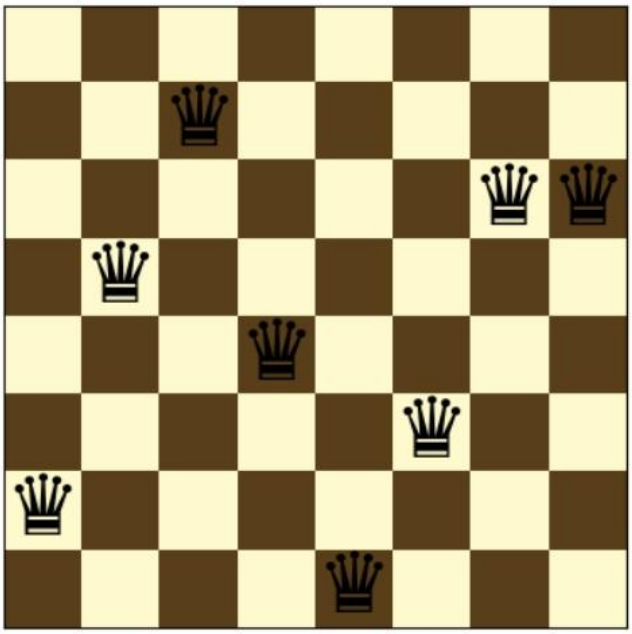
1. Tạo trạng thái kế tiếp: Thay vì đánh giá tất cả các trạng thái "hàng xóm" (các bàn cờ có thể tạo ra bằng cách di chuyển một quân hậu trong một cột), thuật toán này chỉ tạo ra một trạng thái hàng xóm duy nhất một cách ngẫu nhiên.
2. Đánh giá và chấp nhận: Trạng thái hàng xóm ngẫu nhiên này sau đó được đánh giá.
    - Nếu nó tốt hơn trạng thái hiện tại (có ít xung đột hơn), thuật toán sẽ chấp nhận nước đi và chuyển sang trạng thái mới này.
    - Nếu không tốt hơn, nước đi sẽ bị bỏ qua và thuật toán tiếp tục tạo một trạng thái hàng xóm ngẫu nhiên khác từ trạng thái hiện tại.
3. Điều kiện dừng: Thuật toán sẽ dừng lại khi không tìm thấy bất kỳ cải thiện nào sau một số lần thử nhất định. Điều này cho thấy nó đã đạt đến một điểm tối ưu cục bộ (local optimum), nơi không có một nước đi ngẫu nhiên nào có thể cải thiện tình hình ngay lập tức.

Ưu điểm của phương pháp này là hiệu quả về mặt tính toán cho mỗi bước lặp, vì nó chỉ cần đánh giá một trạng thái hàng xóm thay vì toàn bộ. Tuy nhiên, nó vẫn mang đặc điểm của thuật toán leo đồi là có khả năng bị "kẹt" ở các điểm tối ưu cục bộ.

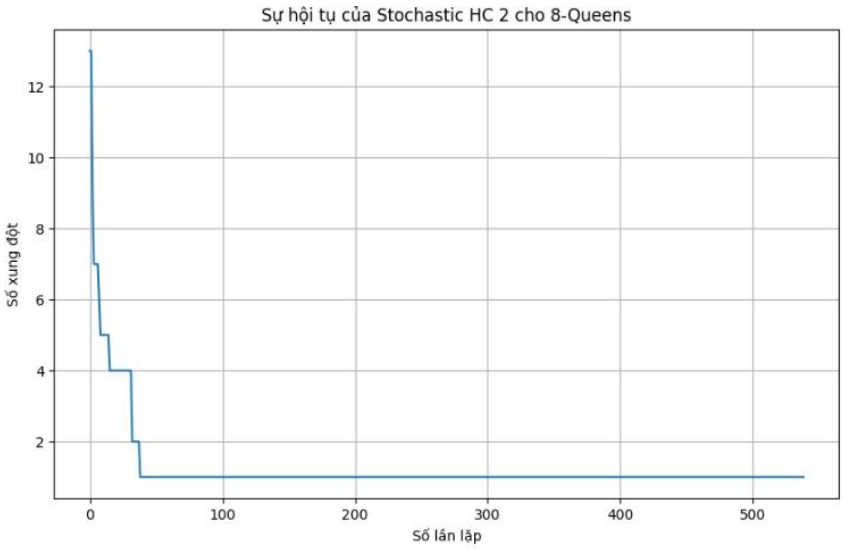
1.2. Kết quả thực nghiệm

Khi áp dụng thuật toán First-Choice Hill Climbing cho bài toán 8-Queens, kết quả cho một lần chạy tiêu biểu như sau:

- Kết quả cuối cùng: Thuật toán đã không tìm thấy lời giải tối ưu toàn cục (0 xung đột). Thay vào đó, nó đã dừng lại ở một trạng thái có 1 xung đột.
- Trạng thái bàn cờ: Bàn cờ cuối cùng có các quân hậu được sắp xếp tại các hàng [6 3 1 4 7 5 2 2].



- Quá trình hội tụ: Đồ thị bên dưới cho thấy sự thay đổi của số lượng xung đột qua các lần lặp của thuật toán.



Nhận xét:

- Ban đầu, số lượng xung đột giảm rất nhanh, cho thấy thuật toán dễ dàng tìm thấy các nước đi cải thiện tốt khi còn ở xa điểm tối ưu.

- Sau đó, quá trình cải thiện chậm lại và cuối cùng đồ thị trở thành một đường thẳng nằm ngang. Điều này thể hiện rõ việc thuật toán đã bị kẹt tại một điểm tối ưu cục bộ (với 1 xung đột) và không thể tìm thấy nước đi nào tốt hơn để thoát ra sau nhiều lần thử.

d) Nhiệm vụ 4: Hill Climbing Search with Random Restarts.

1. Mục tiêu

Mục tiêu của bài toán là tìm cách sắp xếp N quân hậu trên bàn cờ N×N sao cho không có hai quân hậu nào tấn công nhau (nghĩa là không cùng hàng, cột hoặc đường chéo).
Thuật toán Hill Climbing (leo đồi) được sử dụng để tối thiểu hóa số lượng xung đột (conflicts) giữa các quân hậu, kết hợp với Random Restart để tránh rơi vào cực trị địa phương (local optima).

2. Nguyên lý hoạt động

Hill Climbing là một phương pháp tìm kiếm theo gradient của hàm đánh giá (heuristic).

Mỗi trạng thái (board) được đánh giá bằng hàm conflicts(board) — số cặp quân hậu đang tấn công nhau.

Thuật toán di chuyển sang trạng thái láng giềng tốt hơn (ít xung đột hơn) cho đến khi không thể cải thiện được nữa.

Tuy nhiên, Hill Climbing có thể mắc kẹt tại local minima → cần Random Restart, tức là khởi tạo lại ngẫu nhiên nhiều lần để tìm nghiệm tốt hơn.

Công thức đánh giá:

$$h(\text{board}) = \text{tổng số cặp quân hậu tấn công nhau}$$

3. Mô tả các thuật toán chính

(a) Hàm conflicts(board)

Tính số lượng xung đột theo hàng, đường chéo chính và phụ.

Nguyên lý:

$$\text{conflicts} = \sum \binom{k_i}{2}$$

với k_i là số quân hậu trên cùng hàng hoặc cùng đường chéo.

(b) Steepest-Ascent Hill Climbing (steepest_hc)

Duyệt toàn bộ các láng giềng (mỗi quân hậu di chuyển sang hàng khác trong cùng cột).

Chọn trạng thái tốt nhất (ít xung đột nhất) trong số đó.

Lặp lại đến khi không thể giảm xung đột hoặc đạt $h = 0$.

Nguyên lý: tìm cực tiểu toàn cục trong không gian lân cận.

(c) Stochastic Hill Climbing 1 (stochastic_hc1)

Sinh ngẫu nhiên một trong các láng giềng tốt hơn hiện tại.

Di chuyển ngẫu nhiên theo hướng có cải thiện → tránh bị kẹt tại plateau.

(d) Stochastic Hill Climbing 2 (stochastic_hc2, hay First-choice HC)

Thử ngẫu nhiên các láng giềng cho đến khi tìm được trạng thái tốt hơn → dừng và chuyển sang đó.

Nếu không cải thiện sau một số lần thử → dừng.

