| **BỘ GIÁO DỤC VÀ ĐÀO TẠO**  **TRƯỜNG ĐẠI HỌC SÀI GÒN**  **KHOA CÔNG NGHỆ THÔNG TIN**  ---o0o---    **BÁO CÁO BÀI TẬP**  **TUẦN 4**  **Giáo viên hướng dẫn:** Đỗ Như Tài    ***Ngày 5, tháng 10 năm 2025*** |
| --- |

**MỤC LỤC:**

[**Bảng Phân Công Công Việc TTNT(NC) 3**](#_goxgpuwj1q2)

[**Lab03: n-Queens 4**](#_kv9dq49i6hsf)

[Task 1: Steepest-ascend Hill Climbing Search 4](#_dspxxtegejzj)

[Task 2: Stochastic Hill Climbing Search 5](#_76ag6v7v7yps)

[Task 3: First-Choice Hill Climbing Search 7](#_sqtxmzobhhz9)

[Task 4: Hill Climbing Search with Random Restarts 8](#_wlq01wwgz3jn)

[Task 5: Simulated Annealing 9](#_sdj24im8f257)

[Task 6: Algorithm Behavior Analysis 12](#_1lhk3pmz6rp)

[**Lab03: Traveling Salesman Problem 14**](#_kk7sr2llaetl)

[Steepest-ascend Hill Climbing Search 14](#_k575b6y7bd94)

[Steepest-ascend Hill Climbing Search with Random Restarts 16](#_a0avkwbe896f)

[Stochastic Hill Climbing 18](#_m789o56ly7e)

[First-choice Hill Climbing 20](#_4hsiu7n8wj44)

[Simulated Annealing 22](#_iepkc3oz681g)

[Compare Performance 24](#_cuj95v9ynxh3)

# 

# 

# Bảng Phân Công Công Việc TTNT(NC)

**- Thông tin thành viên nhóm.**

| **STT** | **Họ Tên** | **MSSV** | **Link GitHub** |
| --- | --- | --- | --- |
| 1 | Nguyễn Hồng Thiên Bảo(\*) | 3122410020 |  |
| 2 | Quách Huỳnh Gia Bảo | 3122410025 |  |
| 3 | Trần Lê Công Danh | 3122410048 |  |
| 4 | Văn Cẩm Hào | 3122410101 |  |

**(\*) lab02 - search (tt)**

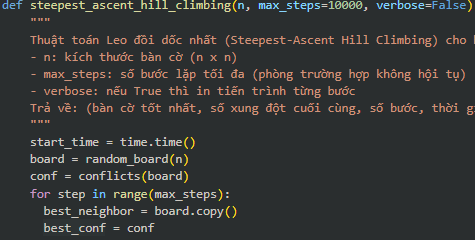
| **STT** | **Họ Tên** | **Tasks** | |
| --- | --- | --- | --- |
| 1 | Quách Huỳnh Gia Bảo | **Traveling Salesman Problem** | Steepest-ascend Hill Climbing Search |
| Steepest-ascend Hill Climbing Search with Random Restarts |
| 2 | Văn Cẩm Hào | **n-Queens** | Task 1: Steepest-ascend Hill Climbing Search |
| Task 2: Stochastic Hill Climbing 1 |
| Task 3: Stochastic Hill Climbing 2 |
| 3 | Nguyễn Hồng Thiên Bảo | **n-Queens** | Task 4: Hill Climbing Search with Random Restarts |
| Task 5: Simulated Annealing |
| Task 6: Algorithm Behavior Analysis |
| 4 | Trần Lê Công Danh | **Traveling Salesman Problem** | Stochastic Hill Climbing |
| First-choice Hill Climbing |
| Simulated Annealing |
| Compare Performance |

# Lab03: n-Queens

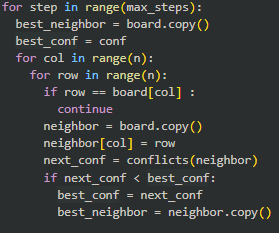
## Task 1: Steepest-ascend Hill Climbing Search

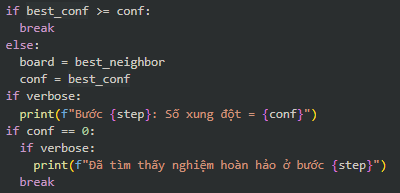
Thuật toán Steepest-Ascent Hill Climbing (leo đồi dốc nhất) là phương pháp tìm kiếm cục bộ, trong đó ở mỗi bước ta xem xét tất cả các trạng thái lân cận của trạng thái hiện tại và chọn nước đi mang lại cải thiện lớn nhất (tức là có ít xung đột nhất trong bài toán n-Queens).

**Bước 1: Khởi tạo các tham số cần thiết cho thuật toán**

* n: kích thước bàn cờ (n × n).
* max\_steps: số bước lặp tối đa (phòng trường hợp thuật toán không hội tụ).
* verbose: nếu True, in ra tiến trình từng bước.
* start\_time / end\_time: thời gian bắt đầu và kết thúc thuật toán để đo hiệu suất.
* board: mảng lưu vị trí của n quân hậu, khởi tạo ngẫu nhiên bằng random\_board(n).
* conf: số xung đột hiện tại trên bàn cờ (mỗi cặp quân hậu tấn công nhau làm tăng 1 xung đột).
* best\_neighbor: lưu trạng thái có số xung đột thấp nhất tạm thời.
* best\_conf: lưu số xung đột thấp nhất tìm được trong vòng lặp hiện tại.

**Bước 2: Duyệt qua tất cả các trạng thái lân cận (neighbors)**

* Thuật toán duyệt từng cột (col) và thử di chuyển quân hậu sang mọi hàng (row) khác có thể, tạo ra các trạng thái lân cận (neighbor).
* Với mỗi neighbor, tính số xung đột bằng conflicts(neighbor).
* Nếu số xung đột của neighbor (next\_conf) nhỏ hơn số xung đột tốt nhất hiện tại (best\_conf), thì cập nhật:
  + best\_conf = next\_conf
  + best\_neighbor = neighbor.copy()
* Sau khi duyệt hết, best\_neighbor là trạng thái tốt nhất trong toàn bộ không gian lân cận, tương ứng với hướng “leo đồi dốc nhất”.

**Bước 3: Cập nhật trạng thái và kiểm tra điều kiện dừng** 

* Nếu không có trạng thái nào tốt hơn (best\_conf >= conf), thuật toán coi như đã đạt cực trị địa phương (local optimum) và dừng lại.
* Nếu có cải thiện, cập nhật rồi tiếp tục vòng lặp:
* board = best\_neighbor
* conf = best\_conf
* Khi conf == 0, nghĩa là tất cả quân hậu đều an toàn, thuật toán kết thúc thành công.
* Nếu verbose = True, chương trình in số bước và số xung đột của mỗi lần lặp để theo dõi quá trình hội tụ.