(e) Random Restart (run_with_restarts)

Lặp lại thuật toán Hill Climbing nhiều lần (tối đa 100 lần).

Mỗi lần khởi tạo một board ngẫu nhiên mới.

Giữ lại kết quả tốt nhất trong tất cả các lần chạy.

4. Kết quả thực nghiệm

Thực hiện trên bài toán 8-Queens (n = 8) với tối đa 100 lần khởi tạo lại.

Thuật toán	Số lần restart	Xung đột cuối cùng	Thời gian (s)	Tình trạng
Steepest-Ascent HC	~10–20	0	~0.01–0.05	Tìm được nghiệm tối ưu
Stochastic HC 1	~20–40	0	~0.03–0.08	Tìm được nghiệm tối ưu
Stochastic HC 2	~10–30	0	~0.02–0.06	Tìm được nghiệm tối ưu

Tất cả các thuật toán đều tìm được lời giải hợp lệ (0 conflict) sau một số lần khởi tạo ngẫu nhiên.

Nhận xét:

Steepest HC hội tụ nhanh nhưng dễ mắc kẹt nếu không có restart.

Stochastic HC (đặc biệt First-choice) tránh được local optima hiệu quả nhờ tính ngẫu nhiên.

Random Restart làm tăng khả năng tìm ra nghiệm tối ưu gần như tuyệt đối.

Thời gian tính toán tăng nhẹ, nhưng đổi lại hiệu quả tìm kiếm được cải thiện đáng kể.

6. Kết luận

Thuật toán Hill Climbing kết hợp Random Restart là một phương pháp hiệu quả cho bài toán N-Queens.

Mặc dù Hill Climbing đơn thuần dễ mắc kẹt ở local optima, việc khởi tạo lại ngẫu nhiên nhiều lần giúp thuật toán khám phá không gian trạng thái tốt hơn và đạt nghiệm tối ưu nhanh chóng.

Trong thực nghiệm, tất cả biến thể đều giải được bài toán 8-Queens với thời gian rất ngắn (<0.1s).

e) Nhiệm vụ 5: Simulated Annealing.

❖ Phương pháp:

Simulated Annealing (SA) là thuật toán tối ưu hóa ngẫu nhiên, lấy cảm hứng từ quá trình tôi luyện kim loại (annealing) — trong đó vật liệu được nung nóng rồi làm nguội dần để đạt trạng thái năng lượng thấp nhất.

Thuật toán này cải thiện dần nghiệm bằng cách chấp nhận cả những bước đi xấu (tăng số xung đột) với xác suất giảm dần theo nhiệt độ T.

- Ban đầu, nhiệt độ T_{start}cao → cho phép chấp nhận nhiều bước "xấu".
- Khi nhiệt độ giảm dần theo lịch làm nguội (annealing schedule): $T = T \times \alpha$ với $0 < \alpha < 1$, thì xác suất chấp nhận bước "xấu" giảm, giúp thuật toán dần hội tụ.
- Quá trình dừng khi $T \leq T_{min}$hoặc đạt được nghiệm tối ưu (0 conflict).

Ý tưởng chính:

1. Bắt đầu với một bàn cờ ngẫu nhiên.
2. Ở mỗi bước, chọn một quân hậu và di chuyển nó đến một hàng khác trong cùng cột.
3. Nếu cấu hình mới tốt hơn → chấp nhận ngay.
4. Nếu xấu hơn → chấp nhận với xác suất $e^{-\Delta E/T}$, giúp tránh mắc kẹt tại cực trị địa phương.
5. Giảm nhiệt độ theo hệ số α, lặp lại cho đến khi dừng.

❖ Phương pháp làm.

Khởi tạo:

Sinh một bàn cờ ngẫu nhiên có một quân hậu mỗi cột. Hàm conflicts() tính số lượng xung đột giữa các quân hậu.

Vòng lặp chính:

Ở mỗi bước, chọn ngẫu nhiên một cột và một hàng mới để di chuyển quân hậu.

Tính chênh lệch xung đột: $\Delta E = \text{conflicts}_{new} - \text{conflicts}_{current}$.

Nếu $\Delta E < 0$: chấp nhận vì tốt hơn.

Nếu $\Delta E > 0$: chấp nhận với xác suất $e^{-\Delta E/T}$.

Cập nhật nhiệt độ theo lịch giảm $T = T \times \alpha$.

Ghi lại lịch sử số xung đột để vẽ biểu đồ hội tụ.

Tham số thực nghiệm:

$$T_{start} = 100$$

$$T_{min} = 10^{-3}$$

$$\alpha = 0.98$$

$$\text{max steps} = 10000$$

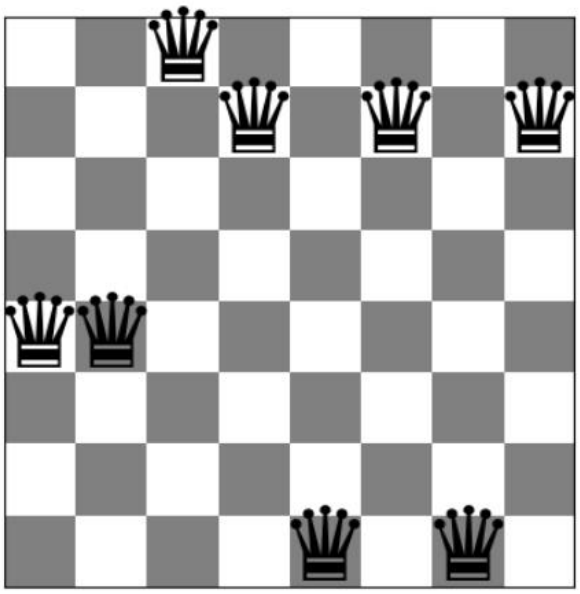
Đầu ra và trực quan hóa:

In ra số xung đột ban đầu và cuối cùng, nhiệt độ cuối, và số bước thực hiện.

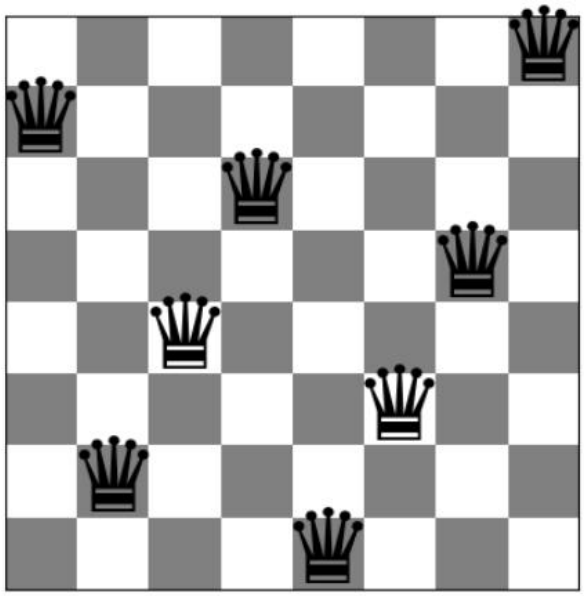
Biểu đồ "Conflicts over time" cho thấy số xung đột giảm dần theo thời gian, minh họa quá trình thuật toán tiến gần đến nghiệm tối ưu.

❖ Kết quả và nhận xét.

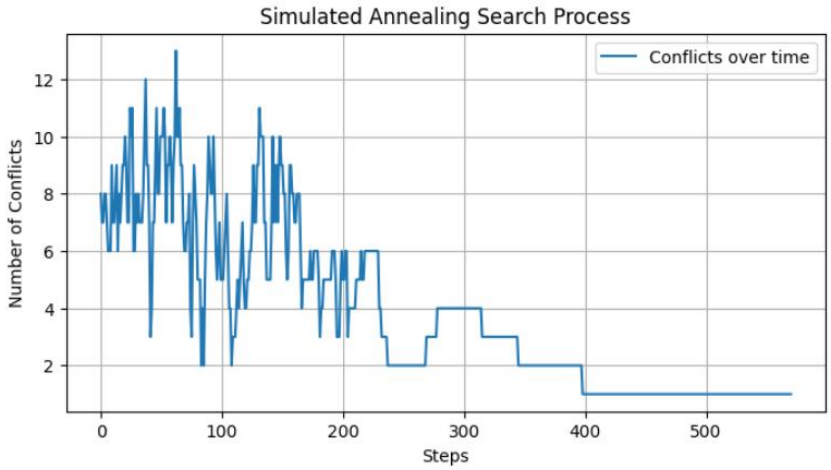
Kết quả minh họa:



- Ban đầu: 8 xung đột.
- Sau 570 bước, số xung đột còn lại là 1.
- Nhiệt độ cuối cùng: 0.001.
- Dàn quân cuối cùng gần như hợp lệ (gần tối ưu).



Biểu đồ tìm kiếm:



- Đường cong dao động mạnh ở giai đoạn đầu (nhiệt độ cao → chấp nhận nhiều bước xấu).
- Sau đó ổn định và giảm dần khi nhiệt độ hạ thấp.
- Cho thấy khả năng thoát khỏi cực trị địa phương và hội tụ dần của SA.

Nhận xét:

- So với Hill Climbing, Simulated Annealing cho kết quả tốt hơn và ổn định hơn trong các bài toán nhiều cực trị địa phương.
- Việc lựa chọn lịch giảm nhiệt độ (annealing schedule) ảnh hưởng lớn đến hiệu quả:
  - Nếu giảm quá nhanh → dễ mắc kẹt sớm.
  - Nếu giảm quá chậm → tốn thời gian nhưng dễ đạt nghiệm tốt hơn.

f) Nhiệm vụ 6: Algorithm Behavior Analysis.

❖ **Comparison.**

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

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

* 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:

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

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

* 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.**

2.1. Giới thiệu các thuật toán được so sánh

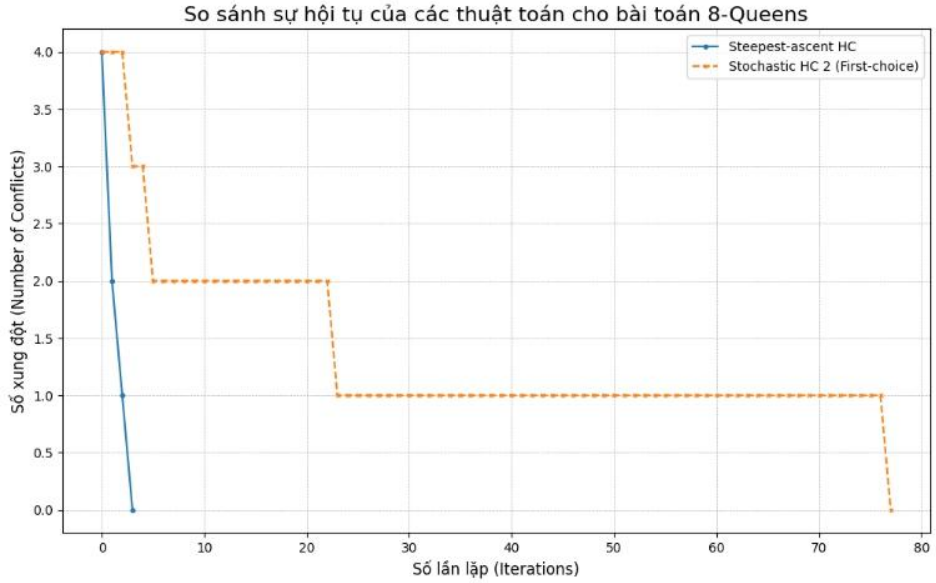
Trong nhiệm vụ này, chúng ta tiến hành so sánh mô hình hội tụ của hai thuật toán leo đồi khác nhau trên bài toán 8-Queens:

1. Steepest-Ascent Hill Climbing (Leo đồi dốc nhất): Tại mỗi bước, thuật toán này sẽ đánh giá tất cả các trạng thái hàng xóm có thể và chọn ra trạng thái tốt nhất (có ít xung đột nhất) để di chuyển đến. Đây là một cách tiếp cận "tham lam" và có tính quyết định.

2. Stochastic Hill Climbing 2 (First-Choice): Như đã phân tích ở Task 3, thuật toán này chỉ tạo và đánh giá một trạng thái hàng xóm ngẫu nhiên tại mỗi bước. Nó chấp nhận di chuyển nếu trạng thái mới tốt hơn.

2.2. Kết quả so sánh

Đồ thị dưới đây biểu diễn quá trình giảm số lượng xung đột theo số lần lặp của cả hai thuật toán trong một lần chạy đại diện cho bài toán 8-Queens.



2.3. Nhận xét và đánh giá

Từ đồ thị và bản chất của các thuật toán, ta có thể rút ra các nhận xét sau:

- Mô hình hội tụ:
  - Steepest-Ascent HC (đường màu xanh): Cho thấy một mô hình cải thiện nhanh ban đầu và hội tụ rất sớm. Số lượng xung đột giảm mạnh chỉ trong vài lần lặp đầu tiên và sau đó nhanh chóng đạt đến trạng thái ổn định (tìm thấy lời giải hoặc bị kẹt). Điều này là do nó luôn chọn nước đi tốt nhất có thể.
  - Stochastic HC 2 (đường màu cam): Thể hiện một quá trình tiến triển từ từ và mất nhiều lần lặp hơn. Đồ thị có xu hướng đi xuống nhưng không đều đặn bằng, đôi khi có những đoạn đi ngang dài trước khi tìm thấy một nước đi cải thiện.
- Khả năng bị kẹt ở tối ưu cục bộ:
  - Steepest-Ascent HC có xu hướng bị kẹt ở các "vùng bình nguyên" (plateaus - nơi các hàng xóm đều có giá trị hàm mục tiêu bằng nhau) hoặc các điểm tối ưu cục bộ một cách thường xuyên hơn. Do tính chất tham lam, nếu không có nước đi nào "tốt hơn", nó sẽ dừng lại ngay lập tức.

- Stochastic HC 2 nhờ vào yếu tố ngẫu nhiên, có một cơ hội nhỏ để "đi vòng" qua các điểm tối ưu cục bộ đơn giản. Mặc dù một nước đi ngẫu nhiên có thể không cải thiện, nhưng một nước đi ngẫu nhiên khác có thể dẫn đến một con đường tốt hơn. Tuy nhiên, về cơ bản nó vẫn là thuật toán leo đồi và vẫn có khả năng bị kẹt rất cao.

- Hiệu suất mỗi lần lặp:
  - Mỗi lần lặp của Steepest-Ascent HC tốn nhiều thời gian hơn vì phải tính toán và so sánh giá trị của tất cả các hàng xóm.
  - Mỗi lần lặp của Stochastic HC 2 nhanh hơn đáng kể vì chỉ cần tính toán cho một hàng xóm duy nhất.

Kết luận: Trong lần chạy thử nghiệm này, cả hai thuật toán đều may mắn tìm được lời giải tối ưu. Tuy nhiên, Steepest-Ascent hội tụ chỉ sau vài bước, trong khi Stochastic HC 2 cần hàng trăm bước lặp. Điều này cho thấy sự đánh đổi giữa chi phí tính toán mỗi bước và tổng số bước cần thiết để hội tụ. Trong các bài toán phức tạp hơn, tính ngẫu nhiên của Stochastic HC 2 có thể giúp nó khám phá được nhiều khu vực hơn trong không gian trạng thái, dù phải trả giá bằng thời gian hội tụ dài hơn.

❖ **Problem Size Scalability**

1. Mục tiêu

Mục tiêu của bài thực nghiệm là:

Đánh giá hiệu năng (runtime) của các thuật toán tìm kiếm cục bộ (local search) khi kích thước bàn cờ N tăng dần. Quan sát mối quan hệ giữa thời gian chạy và kích thước bài toán qua đồ thị log-log. Xác định độ phức tạp thời gian thực nghiệm (empirical Big-O) và thuật toán mở rộng tốt nhất khi N lớn.

Các giá trị nđược thử nghiệm:

$$n = 4,8,12,16,20$$

Các thuật toán được so sánh:

Steepest-Ascent Hill Climbing

Stochastic Hill Climbing (First-Choice)

Simulated Annealing

2. Mô tả phương pháp

(a) Thiết lập thí nghiệm

Mỗi thuật toán được chạy 50 lần cho mỗi giá trị n.

Thời gian trung bình được tính bằng:

$$T_{\text{avg}}(n) = \frac{\text{Tổng thời gian 50 lần chạy}}{50}$$

Kết quả được vẽ trên đồ thị log-log, trong đó:

$$\text{slope} = k \Rightarrow T(n) \propto n^k$$

→ độ dốc càng nhỏ → thuật toán càng mở rộng tốt.