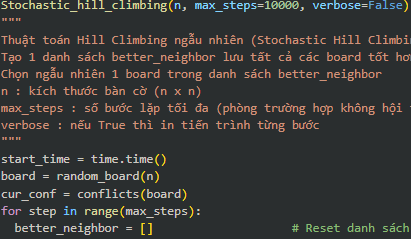
**Bước 4: Kết thúc và trả về kết quả**

* Khi kết thúc do đạt nghiệm hoàn hảo, mắc kẹt tại local optimum, hoặc hết giới hạn max\_steps, thuật toán:
* Tính thời gian chạy tổng cộng: end\_time = time.time() - start\_time
* Trả về bộ giá trị:
  + board: cấu hình bàn cờ cuối cùng
  + conf: số xung đột cuối cùng
  + step + 1: tổng số bước đã thực hiện
  + end\_time: thời gian chạy

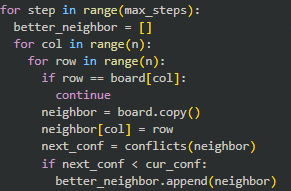
## Task 2: Stochastic Hill Climbing Search

Thuật toán Stochastic Hill Climbing (Leo đồi ngẫu nhiên) là một biến thể của phương pháp tìm kiếm leo đồi, trong đó thuật toán không chọn trạng thái tốt nhất trong tất cả các trạng thái lân cận, mà chọn ngẫu nhiên một trạng thái tốt hơn so với trạng thái hiện tại. Cách tiếp cận này giúp tránh việc rơi vào cực trị địa phương (local optimum) sớm, đồng thời vẫn duy trì hướng cải thiện dần của lời giải trong bài toán N-Queens.

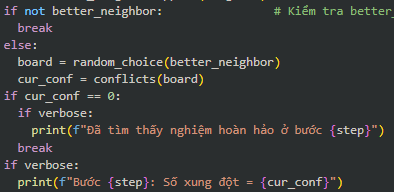
**Bước 1: Khởi tạo các tham số cần thiết cho thuật toán**

* n: kích thước bàn cờ (n × n).
* max\_steps: số bước lặp tối đa (phòng trường hợp thuật toán không hội tụ).
* verbose: nếu True, in ra tiến trình từng bước.
* start\_time / end\_time: thời gian bắt đầu và kết thúc thuật toán để đo hiệu suất.
* board: mảng lưu vị trí của n quân hậu, được khởi tạo ngẫu nhiên thông qua random\_board(n).
* cur\_conf: số xung đột hiện tại của bàn cờ (mỗi cặp quân hậu tấn công nhau làm tăng 1 xung đột).
* better\_neighbor: danh sách chứa tất cả các trạng thái lân cận tốt hơn trạng thái hiện tại (tức là có ít xung đột hơn).

**Bước 2: Duyệt qua các trạng thái lân cận (neighbors)**

* Với mỗi cột (col) trên bàn cờ, thuật toán thử di chuyển quân hậu đến tất cả các hàng (row) khác có thể để tạo ra các trạng thái lân cận (neighbor).
* Với mỗi neighbor, tính số xung đột mới bằng hàm conflicts(neighbor).
* Nếu số xung đột mới (next\_conf) nhỏ hơn số xung đột hiện tại (cur\_conf), trạng thái đó được xem là tốt hơn và được thêm vào danh sách better\_neighbor.
* Sau khi duyệt hết toàn bộ không gian lân cận, danh sách better\_neighbor sẽ chứa tất cả các hướng di chuyển có thể giúp cải thiện lời giải.

**Bước 3: Chọn ngẫu nhiên và cập nhật trạng thái**

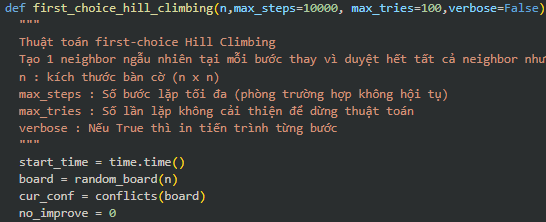
* Nếu danh sách better\_neighbor rỗng, tức là không tồn tại trạng thái nào tốt hơn, thuật toán coi như đã đạt cực trị địa phương (local optimum) và dừng lại.
* Ngược lại, thuật toán chọn ngẫu nhiên một trạng thái trong danh sách better\_neighbor bằng random.choice() để di chuyển đến.
* Cập nhật:
  + board = neighbor (trạng thái mới được chọn ngẫu nhiên)
  + cur\_conf = conflicts(board) (số xung đột của trạng thái mới)
* Nếu cur\_conf == 0, tức là không còn xung đột nào giữa các quân hậu, thuật toán kết thúc với nghiệm hoàn hảo.
* Nếu verbose = True, chương trình in ra thông tin ở mỗi bước, bao gồm số bước hiện tại và số xung đột tương ứng, để theo dõi tiến trình hội tụ.

**Bước 4: Kết thúc và trả về kết quả**

* Khi vòng lặp kết thúc do đạt nghiệm hoàn hảo, mắc kẹt tại local optimum, hoặc hết giới hạn max\_steps, thuật toán:
  + Tính thời gian chạy tổng cộng:
    - end\_time = time.time() - start\_time
  + Trả về bộ giá trị:
    - board: cấu hình bàn cờ cuối cùng
    - cur\_conf: số xung đột cuối cùng
    - step + 1: tổng số bước đã thực hiện
    - end\_time: thời gian chạy thực tế

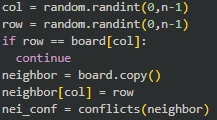
## Task 3: First-Choice Hill Climbing Search

Thuật toán First-Choice Hill Climbing (Leo đồi chọn ngẫu nhiên đầu tiên) là một biến thể khác của phương pháp leo đồi. Thay vì duyệt toàn bộ các trạng thái lân cận (neighbors) như trong Steepest-Ascent hay chọn ngẫu nhiên từ tất cả neighbor tốt hơn như trong Stochastic Hill Climbing, thuật toán này chỉ chọn ngẫu nhiên một neighbor tại mỗi bước và cập nhật ngay nếu neighbor đó tốt hơn. Cách tiếp cận này giúp giảm chi phí tính toán (vì không cần duyệt hết mọi khả năng) và vẫn duy trì khả năng cải thiện lời giải dần dần trong bài toán N-Queens.

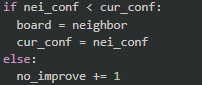
**Bước 1: Khởi tạo các tham số cần thiết cho thuật toán** ****

* n: Kích thước bàn cờ (n × n).
* max\_steps: Số bước lặp tối đa (phòng trường hợp thuật toán không hội tụ).
* max\_tries: Số lần lặp liên tiếp mà không cải thiện được lời giải trước khi dừng thuật toán.
* verbose: Nếu đặt là True, thuật toán sẽ in tiến trình tìm kiếm từng bước.
* start\_time / end\_time: Thời điểm bắt đầu và kết thúc để đo thời gian chạy.
* board: Mảng lưu vị trí của n quân hậu, được khởi tạo ngẫu nhiên thông qua random\_board(n).
* cur\_conf: Số xung đột hiện tại của bàn cờ (số cặp quân hậu đang tấn công nhau).
* no\_improve: Biến đếm số lần liên tiếp không cải thiện được trạng thái hiện tại.