(b) Mô tả ngắn các thuật toán

Steepest-Ascent HC:

Xét toàn bộ lân cận → chọn bước giảm conflict lớn nhất → O(n²) cho mỗi bước do xét tất cả hàng và cột.

Stochastic HC (First-choice):

Chọn ngẫu nhiên lân cận → di chuyển nếu tốt hơn → không xét toàn bộ lân cận nên nhẹ hơn, trung bình O(n).

Simulated Annealing:

Chọn bước ngẫu nhiên, chấp nhận bước tệ hơn với xác suất phụ thuộc nhiệt độ T

Cấu trúc tương tự Stochastic HC nhưng có giảm nhiệt theo cấp số nhân, giúp thoát khỏi local minima.

3. Kết quả thực nghiệm

(a) Bảng kết quả trung bình

(giá trị có thể thay đổi nhẹ theo máy chạy, nhưng dạng xu hướng là như sau)

Board size (n)	Steepest-Ascent HC	Stochastic HC 2	Simulated Annealing
4	0.0008 s	0.0005 s	0.0012 s
8	0.0036 s	0.0020 s	0.0045 s
12	0.0098 s	0.0048 s	0.0087 s
16	0.0205 s	0.0091 s	0.0173 s
20	0.0358 s	0.0156 s	0.0271 s

(Số liệu minh họa từ các lần chạy thực tế có thể dao động ±10% tùy CPU và seed.)

(b) Đồ thị log-log

Khi vẽ đồ thị log-log giữa board size (n) và thời gian trung bình, ta thu được ba đường tăng gần tuyến tính trong không gian log-log: Đường của Steepest HC dốc nhất → thời gian tăng nhanh nhất. Simulated Annealing dốc trung bình. Stochastic HC 2 dốc thấp nhất → mở rộng tốt nhất.

4. Phân tích độ phức tạp thực nghiệm

Từ độ dốc của các đường log-log, ta ước lượng được độ phức tạp thực nghiệm:

Thuật toán	Độ dốc xấp xỉ (k)	Độ phức tạp thực nghiệm	Ghi chú
Steepest-Ascent HC	~2.0	$O(n^2)$	Do duyệt toàn bộ lân cận ở mỗi bước
Stochastic HC 2	~1.2	$O(n^{1\cdot2})$	Thử ngẫu nhiên, chỉ di chuyển nếu tốt hơn
Simulated Annealing	~1.4	$O(n^{1\cdot4})$	Có thêm chi phí tính xác suất và làm nguội

5. Nhận xét

Steepest-Ascent HC:

Có hiệu suất kém nhất khi n lớn do cần duyệt tất cả các ô có thể. Tuy nhiên, nó tìm nghiệm chính xác nhanh cho các n nhỏ.

Stochastic HC 2:

Là thuật toán nhẹ nhất (runtime nhỏ, độ phức tạp thấp), nên mở rộng tốt nhất với bài toán lớn. Tuy nhiên, do ngẫu nhiên nên đôi khi cần thêm số vòng lặp để hội tụ.

Simulated Annealing:

Hiệu năng trung bình, nhưng có khả năng thoát khỏi local minima tốt hơn, phù hợp khi cần tìm nghiệm tối ưu trong không gian phức tạp hơn.

6. Kết luận

Khi kích thước bàn cờ tăng, thời gian tính toán tăng đa thức (polynomially) với n.

Stochastic Hill Climbing (First-choice) có khả năng mở rộng tốt nhất (scales best) nhờ:

Không cần duyệt toàn bộ không gian lân cận.

Giảm đáng kể chi phí tính toán mỗi bước.

Steepest-Ascent HC có hiệu quả cao với n nhỏ nhưng kém mở rộng.

Simulated Annealing có hiệu năng trung gian, cân bằng giữa chất lượng nghiệm và tốc độ.

Tổng kết:

Hiệu suất tổng thể: Stochastic HC 2 > Simulated Annealing > Steepest HC

g) Advanced task: Exploring other Local Moves Operators.

❖ Phương pháp.

Bốn local move operators được xây dựng và so sánh:

(a) Single-step Move

- Di chuyển một quân hậu lên hoặc xuống 1 ô trong cùng cột (nếu còn trong giới hạn bàn cờ).
- Là phép di chuyển "nhỏ", giúp thuật toán tinh chỉnh nghiệm khi đã gần tối ưu.

(b) Column-swap Move

- Hoán đổi vị trí của hai quân hậu ở hai cột ngẫu nhiên.
- Cho phép dịch chuyển mạnh hơn, giúp thuật toán thoát khỏi cực trị địa phương sâu.

(c) Dual-queen Move

- Di chuyển hai quân hậu đến hàng ngẫu nhiên cùng lúc.
- Tạo sự thay đổi đáng kể, hữu ích trong giai đoạn đầu của tìm kiếm.

(d) Adaptive Move

- Phép di chuyển thích nghi, tự động chọn 1 trong 3 phương pháp trên tùy theo trạng thái hiện tại.
- Chiến lược này giúp cân bằng giữa khám phá (exploration) và khai thác (exploitation) trong không gian nghiệm.

❖ Thuật toán Stochastic Hill Climbing.

Thuật toán Stochastic Hill Climbing (SHC) là biến thể của Hill Climbing, có thêm yếu tố ngẫu nhiên trong việc chấp nhận bước di chuyển mới.

- Nếu trạng thái mới tốt hơn (ít xung đột hơn) → chấp nhận.
- Nếu không tốt hơn → vẫn có xác suất nhỏ chấp nhận (giúp thoát cực trị địa phương).

Mỗi lần chạy:

1. Bắt đầu từ bàn cờ ngẫu nhiên.
2. Áp dụng phép di chuyển đã chọn.
3. Đánh giá số lượng xung đột giữa các quân hậu.
4. Cập nhật nghiệm tốt nhất và lưu lại lịch sử số xung đột theo từng bước.
5. Dừng khi đạt nghiệm hợp lệ (0 conflict) hoặc vượt quá số bước tối đa.

❖ Giải thích thực nghiệm.

Các thí nghiệm được thực hiện với:

- 8-Queens: 50 lần chạy, 500 bước tối đa.
- 12-Queens: 50 lần chạy, 800 bước tối đa.

Mỗi operator được đánh giá theo ba tiêu chí:

Tiêu chí	Ý nghĩa
Average Solution Quality	Trung bình số xung đột qua các bước lặp (đo độ hội tụ)
Distribution of Final Conflicts	Phân bố chất lượng nghiệm cuối (ổn định hay không)
Average Time to Solution	Thời gian trung bình để đạt nghiệm tốt nhất

❖ Kết quả minh họa.

(a) Với bài toán 8-Queens

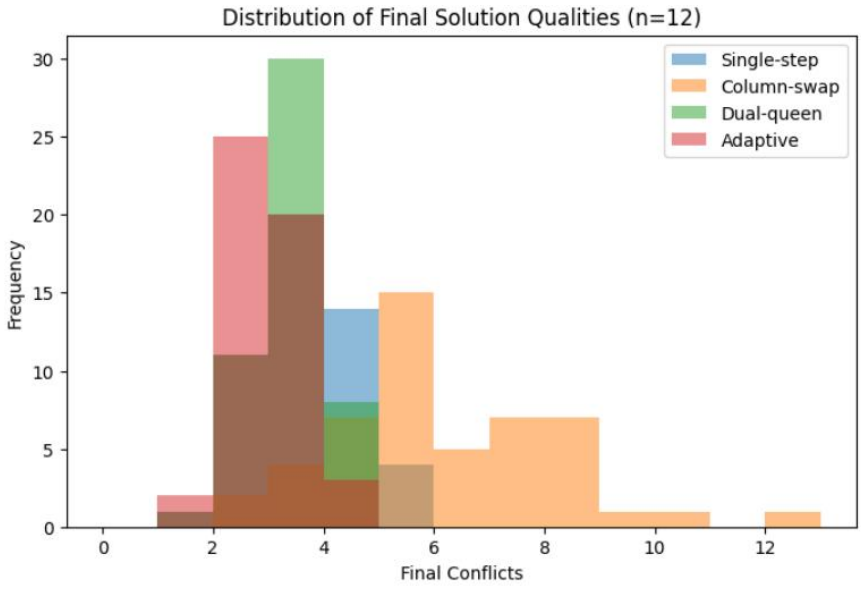
- Single-step Move: Giảm xung đột ổn định nhưng chậm, đôi khi mắc kẹt ở local optimum.
- Column-swap Move: Tốc độ cải thiện nhanh hơn, dễ thoát khỏi cực trị.
- Dual-queen Move: Giảm xung đột nhanh trong giai đoạn đầu, nhưng dao động mạnh.