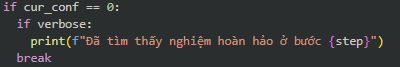
**Bước 2: Sinh và đánh giá trạng thái lân cận ngẫu nhiên**

* Ở mỗi bước lặp, thuật toán chọn ngẫu nhiên một cột (col) và một hàng (row) mới trên bàn cờ.
* Nếu hàng vừa chọn trùng với vị trí hiện tại của quân hậu ở cột đó, thuật toán bỏ qua và chọn lại (vì không thay đổi trạng thái).
* Tạo neighbor — trạng thái mới bằng cách di chuyển quân hậu ở cột col sang hàng row vừa chọn.
* Tính số xung đột nei\_conf của neighbor bằng hàm conflicts(neighbor).

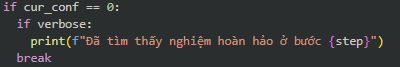
**Bước 3: So sánh và cập nhật trạng thái**

* Nếu nei\_conf < cur\_conf, tức là trạng thái mới tốt hơn:
* Cập nhật bàn cờ hiện tại: board = neighbor
* Cập nhật số xung đột: cur\_conf = nei\_conf
* Đặt lại bộ đếm no\_improve = 0 (vì đã có cải thiện).
* Ngược lại, nếu neighbor không tốt hơn, tăng bộ đếm no\_improve += 1.

**Bước 4: Điều kiện dừng thuật toán**

* Thuật toán dừng lại khi một trong các điều kiện sau được thỏa mãn:
  + cur\_conf == 0: Không còn xung đột nào giữa các quân hậu tức là đã tìm thấy nghiệm hoàn hảo.
  + no\_improve > max\_tries: Sau nhiều lần thử ngẫu nhiên nhưng không cải thiện được nghĩa là thuật toán mắc kẹt ở cực trị địa phương.
  + Đạt đến max\_steps: Hết số bước lặp cho phép mà vẫn chưa tìm được nghiệm tốt hơn.

**Bước 5: Kết thúc và trả về kết quả**

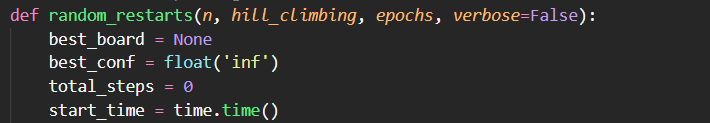
* Ghi nhận thời gian chạy thực tế:
  + end\_time = time.time() - start\_time
* Trả về bộ giá trị kết quả:
  + board: Cấu hình bàn cờ cuối cùng
  + cur\_conf: Số xung đột cuối cùng
  + step: Số bước đã thực hiện
  + end\_time: Tổng thời gian chạy của thuật toán
* Nếu verbose=True, chương trình sẽ in ra:
  + Số bước hiện tại
  + Số xung đột tương ứng
  + Thông báo khi tìm được nghiệm hoàn hảo.

## Task 4: Hill Climbing Search with Random Restarts

Hill climbing thường kẹt ở cực trị cục bộ, vì vậy để tìm giải pháp tốt hơn (có thể là tối ưu toàn cục), thuật toán cần được khởi động lại nhiều lần (restart) từ các bảng ngẫu nhiên mới, tối đa 100 lần, và lưu lại kết quả tốt nhất trong tất cả các lần chạy.

**Bước 1: Khởi tạo các tham số cần thiết cho thuật toán**

* + best\_board : lưu trạng thái bảng tốt nhất tìm được.
  + best\_conf: lưu số xung đột ít nhất.
  + total\_steps: đếm tổng số bước thực hiện.
  + start\_time: lưu thời gian bắt đầu



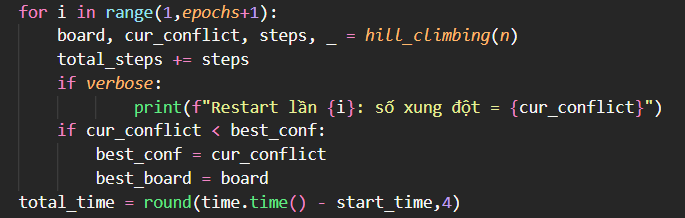
Trong đó:

* + n: kích thước bàn cờ.
  + Hill\_climbing: thuật toán thực thi.
  + Epochs: số lần mốt chạy restart.

Chú thích:

* + float(‘inf’): giá trị vô cực dương.

**Bước 2: Chạy vòng lặp Restart để đạt được giá trị tốt nhất trong n lần chạy**



- Chúng ta sẽ chạy số vòng lặp restart để thuật toán có thể tạo ra nhiều lần chạy khác nhau với các bàn cờ khác nhau và đạt được giá trị tốt nhất.

- Trong quá trình chạy vòng lặp ta sẽ gọi tên hàm thực thi thuật toán và lấy các giá trị: board, cur\_conflicts, steps để có thể so sánh lưu kết quả tốt nhất.

- Sau đó tính toán thời gian thực thi hàm random\_restarts().

**Bước 3: Kết thúc và trả về kết quả**



## Task 5: Simulated Annealing

Simulated annealing là một dạng của **stochastic hill climbing** giúp tránh cực trị cục bộ bằng cách **cho phép di chuyển xuống** với xác suất tỉ lệ thuận với nhiệt độ. Nhiệt độ được giảm dần trong mỗi vòng lặp theo một **lịch trình annealing**. Bạn cần thử nghiệm với lịch trình annealing (tìm trên Google để tìm hướng dẫn).

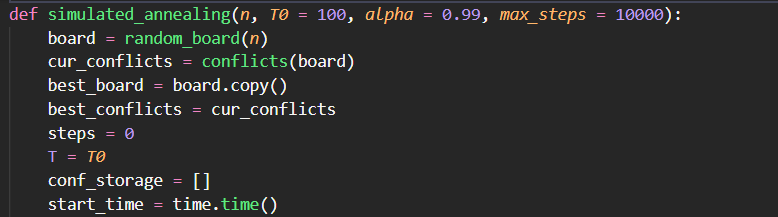
1. Triển khai **simulated annealing** cho bài toán n-Queens.
2. Tạo **trực quan hóa quá trình tìm kiếm** (biểu đồ đường thể hiện số xung đột thay đổi khi thuật toán tiến triển).
3. Sử dụng trực quan hóa này để thử nghiệm với các lựa chọn khác nhau của lịch trình annealing và thảo luận những gì bạn học được.

**Bước 1: Khởi tạo các tham số cần thiết cho thuật toán**

* + board : lưu bàn cờ được tạo.
  + cur\_conflicts: lưu số xung đột hiện tại.
  + best\_board: lưu bàn cờ tốt nhất.
  + best\_conflicts: lưu số xung đột tốt nhất.
  + T: giá trị nhiệt độ.
  + conf\_storage: lưu các giá trị xung đột qua những lần lặp.
  + start\_time: lưu thời gian bắt đầu

Trong đó

* + n: là kích thước bàn cờ.
  + T0: nhiệt độ ban đầu.
  + alpha: % tốc độ giảm nhiệt.
  + max\_steps: số bước tối đa có thể dung.

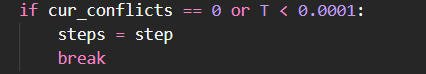


**Bước 2: Chạy vòng lặp cho đến khi đạt được xung đột bằng 0 hoặc hạ nhiệt độ thấp nhất.**

- Chạy vòng lặp.



- Điều kiện dừng vòng lặp: chỉ khi xung đột bằng 0 hoặc nhiệt độ đạt giá trị nhỏ hơn giá trị đặt ra thì kết thúc vòng lặp.

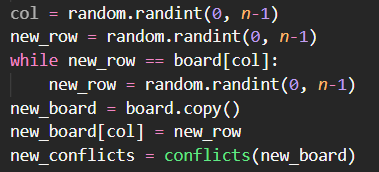


- Để tránh tình trạng rơi vào giá trị tốt nhất ở cục bộ chứ không phải toàn bộ thì cần tạo ngẫu nhiên nơi cần đặt để đạt được hiệu quả.

- Nhưng nếu giá trị cần đặt có giá trị bằng giá trị tạo mới thì sẽ tạo mới giá trị cần đặt đến khi không trùng.

- Sau đó tạo ra bàn cờ mới dựa khi cập nhật giá trị mới vào trong bàn cờ.

- Tính lại giá trị xung đột trong bàn cờ.



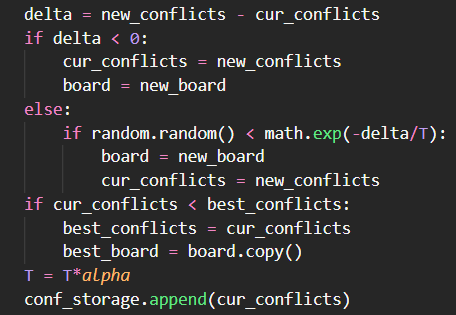
- Tính delta độ chênh lệch số xung đột. Nếu delta < 0 thì tức là giá trị xung đột đã giảm.

- Dựa vào delta để phán đoán xem coi step đó có đạt được giải pháp tốt không nếu có thì cập nhật lại cur\_conflicts và board. Còn nếu không thì sẽ dựa vào xác xuát tính tại giải pháp đó so sánh vs giá trị ngẫu nhiên trong khoảng 0 đến 1 để quyết định chấp nhập giải pháp xấu hiện tại để tìm ra giá trị tối ưu nhất trên toàn bài.

- Nếu số xung đột hiện tại ít hơn so với giá trị tốt nhất đang lưu trữ thì cập nhật lại giá trị tốt nhất cho số xung đột và bàn cờ.

- Cuối cùng là hạ nhiệt độ. Lấy nhiệt độ hiện tại nhân với alpha

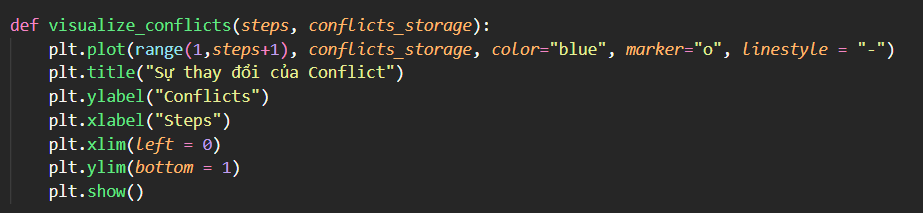
- Lưu giá trị xung đột tại bước đó vào danh sách chứa các giá trị xung đột.



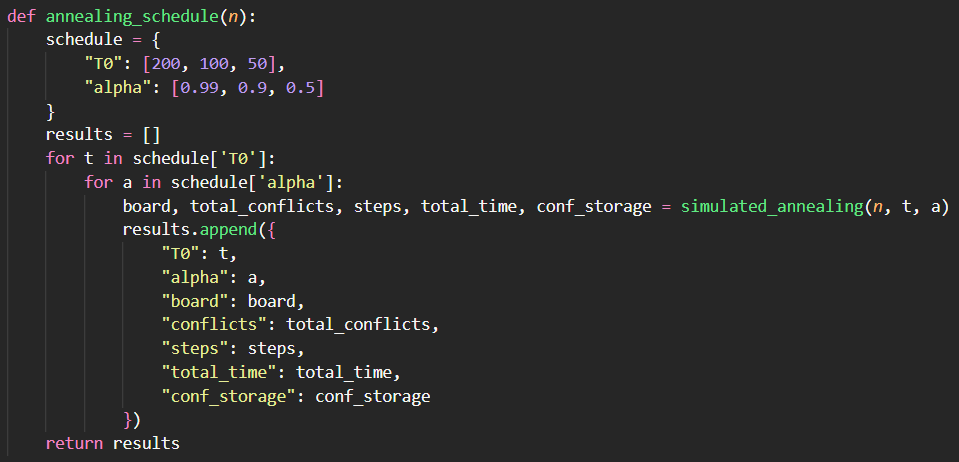
**Bước 3: Kết thúc và trả về kết quả**



**Bước 4: trực quan hóa giữa giá trị steps và conflicts.**



**Bước 6: Thử nghiệm thuật toán với các giá trị khác nhau.**



- Dựa vào các giá trí khác nhau sẽ ảnh hưởng đến việc giảm xung đột hiệu quả hay không và tốc độ chay thuật toán như nào.

- Nếu nhiệt độ càng cao - tốc độ hạ nhiệt thấp: thì tăng thời gian giải bài toán lên nhưng sẽ giúp tăng khả năng tìm ra giải pháp tốt nhất toàn bài sẽ cao và hiệu quả hơn.

- Nếu nhiệt độ cao - tốc độ hạ nhiệt cao: thì bài toán khó có thể không tìm ra giải pháp tốt nhất

- Nếu nhiệt độ thấp - tốc độ cao: bài toán sẽ được giải quyết một cách quá nhanh không kịp tìm thấy giải pháp tối ưu cục bộ luôn chứ đừng nói đến toàn cục.

## Task 6: Algorithm Behavior Analysis

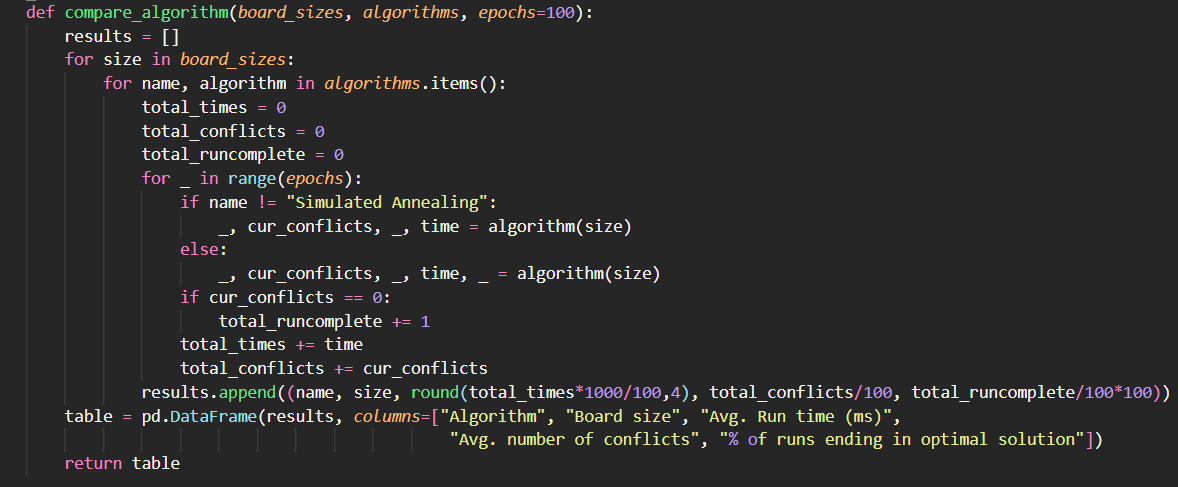
**\* So Sánh:**

- So sánh các thuật toán bằng thời gian chạy và giá trị hàm mục tiêu. Sử dụng các bảng có kích thước 4 và 8 để khảo sát hiệu suất của các thuật toán khác nhau. Đảm bảo chạy mỗi thuật toán cho mỗi kích thước bảng nhiều lần (ít nhất 100 lần) với các bảng khởi đầu khác nhau và báo cáo giá trị trung bình.