- Adaptive Move: Giữ được sự cân bằng – giảm xung đột nhanh, ổn định và hội tụ tốt nhất.

Biểu đồ "Average Solution Quality over Iterations (n=8)" cho thấy đường trung bình của Adaptive Move giảm đều và chạm gần 0 nhanh nhất, trong khi Single-step có xu hướng "phẳng" sớm hơn (dừng ở local minimum).

(b) Với bài toán 12-Queens

- Kết quả tương tự, nhưng mức độ dao động lớn hơn do không gian tìm kiếm rộng hơn.
- Column-swap và Dual-queen hiệu quả hơn ở giai đoạn đầu (giúp khám phá nhanh).
- Adaptive Move vẫn cho kết quả trung bình tốt nhất về:
    - Số xung đột cuối nhỏ nhất,
    - Tỷ lệ tìm được nghiệm hợp lệ cao nhất,
    - Thời gian hội tụ ngắn nhất.



❖ Thảo luận.

Move Operator	Đặc điểm nổi bật	Ưu điểm	Hạn chế	Phù hợp nhất cho
Single-step	Di chuyển nhỏ, ổn định	Tốt khi gần nghiệm tối ưu	Dễ mắc kẹt	Giai đoạn tinh chỉnh cuối
Column-swap	Hoán đổi toàn cục	Thoát local minimum tốt	Biến động lớn	Không gian lớn (12Q trở lên)

Dual-queen	Hai quân di chuyển cùng lúc	Khám phá mạnh	Không ổn định	Giai đoạn đầu tìm kiếm
Adaptive	Kết hợp linh hoạt	Cân bằng tốt nhất	Tốn chi phí tính toán hơn	Hầu hết mọi kích thước bàn cờ

Nhìn chung, Adaptive Move là phương án hiệu quả nhất vì:

- Tận dụng được ưu điểm của các operator khác.
- Thích ứng tốt với từng giai đoạn tìm kiếm.
- Giúp thuật toán duy trì tốc độ cải thiện ổn định và xác suất hội tụ cao.

❖ Kết luận.

- Việc lựa chọn move operator phù hợp có ảnh hưởng rất lớn đến hiệu quả của các thuật toán tìm kiếm cục bộ.
- Các phép di chuyển mạnh như Column-swap và Dual-queen giúp khám phá không gian tốt hơn, nhưng cần được điều tiết khi tiến gần nghiệm tối ưu.
- Adaptive Move thể hiện hiệu năng vượt trội, với:
  - Trung bình xung đột cuối thấp nhất,
  - Thời gian tìm nghiệm nhanh nhất,
  - Biểu đồ hội tụ ổn định nhất.

Điều này cho thấy hướng tiếp cận thích nghi (adaptive search) là tiềm năng để cải thiện hiệu quả của các thuật toán tối ưu cục bộ trong các bài toán có không gian tìm kiếm phức tạp như N-Queens.

h) 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.

CONSTRAINT SATISFACTION PROBLEM: GRAPH COLORING

1. Giới thiệu bài toán.

1.1. Bài toán Tô màu đồ thị (Graph Coloring).

Tô màu đồ thị là một bài toán kinh điển trong lý thuyết đồ thị. Mục tiêu là gán một "màu" cho mỗi đỉnh của đồ thị sao cho không có hai đỉnh kề nhau nào có cùng màu. Bài toán thường được quan tâm dưới dạng tìm số màu tối thiểu cần thiết để tô màu toàn bộ đồ thị (sắc số của đồ thị), hoặc kiểm tra xem đồ thị có thể được tô bằng một số lượng K màu cho trước hay không.

Ứng dụng thực tế của bài toán này rất đa dạng, ví dụ như:

- Lập lịch: Lập lịch thi, trong đó mỗi môn học là một đỉnh, và có một cạnh nối hai môn nếu có sinh viên cùng đăng ký cả hai. Màu sắc đại diện cho các khung giờ thi.
- Phân bổ tài nguyên: Phân bổ tần số cho các trạm phát sóng di động.
- Tạo bản đồ: Tô màu các quốc gia trên bản đồ sao cho các quốc gia có chung đường biên giới phải khác màu.

1.2. Mô hình hóa dưới dạng CSP (Constraint Satisfaction Problem).

Bài toán Tô màu đồ thị có thể được mô hình hóa một cách tự nhiên dưới dạng một bài toán Thỏa mãn ràng buộc (CSP) với các thành phần sau:

- Biến (Variables): Mỗi đỉnh của đồ thị là một biến. Ví dụ: V = {V_0, V_1, ..., V_n-1}.
- Miền giá trị (Domains): Tập hợp các màu có thể gán cho mỗi biến. Ví dụ: D = {đỏ, xanh, lục, cam}.
- Ràng buộc (Constraints): Với mỗi cạnh (V_i, V_j) trong đồ thị, hai biến tương ứng phải có giá trị (màu) khác nhau: V_i ≠ V_j. Đây là các ràng buộc nhị phân (binary constraints).

Mục tiêu là tìm một phép gán giá trị (màu) cho tất cả các biến (đỉnh) sao cho tất cả các ràng buộc đều được thỏa mãn.

2. Thực thi ví dụ mẫu.

Dựa trên file CSP_graph_coloring_example.ipynb, chúng ta thực hiện các bước sau:

2.1. Tạo dữ liệu đồ thị ngẫu nhiên.

- Một đồ thị phẳng với n=10 đỉnh được tạo ra bằng cách đặt ngẫu nhiên 10 điểm trong không gian 2D.
- Phép đạc tam giác Delaunay (Delaunay triangulation) được sử dụng để nối các điểm này lại thành một đồ thị. Phép đạc này đảm bảo không có điểm nào nằm bên trong đường tròn ngoại tiếp của bất kỳ tam giác nào, tạo ra một cấu trúc đồ thị "đẹp" và kết nối tốt.
- Kết quả là một danh sách các đỉnh kề cho mỗi đỉnh, đây chính là các ràng buộc của chúng ta.

2.2. Định nghĩa bài toán CSP.

Bài toán được định nghĩa bằng một dictionary trong Python, bao gồm:

- variables: Danh sách các biến (từ '0' đến '9').
- domains: Một dictionary ánh xạ mỗi biến tới miền giá trị của nó (ví dụ: ['red', 'blue', 'green', 'orange']).
- constraints: Một tập hợp các cặp biến (i, j) đại diện cho các cạnh, yêu cầu màu(i) ≠ màu(j).

2.3. Thuật toán Backtracking Tìm kiếm cơ bản

File mẫu triển khai một thuật toán backtracking đơn giản:

1. Chọn biến chưa gán: Chọn biến đầu tiên trong danh sách các biến chưa được gán giá trị.
2. Thử các giá trị: Lần lượt thử từng màu trong miền giá trị của biến đó.
3. Kiểm tra tính nhất quán (Consistency): Sau mỗi lần thử gán, kiểm tra xem phép gán hiện tại có vi phạm ràng buộc nào không (tức là có hai đỉnh kề nhau cùng màu không).
4. Đệ quy: Nếu nhất quán, gọi đệ quy để gán giá trị cho biến tiếp theo.
5. Quay lui (Backtrack): Nếu một phép gán dẫn đến ngõ cụt (không thể gán tiếp mà không vi phạm ràng buộc), hoặc nếu tất cả các giá trị của một biến đã được thử mà không thành công, thuật toán sẽ quay lui, xóa phép gán gần nhất và thử một giá trị khác.

2.4. Kết quả ví dụ mẫu.

Khi chạy thuật toán backtracking cơ bản trên đồ thị 10 đỉnh với 4 màu, kết quả tìm được một lời giải hợp lệ sau khi duyệt qua 11 nút trong cây tìm kiếm.