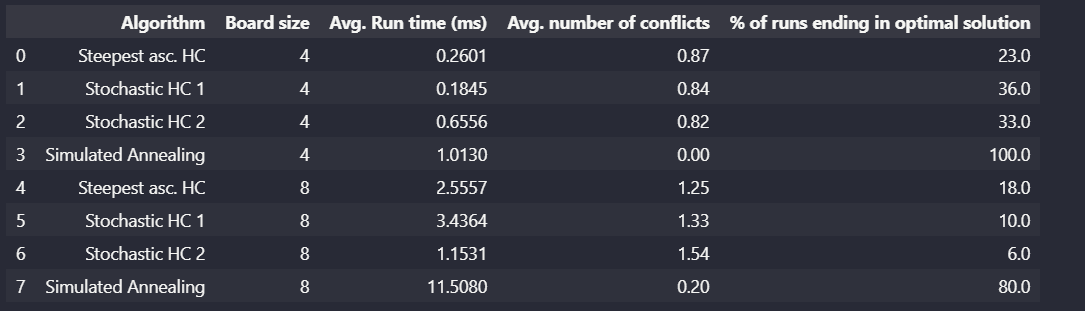
- Các cột giá trị cần tìm để so sánh:

* + Avg. Run time (ms): Thời gian chạy trung bình
  + Avg. number of conflicts: Số xung đột trung bình.
  + % of runs ending in optimal solution: Phần trăm tỉ lệ chạy ra kết quả tối ưu.

- Code:

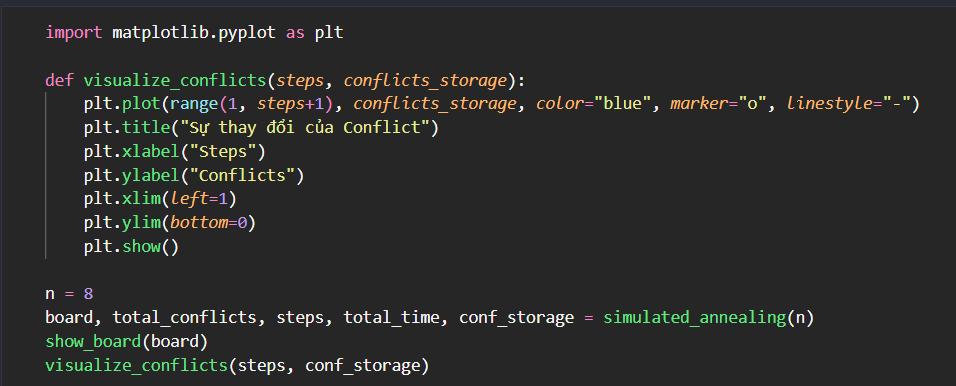


- Kết quả so sánh:

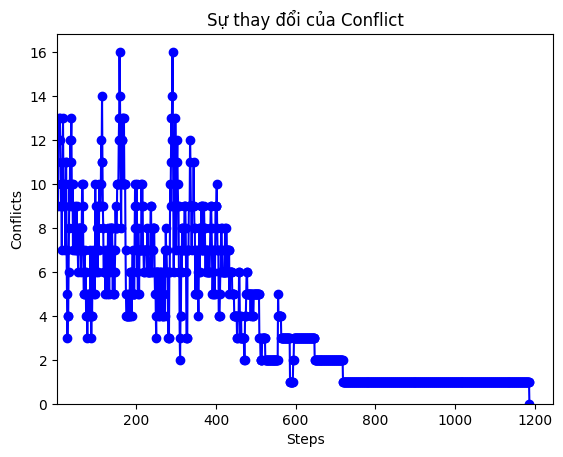


**\* Hội tụ của thuật toán:**

- Code:



- Kết quả:



# Lab03: Traveling Salesman Problem

## Steepest-ascend Hill Climbing Search

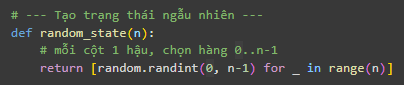
Thuật toán **Steepest-Ascent Hill Climbing** là một phương pháp tìm kiếm cục bộ (local search), thường được sử dụng để giải các bài toán tối ưu. Ý tưởng chính là luôn di chuyển đến trạng thái lân cận tốt nhất (ít "conflict" nhất trong bài toán N-Queens) cho đến khi không còn cải thiện được nữa. Nếu không tìm được trạng thái tốt hơn, thuật toán dừng lại tại cực trị địa phương (local optimum).

Trong bài toán **N quân hậu (N-Queens)**, mục tiêu là đặt N quân hậu trên bàn cờ NxN sao cho không có cặp hậu nào tấn công nhau. Mỗi quân hậu được đặt trên một cột khác nhau, và vị trí hàng trong mỗi cột được thay đổi để tìm ra cấu hình ít xung đột nhất.

**Các bước làm:**

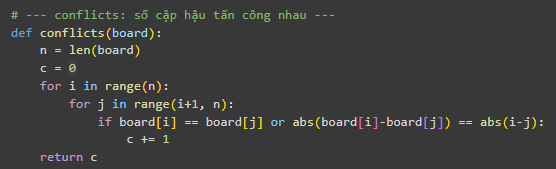
Bước 1: Tạo trạng thái khởi đầu

* Một trạng thái ban đầu được tạo ngẫu nhiên với mỗi quân hậu nằm ở một hàng ngẫu nhiên trong từng cột.
* Hàm sử dụng: random\_state(n)



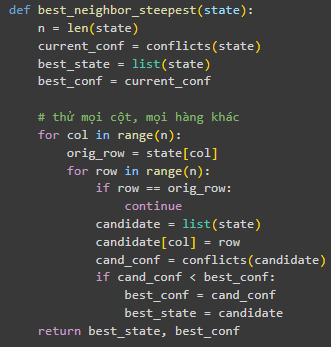
Bước 2: Tính số lượng xung đột (conflicts)

* Hàm conflicts(board) tính số cặp hậu tấn công nhau (cùng hàng hoặc cùng đường chéo).
* Đây là hàm mục tiêu cần tối thiểu hóa.



Bước 3: Tìm trạng thái lân cận tốt nhất

* Trong mỗi bước, thuật toán xem xét di chuyển từng quân hậu trong cột của nó đến các hàng khác trong cùng cột.
* Đối với mỗi khả năng, thuật toán tính số lượng xung đột.
* Chọn trạng thái lân cận có số xung đột thấp nhất (steepest improvement).
* Hàm sử dụng: best\_neighbor\_steepest(state)



Bước 4: Di chuyển hoặc dừng lại

* Nếu trạng thái lân cận tốt hơn, cập nhật trạng thái hiện tại.
* Nếu không tìm thấy trạng thái nào tốt hơn, dừng lại vì đã đạt cực trị địa phương.
* Nếu số xung đột bằng 0, thuật toán kết thúc với lời giải hợp lệ.