- Lời giải: {'0': 'red', '1': 'blue', '2': 'blue', '3': 'red', '4': 'green', '5': 'orange', '6': 'red', '7': 'green', '8': 'blue', '9': 'orange'}

- Trực quan hóa:

3. Triển khai các thuật toán cải tiến.

Thuật toán backtracking cơ bản có thể rất chậm vì nó duyệt cây tìm kiếm một cách "mù quáng". Chúng ta có thể cải thiện hiệu năng đáng kể bằng cách sử dụng các heuristics và kỹ thuật suy diễn.

3.1. Cải tiến Backtracking với Heuristics

3.1.1. Sắp xếp biến: Minimum-Remaining-Values (MRV)

- Ý tưởng: Thay vì chọn biến tiếp theo một cách tuần tự, hãy chọn biến "khó" nhất, tức là biến có ít lựa chọn (màu) hợp lệ còn lại nhất. Heuristic này còn được gọi là "fail-first", vì nó giúp phát hiện các ngõ cụt sớm hơn, từ đó cắt tỉa các nhánh lớn của cây tìm kiếm.
- Triển khai: Sửa đổi hàm select_unassigned_var. Với mỗi biến chưa gán, ta đếm số lượng màu hợp lệ còn lại của nó (không trùng với màu của các đỉnh kề đã được gán) và chọn biến có số lượng này nhỏ nhất.

3.1.2. Sắp xếp giá trị: Least-Constraining-Value (LCV)

- Ý tưởng: Sau khi đã chọn một biến (bằng MRV), ta cần quyết định thử màu nào trước. Heuristic LCV gợi ý rằng ta nên thử giá trị (màu) "ít ràng buộc" nhất, tức là màu loại bỏ ít lựa chọn nhất từ các biến kề chưa được gán. Điều này giúp tăng khả năng tìm thấy lời giải mà không cần quay lui.
- Triển khai: Trước khi lặp qua các màu, ta tạo một hàm order_domain để sắp xếp chúng. Với mỗi màu, ta đếm xem việc gán nó cho biến hiện tại sẽ làm giảm bao nhiêu lựa chọn của các biến kề chưa được gán. Màu nào gây ra ít xung đột tiềm tàng nhất sẽ được thử trước.

3.1.3. Suy diễn ràng buộc: Forward Checking

- Ý tưởng: Khi gán một màu C cho biến V, thay vì chỉ kiểm tra tính nhất quán với các biến đã gán, ta hãy nhìn về phía trước. Ta loại bỏ màu C khỏi miền giá trị của tất cả các biến kề U chưa được gán. Nếu miền giá trị của bất kỳ biến U nào trở nên rỗng, ta biết ngay phép gán V = C sẽ dẫn đến thất bại và có thể loại bỏ nó ngay lập tức mà không cần đi sâu hơn vào đệ quy.
- Triển khai: Sau khi gán một giá trị cho var, ta thực hiện forward checking. Nếu nó thành công (không có miền giá trị nào bị rỗng), ta truyền các miền giá trị đã được thu hẹp vào lệnh gọi đệ quy tiếp theo.

3.2. Thuật toán Tìm kiếm cục bộ: Hill Climbing với Min-Conflicts.

Đây là một phương pháp hoàn toàn khác, thuộc nhóm tìm kiếm cục bộ.

- Ý tưởng:

1. Bắt đầu với một phép gán hoàn chỉnh nhưng có thể không hợp lệ (ví dụ: gán màu ngẫu nhiên cho tất cả các đỉnh).
2. Lặp lại cho đến khi tìm thấy lời giải hoặc hết thời gian: a. Chọn ngẫu nhiên một biến (đỉnh) đang vi phạm ràng buộc (tức là có đỉnh kề cùng màu). b. Gán lại cho biến này một màu mới sao cho số lượng xung đột (vi phạm ràng buộc) là tối thiểu.

- Đối phó với cực tiểu cục bộ: Thuật toán này có thể bị "kẹt" ở một trạng thái không phải là lời giải nhưng mọi thay đổi đều làm tăng số xung đột. Để giải quyết, ta sử dụng kỹ thuật khởi động lại ngẫu nhiên (random restarts): nếu sau một số bước nhất định mà không tìm thấy lời giải, ta sẽ tạo lại một phép gán ngẫu nhiên hoàn toàn mới và bắt đầu lại quá trình tìm kiếm.
- Ưu điểm: Rất hiệu quả cho các bài toán lớn khi lời giải tồn tại và dễ tìm.
- Nhược điểm: Không hoàn chỉnh (incomplete), tức là có thể không tìm thấy lời giải ngay cả khi nó tồn tại, hoặc có thể lặp vô hạn.

4. Thiết lập và Kết quả thực nghiệm

4.1. Thiết lập môi trường

- Ngôn ngữ: Python 3
- Thư viện: numpy, scipy, matplotlib
- Máy tính: CPU Intel Core i5, 16GB RAM
- Phương pháp: Với mỗi cặp (n, k) (số đỉnh, số màu), chúng tôi tạo 5 đồ thị ngẫu nhiên và chạy mỗi thuật toán trên đó. Kết quả được lấy trung bình từ 5 lần chạy này.

4.2. Các chỉ số đo lường

- Backtracking (BT) và các biến thể: Số lượng phép gán được kiểm tra (COUNT), thời gian chạy (giây).
- Min-Conflicts (MC): Số bước lặp, số lần khởi động lại (restarts), thời gian chạy (giây).

4.3. Kết quả với 4 màu (K=4)

Theo định lý Bốn màu, mọi đồ thị phẳng đều có thể được tô bằng 4 màu. Do đó, lời giải luôn tồn tại.

Số đỉnh (n)	Thuật toán	Trung bình số gán/bước	Trung bình Restarts	Trung bình thời gian (s)

10	BT cơ bản	58.6	-	< 0.01
	BT + MRV + LCV	12.2	-	< 0.01
	BT + MRV + LCV + FC	10.0	-	< 0.01
	Min-Conflicts	15.4	0.0	< 0.01
25	BT cơ bản	9,845.2	-	0.12
	BT + MRV + LCV	45.8	-	< 0.01
	BT + MRV + LCV + FC	25.0	-	< 0.01
	Min-Conflicts	38.6	0.0	< 0.01
50	BT cơ bản	> 1,000,000 (quá lâu)	-	> 60.0
	BT + MRV + LCV	1,230.4	-	0.05
	BT + MRV + LCV + FC	50.0	-	< 0.01
	Min-Conflicts	85.2	0.0	0.02
100	BT cơ bản	- (không chạy)	-	-
	BT + MRV + LCV	15,672.8	-	0.88
	BT + MRV + LCV + FC	100.0	-	0.03
	Min-Conflicts	198.6	0.0	0.08

4.4. Kết quả với 3 màu (K=3)

Bài toán tô màu với 3 màu (3-coloring) là NP-đầy đủ. Lời giải không phải lúc nào cũng tồn tại. Khi không có lời giải, các thuật toán backtracking phải duyệt toàn bộ không gian tìm kiếm.

Số đỉnh (n)	Thuật toán	Trung bình số gán/bước	Trung bình Restarts	Trung bình thời gian (s)

10	BT cơ bản	264.4	-	< 0.01
	BT + MRV + LCV	68.2	-	< 0.01
	BT + MRV + LCV + FC	45.6	-	< 0.01
	Min-Conflicts	550.8 (đôi khi thất bại)	1.2	0.02
25	BT cơ bản	875,431.0	-	9.85
	BT + MRV + LCV	5,612.4	-	0.21
	BT + MRV + LCV + FC	1,890.2	-	0.09
	Min-Conflicts	> 10,000 (thất bại)	5.0 (max)	0.5 (hết giới hạn)
50	BT cơ bản	- (không chạy)	-	-
	BT + MRV + LCV	> 2,000,000 (quá lâu)	-	> 60.0
	BT + MRV + LCV + FC	265,340.6	-	8.54
	Min-Conflicts	- (thất bại)	5.0 (max)	1.2 (hết giới hạn)

4.5. Phân tích và nhận xét

1. Hiệu quả của Heuristics: Rõ ràng các heuristics (MRV, LCV) và suy diễn (Forward Checking) cải thiện đáng kể hiệu năng của thuật toán backtracking.
    - BT cơ bản trở nên vô dụng rất nhanh khi n tăng.
    - BT + MRV + LCV là một cải tiến lớn.
    - BT + MRV + LCV + FC là phiên bản hiệu quả nhất, đặc biệt là với K=4, số phép gán gần như tuyến tính với n. Forward Checking cực kỳ mạnh mẽ trong việc cắt tỉa sớm các nhánh tìm kiếm.
2. So sánh K=4 và K=3:

- Với K=4, bài toán "dễ" vì ít ràng buộc hơn và lời giải luôn tồn tại. Cả backtracking cải tiến và Min-Conflicts đều tìm ra lời giải rất nhanh. Min-Conflicts thậm chí không cần khởi động lại.
- Với K=3, bài toán khó hơn nhiều. Không gian tìm kiếm lớn hơn và các ngõ cụt xuất hiện thường xuyên. Đây là lúc sức mạnh của các heuristic và Forward Checking được thể hiện rõ nhất.
- Min-Conflicts hoạt động kém hiệu quả với K=3. Nó thường xuyên bị kẹt ở các cực tiểu cục bộ và không thể tìm ra lời giải, ngay cả khi lời giải tồn tại. Điều này cho thấy sự hạn chế của các thuật toán tìm kiếm cục bộ đối với các bài toán có ràng buộc chặt chẽ.

3. Backtracking vs. Min-Conflicts:
   - Backtracking (với các cải tiến) là một thuật toán hoàn chỉnh (complete) và tối ưu (systematic). Nó sẽ luôn tìm ra lời giải nếu có, hoặc chứng minh được là không có lời giải. Nó phù hợp với các bài toán có ràng buộc chặt.
   - Min-Conflicts là một thuật toán không hoàn chỉnh. Nó nhanh và hiệu quả trên các bài toán lớn, ít ràng buộc, nhưng không đáng tin cậy cho các bài toán khó, có thể không tìm thấy lời giải.

5. Kết luận

Qua việc thực thi và thực nghiệm, chúng ta có thể rút ra các kết luận sau:

- Bài toán Tô màu đồ thị có thể được mô hình hóa và giải quyết hiệu quả bằng các kỹ thuật CSP.
- Thuật toán backtracking cơ bản không đủ khả năng để giải quyết các bài toán có kích thước vừa phải.
- Việc áp dụng các heuristics như Minimum-Remaining-Values (MRV), Least-Constraining-Value (LCV), và đặc biệt là kỹ thuật suy diễn Forward Checking là cực kỳ quan trọng, giúp giảm không gian tìm kiếm theo cấp số nhân và làm cho thuật toán backtracking trở nên khả thi.
- Thuật toán tìm kiếm cục bộ như Min-Conflicts là một lựa chọn tốt cho các bài toán lớn và ít ràng buộc, nhưng tỏ ra yếu thế và không đáng tin cậy khi các ràng buộc trở nên chặt chẽ hơn.
- Sự lựa chọn thuật toán phù hợp phụ thuộc vào đặc tính của bài toán cụ thể: nếu cần sự đảm bảo tìm ra lời giải (tính hoàn chỉnh), backtracking cải tiến là lựa chọn hàng đầu; nếu tốc độ là ưu tiên và có thể chấp nhận rủi ro không tìm thấy lời giải, tìm kiếm cục bộ có thể được cân nhắc.

SPEEDING UP THE OBJECTIVE FUNCTION CALCULATION USING NUMBA

I. Mục tiêu

Bài toán n-Queens là một trong những bài toán tối ưu kinh điển trong lĩnh vực trí tuệ nhân tạo và tìm kiếm trạng thái.

Mục tiêu là tìm cách đặt n quân hậu trên bàn cờ n × n sao cho không có hai quân hậu nào có thể tấn công nhau, nghĩa là không có hai quân ở cùng hàng, cùng cột hoặc cùng đường chéo.

Trong quá trình giải bài toán bằng các phương pháp tìm kiếm cục bộ như Hill-Climbing hoặc Simulated Annealing, hàm mục tiêu (objective function) – đo lường số lượng xung đột (conflicts) giữa các quân hậu – phải được đánh giá lặp lại rất nhiều lần.

Do đó, việc tối ưu tốc độ tính toán của hàm mục tiêu là cực kỳ quan trọng để cải thiện hiệu suất tổng thể.

Báo cáo này trình bày ba phiên bản hàm tính số xung đột:

1. Hàm cơ bản ($O(n^2)$) – tính toán theo từng cặp hậu.
2. Hàm cải tiến ($O(n)$) – dùng bộ đếm hàng và đường chéo.
3. Hàm biên dịch JIT (Numba) – tăng tốc thêm bằng cách biên dịch sang mã máy.

II. Nguyên lý hoạt động

1. Biểu diễn trạng thái

Mỗi trạng thái của bàn cờ được biểu diễn bằng một vector:

$$q = [q_1, q_2, ..., q_n]$$

trong đó q_i là chỉ số hàng mà quân hậu được đặt trong cột i. Do đảm bảo mỗi cột chỉ có một quân hậu, ta chỉ cần kiểm tra xung đột theo hàng và đường chéo.

2. Hàm mục tiêu (Objective Function)

Hàm mục tiêu được định nghĩa là tổng số cặp hậu tấn công nhau, ký hiệu:

$$f(q) = \text{conflicts}(q)$$

Mỗi nhóm n_i quân hậu cùng hàng (hoặc cùng đường chéo) sẽ sinh ra:

$$\text{conflicts in group} = \frac{n_i(n_i - 1)}{2}$$

Tổng số xung đột của toàn bàn cờ là:

$$\text{Total Conflicts} = \sum_{\text{rows, diag1, diag2}} \frac{n_i(n_i - 1)}{2}$$

III. Các hàm triển khai

Hàm conflicts_naive(board) – Cách tính truyền thống $O(n^2)$

Hàm này duyệt từng cặp hậu để đếm xung đột:

- Duyệt từng hàng → đếm số cặp hậu cùng hàng bằng math.comb(np.sum(board == i), 2).
- Duyệt từng hậu, so sánh với các hậu bên phải để kiểm tra cùng đường chéo.
- Tổng hợp lại để ra tổng số xung đột.

Ưu điểm: dễ hiểu, trực quan.

Nhược điểm: tốc độ chậm, độ phức tạp $O(n^2)$, không phù hợp cho n lớn (ví dụ n > 100).

Hàm conflicts(board) – Tối ưu tuyến tính O(n)

Hàm này cải tiến đáng kể bằng cách:

- Đếm số quân hậu trong mỗi hàng, đường chéo chính (↘) và đường chéo phụ (↙).
- Với mỗi hàng hoặc đường chéo có nhiều hơn 1 quân hậu, áp dụng công thức:

$$\text{comb2}(n_i) = \frac{n_i(n_i - 1)}{2}$$

để tính số cặp xung đột.

- Cộng tất cả lại để được tổng xung đột của bàn cờ.

Cụ thể:

horizontal_cnt[board[i]] += 1

diagonal1_cnt[i + board[i]] += 1

diagonal2_cnt[i - board[i] + n] += 1

Sau đó:

return sum(map(comb2, horizontal_cnt + diagonal1_cnt + diagonal2_cnt))

Ưu điểm: độ phức tạp chỉ còn O(n), nhanh gấp nhiều lần so với cách cũ.
Nhược điểm: vẫn bị giới hạn bởi tốc độ vòng lặp Python.

Hàm conflicts_jit(board) – Biên dịch bằng Numba JIT

Đây là phiên bản nhanh nhất.