**Kết quả:**

Trong lần chạy thuật toán với n = 4 và seed = 1234, quá trình tìm kiếm diễn ra như sau:

* Trạng thái khởi đầu: [2, 3, 0, 0] (ẩn trong code nhưng được xác định ngẫu nhiên từ seed). Số lượng xung đột ban đầu là 4.
* Step 1: Thuật toán tìm được trạng thái lân cận tốt hơn với 2 xung đột.

Step 1: conflicts = 2

* Step 2: Tìm được trạng thái tiếp theo có 1 xung đột.

Step 2: conflicts = 1

* Step 3: Không còn trạng thái lân cận nào tốt hơn, thuật toán dừng lại tại cực trị địa phương.  
   Stopped at step 2. Local optimum with 1 conflicts.

Trạng thái cuối cùng:

* Board: [3, 0, 2, 1]
* Số lượng xung đột còn lại: 1

Kết luận: Trường hợp này cho thấy thuật toán không đảm bảo tìm được lời giải tối ưu nếu chỉ chạy một lần.

## Steepest-ascend Hill Climbing Search with Random Restarts

Steepest-Ascent Hill Climbing with Random Restarts là phiên bản cải tiến của thuật toán Steepest-Ascent Hill Climbing nhằm khắc phục nhược điểm dễ mắc kẹt tại cực trị địa phương (local optimum).

Cách hoạt động:

* Thuật toán thực hiện nhiều lần tìm kiếm theo phương pháp hill climbing từ các trạng thái khởi đầu ngẫu nhiên khác nhau.
* Nếu một lần chạy không tìm được lời giải (conflicts ≠ 0), thuật toán sẽ "khởi động lại" (restart) bằng một trạng thái ngẫu nhiên mới.
* Quá trình lặp lại cho đến khi tìm được lời giải hoặc đạt số lần restart tối đa.

**Các bước làm:**

Bước 1: Khởi tạo

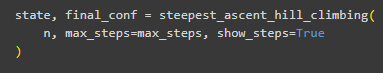
* Người dùng cung cấp các tham số:
  + n: số lượng quân hậu (n x n bàn cờ)  
    max\_steps: số bước tối đa cho mỗi lần hill climbing
  + max\_restarts: số lần khởi động lại tối đa
* seed: để đảm bảo tái hiện kết quả (nếu có)

Bước 2: Thực hiện Hill Climbing với Restart

* Với mỗi lần khởi động lại:
  1. Tạo một trạng thái ban đầu ngẫu nhiên.



* 1. Áp dụng hill climbing (tăng dần dốc nhất).



* 1. Nếu tìm được lời giải (số xung đột = 0), dừng lại và trả kết quả.



* 1. Nếu không, tiếp tục khởi động lại với trạng thái mới.

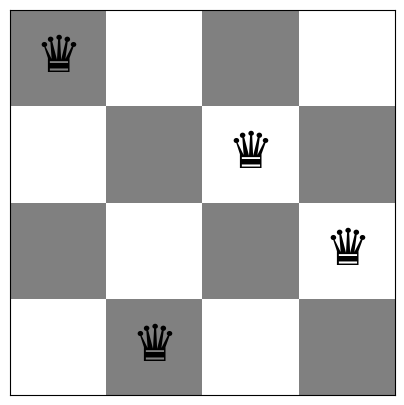


* Hàm được sử dụng:  
   hill\_climbing\_with\_random\_restarts\_show(...)  
   Gọi đến:  
   steepest\_ascent\_hill\_climbing(...) (đã được trình bày trong phần trước)

**Kết quả:** Trong lần chạy với n = 4, số lần restart tối đa là 20 và seed = 1234, kết quả thuật toán thu được như sau:

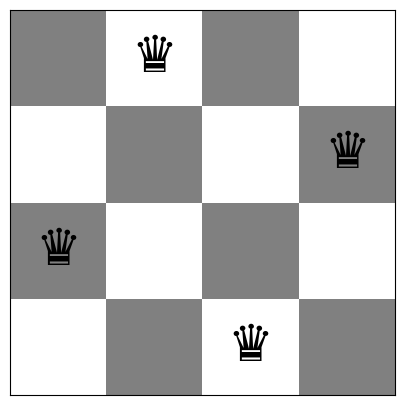
Restart #1:

* Trạng thái ban đầu có 4 xung đột conflicts = 4
* Bước 1: Di chuyển hậu và giảm số xung đột xuống còn 2.
* Bước 2: Tiếp tục cải thiện, còn 1 xung đột.
* Không còn trạng thái lân cận nào tốt hơn ⇒ thuật toán dừng lại tại cực trị địa phương với 1 xung đột.



Restart #2:

* Trạng thái ban đầu tiếp tục có 4 xung đột.
* Bước 1: Giảm xuống còn 1 xung đột.
* Bước 2: Tiếp tục cải thiện, đạt 0 xung đột.
* Đây là lời giải hợp lệ cho bài toán 4 hậu.
* Thuật toán dừng ngay sau restart thứ 2.



Trạng thái cuối cùng tìm được:

* Final solution: [1, 3, 0, 2]
* Final conflicts: 0

Kết luận:

* Thuật toán đã thành công trong việc tránh mắc kẹt tại cực trị địa phương nhờ vào kỹ thuật random restarts.
* Dù lần đầu không thành công, nhưng chỉ sau 2 lần thử, thuật toán đã tìm được lời giải chính xác.
* Điều này thể hiện rõ ưu điểm của phương pháp restart: cải thiện độ tin cậy của thuật toán hill climbing, đặc biệt trong không gian trạng thái có nhiều cực trị địa phương.

## Stochastic Hill Climbing

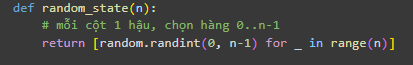
**Stochastic Hill Climbing** là một biến thể của thuật toán **Hill Climbing**, với điểm khác biệt chính là: Thay vì luôn chọn trạng thái lân cận tốt nhất (best move), thuật toán sẽ **chọn ngẫu nhiên một trong số các trạng thái cải thiện** (tức là các “uphill moves” – có ít xung đột hơn trạng thái hiện tại).

Mục tiêu chính là tránh rơi vào các cực trị địa phương như trong thuật toán hill climbing truyền thống bằng cách đưa vào tính ngẫu nhiên trong quá trình lựa chọn bước tiếp theo.

**Các bước làm**:

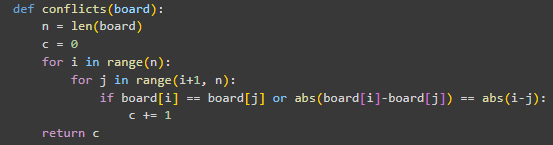
Bước 1: Tạo trạng thái khởi đầu

* Thuật toán tạo một trạng thái ban đầu ngẫu nhiên.
* Mỗi quân hậu được đặt ở một hàng ngẫu nhiên trong từng cột.



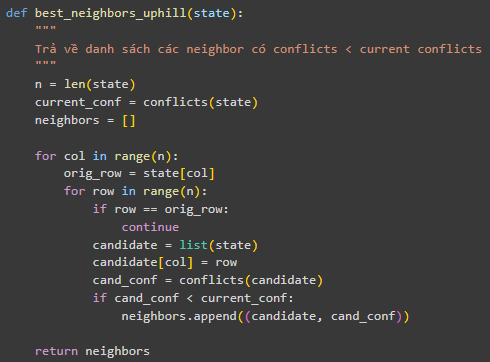
Bước 2: Tính số xung đột

* Dùng hàm conflicts(board) để tính số lượng xung đột giữa các quân hậu.
* Đây là hàm mục tiêu cần tối thiểu hóa.



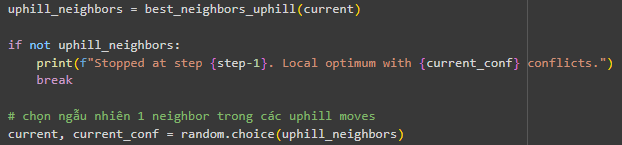
Bước 3: Tìm các trạng thái “uphill”

* Với mỗi hậu, thử di chuyển nó đến các hàng khác trong cùng cột.
* Nếu nước đi nào giúp giảm số xung đột so với trạng thái hiện tại → đó là một **uphill move**.
* Thuật toán thu thập **tất cả các uphill neighbors** vào danh sách.



Bước 4: Chọn ngẫu nhiên và cập nhật

* Nếu danh sách uphill neighbors **không rỗng**, chọn ngẫu nhiên một trạng thái để di chuyển.
* Nếu không có trạng thái cải thiện nào, dừng lại tại **local optimum**.



Bước 5: Kiểm tra điều kiện dừng

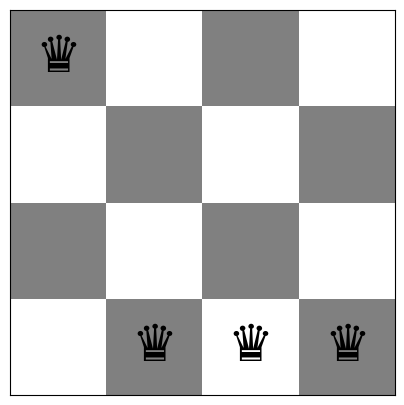
* Nếu trạng thái hiện tại có **0 xung đột**, thì đã tìm được lời giải.
* Nếu không còn hướng đi cải thiện → dừng thuật toán.



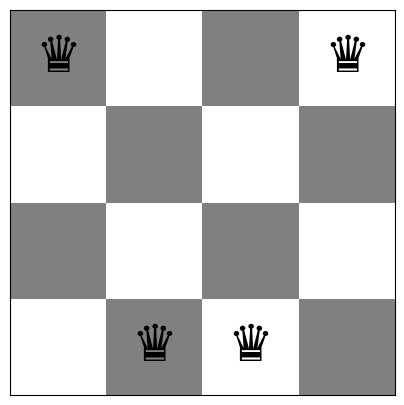
**Kết quả:**

Với đầu vào n = 4 và seed = 1234, thuật toán cho kết quả như sau:

**Initial state (conflicts = 4)**



**Step 1: conflicts = 2**

****

**Stopped at step 1. Local optimum with 2 conflicts.**

**Final solution: [3, 0, 0, 3]**

**Final conflicts: 2**

Phân tích kết quả:

* **Trạng thái ban đầu** có **4 xung đột**.
* **Bước 1**, thuật toán tìm được một trạng thái lân cận có **2 xung đột** và di chuyển đến đó.
* Ở bước tiếp theo, **không còn hướng đi nào có thể cải thiện thêm** → thuật toán dừng lại tại **cực trị địa phương** với **2 xung đột còn lại**.
* Thuật toán **không tìm được lời giải hoàn chỉnh** (số xung đột ≠ 0).

## First-choice Hill Climbing

**First-Choice Hill Climbing** là một biến thể của **Stochastic Hill Climbing**, nhưng thay vì tạo toàn bộ các trạng thái lân cận rồi chọn một cái ngẫu nhiên, thuật toán **chỉ sinh ra một neighbor (lân cận) ngẫu nhiên tại mỗi bước**.

Nếu neighbor đó **cải thiện kết quả** (tức là có số xung đột ít hơn trạng thái hiện tại), nó được **chấp nhận ngay lập tức**. Nếu không, thuật toán tiếp tục tạo neighbor mới ở bước kế tiếp.

Đây là một cách tiếp cận nhanh hơn, đơn giản hơn và phù hợp với các không gian trạng thái lớn vì không cần duyệt tất cả các lân cận.

**Các bước làm:**

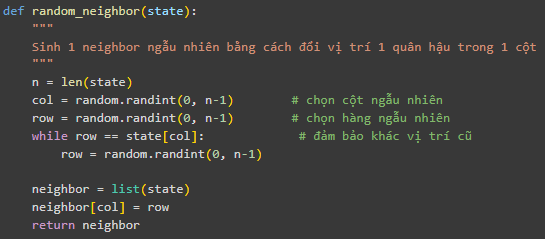
Bước 1: Sinh trạng thái khởi đầu

* Sử dụng hàm **random\_state(n)** để sinh một cấu hình ngẫu nhiên với mỗi quân hậu nằm ở một hàng bất kỳ trong từng cột.



Bước 2: Sinh 1 neighbor ngẫu nhiên

* Dùng hàm random\_neighbor(state) để di chuyển ngẫu nhiên một quân hậu đến hàng khác trong cùng cột
* Chỉ 1 neighbor được sinh tại mỗi bước (khác với các thuật toán sinh tất cả neighbors).

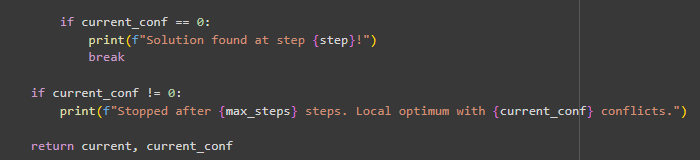


Bước 3: So sánh và chấp nhận

* Nếu neighbor có số xung đột ít hơn trạng thái hiện tại → chấp nhận move đó.
* Nếu không tốt hơn → bỏ qua, không thay đổi trạng thái hiện tại.

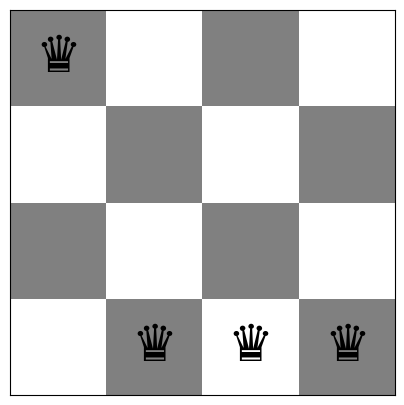


Bước 4: Dừng thuật toán

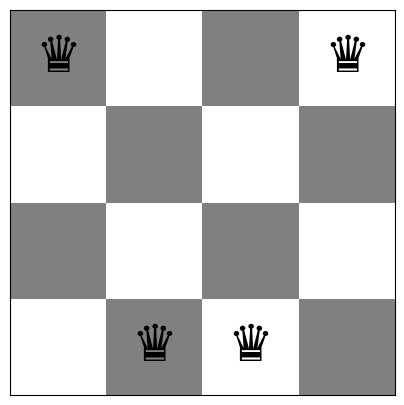
* Dừng nếu:
  + Tìm được trạng thái không còn xung đột (lời giải).
* Hoặc đã thực hiện đến max\_steps mà vẫn chưa tìm được lời giải → xem như rơi vào cực trị địa phương. ****

**Kết quả;** Với đầu vào n = 4, max\_steps = 500, và seed = 1234, thuật toán cho ra kết quả sau:

Initial state (conflicts = 4)



Step 4: conflicts = 2



Stopped after 500 steps. Local optimum with 2 conflicts.