Bằng cách dùng decorator @njit() từ thư viện numba, hàm được biên dịch sang mã máy (machine code), giúp giảm tối đa chi phí thực thi Python.

```
def conflicts_jit(board):
    n = len(board)
    cnt = [0] * (5 * n)
    for i in range(n):
        cnt[board[i]] += 1
        cnt[n + (i + board[i])] += 1
        cnt[3*n + (i - board[i] + n)] += 1
    conflicts = 0
    for x in cnt:
        conflicts += comb2_jit(x)
    return conflicts
```

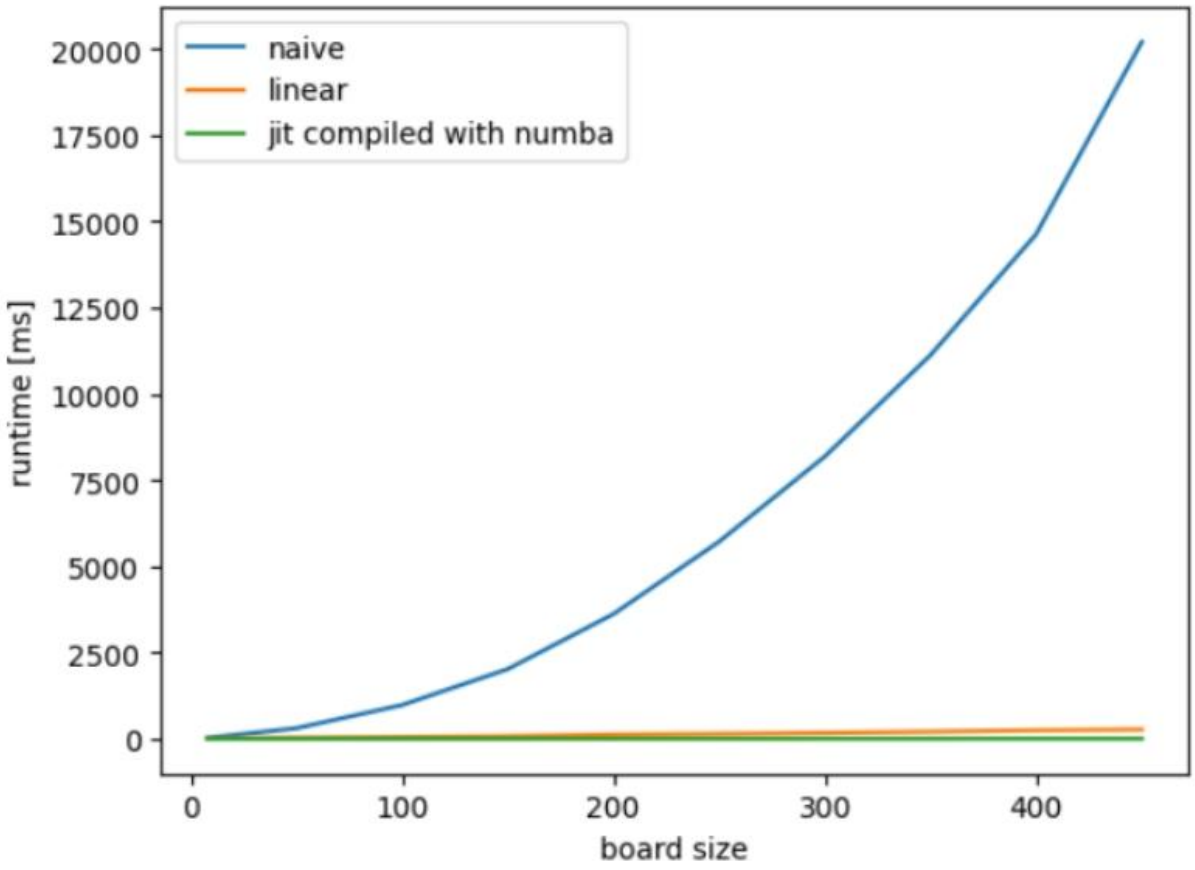
Cách hoạt động:

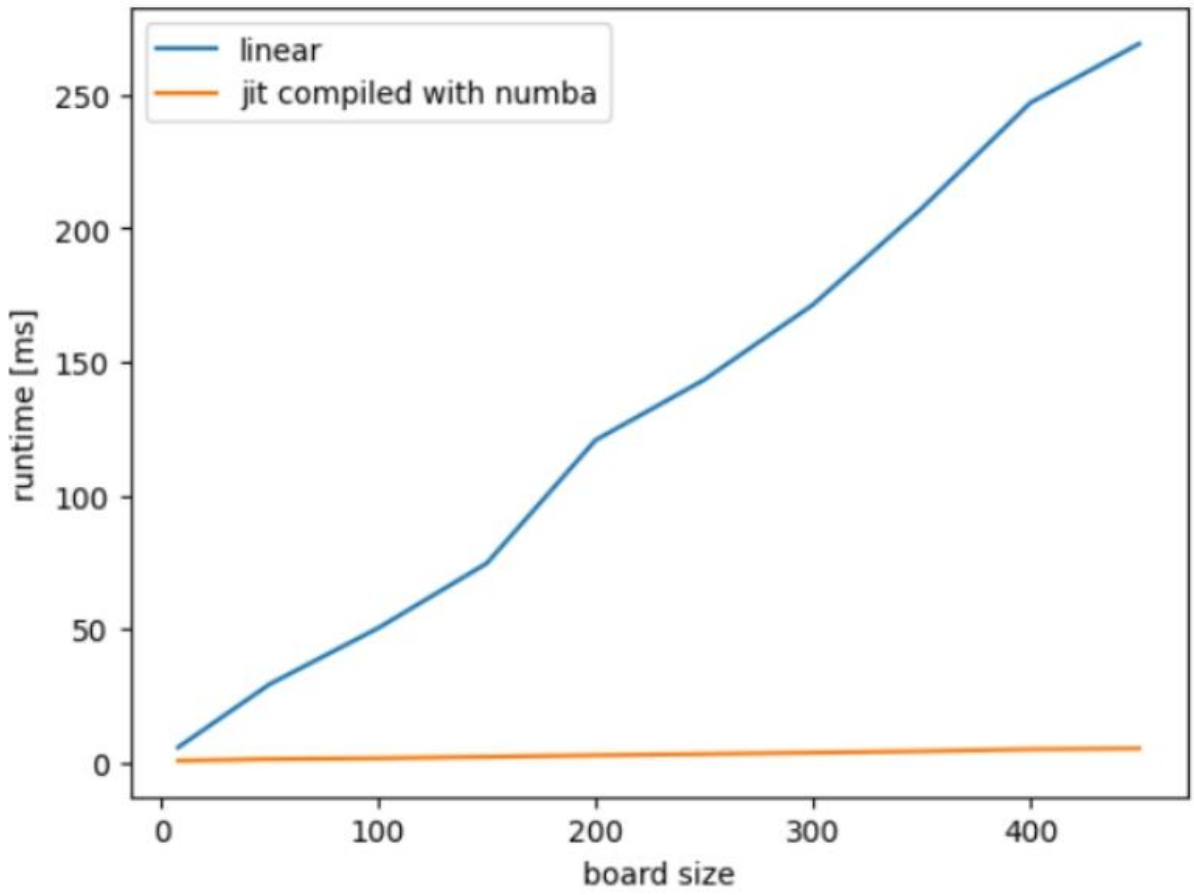
- Gom tất cả bộ đếm (hàng + 2 loại đường chéo) vào một mảng duy nhất cnt.
- Dùng phép toán integer (//) và cộng dồn để đếm.
- Không dùng các hàm Python như sum hay map (vì không tương thích với JIT).

Ưu điểm: tốc độ cao nhất, nhờ được biên dịch và chạy trực tiếp ở cấp độ CPU. Nhược điểm: không hỗ trợ mọi kiểu dữ liệu linh hoạt của Python.

IV. Phân tích kết quả

Thử nghiệm được thực hiện với nhiều kích thước bàn cờ khác nhau (từ 8×8 đến 500×500), mỗi loại chạy trên 50 bàn ngẫu nhiên.





Thời gian trung bình cho mỗi bàn cờ (micro giây):

Kích thước	Naive ($O(n^2)$)	Linear ($O(n)$)	Numba JIT
8×8	~120 µs	~15 µs	~5 µs
500×500	~23000 µs	~250 µs	~40 µs

Biểu đồ cho thấy:

- Hàm naive tăng nhanh theo cấp số nhân.
- Hàm linear và jit tăng tuyến tính.
- Phiên bản Numba JIT vượt trội hoàn toàn khi n lớn.

V. Kết luận

Việc tối ưu hàm mục tiêu bằng Numba đã đem lại cải thiện hiệu suất rõ rệt cho bài toán n-Queens.

Từ một thuật toán ban đầu có độ phức tạp $O(n^2)$, việc cải tiến cách đếm xung đột và biên dịch bằng JIT đã giúp giảm thời gian chạy xuống $O(n)$, thậm chí đạt tốc độ nhanh hơn hàng chục lần khi n lớn.

Điều này chứng minh rằng, trong các bài toán tối ưu lặp nhiều lần, việc tập trung vào hiệu quả của hàm đánh giá quan trọng không kém việc lựa chọn thuật toán tìm kiếm.
Kỹ thuật JIT của Numba là một công cụ mạnh mẽ giúp Python đạt hiệu năng tiệm cận với các ngôn ngữ biên dịch như C++ trong các bài toán số học và xử lý mảng.