Final solution: [3, 0, 0, 3]

Final conflicts: 2

Phân tích quá trình:

* Bắt đầu từ trạng thái có **4 xung đột**.
* Sau vài bước, **bước thứ 4** cải thiện xuống còn **2 xung đột**.
* Tuy nhiên, trong **496 bước còn lại**, thuật toán không tìm được neighbor nào tốt hơn
* → Thuật toán dừng sau 500 bước mà **không tìm được lời giải hoàn chỉnh**.

## Simulated Annealing

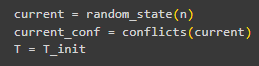
**Simulated Annealing** (tạm dịch: "tôi luyện mô phỏng") là một thuật toán tối ưu ngẫu nhiên, bắt nguồn từ quá trình tôi luyện kim loại trong vật lý. Ý tưởng cốt lõi là:

* Thuật toán có thể **chấp nhận các bước đi kém hơn tạm thời** với một xác suất nhất định, nhằm **thoát khỏi cực trị địa phương (local optimum)**.
* Xác suất chấp nhận này **giảm dần theo thời gian**, mô phỏng quá trình làm nguội từ từ của vật liệu trong tôi luyện kim loại.

**Các bước làm:**

Bước 1: Khởi tạo trạng thái

* Sinh một trạng thái ngẫu nhiên ban đầu với random\_state(n).
* Gán nhiệt độ khởi điểm T\_init (ví dụ: 100).
* Tính số xung đột ban đầu.



Bước 2: Sinh neighbor ngẫu nhiên

* Tạo một neighbor ngẫu nhiên bằng cách thay đổi vị trí 1 quân hậu (random\_neighbor(state)).

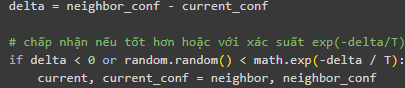


Bước 3: Tính độ chênh lệch (delta)

Nếu delta < 0 → neighbor tốt hơn → **chấp nhận ngay**.

Nếu delta >= 0 → neighbor tệ hơn → **chấp nhận với xác suất**:

Xác suất này **cao hơn khi T lớn** và giảm dần theo thời gian.



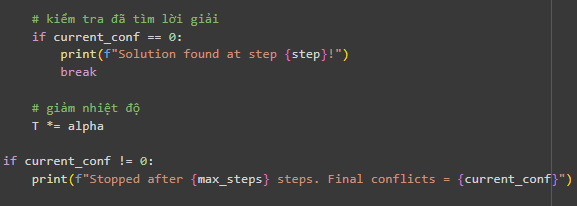
Bước 4: Giảm nhiệt độ

Sau mỗi bước, làm nguội nhiệt độ bằng: T \*= alpha

Ví dụ: alpha = 0.99 nghĩa là sau mỗi bước, nhiệt độ giảm 1%.

Bước 5: Điều kiện dừng

* Nếu tìm được lời giải (conflicts == 0) → kết thúc.
* Nếu đạt tới max\_steps mà chưa giải được → dừng.



**Kết quả:** khi chạy với n=4, T\_init=100, alpha=0.99, seed=1234:

Initial state (conflicts = 4)

**...**

**Step 4: conflicts = 1**

**...**

**Step 30: conflicts = 1**

**...**

**Step 53: conflicts = 1**

**...**

**Step 79: conflicts = 0**

**Solution found at step 79!**

**Final solution: [2, 0, 3, 1]**

**Final conflicts: 0**

Phân tích:

* Bắt đầu với trạng thái có **4 xung đột**.
* Trong quá trình, thuật toán **nhiều lần chấp nhận các trạng thái tệ hơn** (ví dụ: từ 1 → 3 xung đột) nhờ nhiệt độ còn cao.
* Sau **79 bước**, thuật toán tìm được **lời giải hoàn chỉnh** với **0 xung đột**.
* Kết quả chứng minh sức mạnh của Simulated Annealing trong việc **thoát khỏi local optimum** nhờ cơ chế chấp nhận ngẫu nhiên thông minh.

## Compare Performance

**1. Mục tiêu**

So sánh hiệu năng của 4 thuật toán giải bài toán **N-Queens**:

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

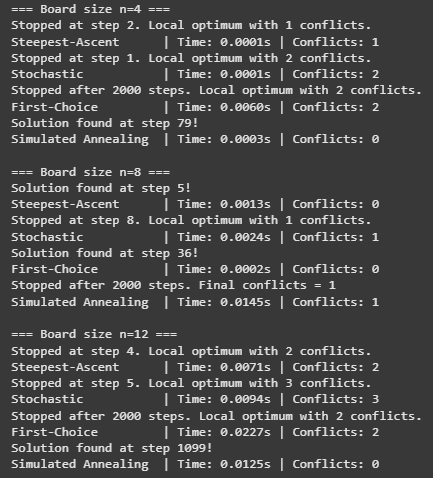
Theo các tiêu chí:

* Thời gian chạy (runtime)
* Giá trị hàm mục tiêu (số xung đột còn lại)
* Khả năng mở rộng (scalability) khi tăng kích thước bàn cờ (n = 4, 8, 12)

2. Cách thực hiện

* Sử dụng thư viện time để đo thời gian chạy của mỗi thuật toán.
* Mỗi thuật toán được chạy với cùng seed (1234) để đảm bảo tính so sánh công bằng.
* Không hiển thị từng bước (show\_steps=False) để tối ưu tốc độ benchmark.
* Giới hạn số bước tối đa là 2000.

3. Kết quả benchmark



4. Phân tích chi tiết

Thời gian chạy (runtime)

* Tất cả các thuật toán đều rất **nhanh với n = 4**, thường < 0.01s.
* Khi n tăng lên:
  + **Simulated Annealing** có thời gian tăng chậm, ổn định.
  + **First-Choice** có xu hướng tốn thời gian nhiều hơn khi n lớn.
  + **Steepest-Ascent** và **Stochastic** vẫn chạy khá nhanh.

Giá trị hàm mục tiêu (final conflicts)

* Với n=4:
  + Chỉ **Simulated Annealing** tìm ra lời giải hoàn chỉnh (0 xung đột).
* Với n=8:
  + **Steepest-Ascent** và **First-Choice** tìm được lời giải.
* Với n=12:
  + **Chỉ Simulated Annealing** giải được hoàn toàn.
  + Các thuật toán khác **bị kẹt tại local optimum**.

Khả năng mở rộng (scalability)

* **Simulated Annealing** thể hiện rõ khả năng **giải quyết được bài toán ở quy mô lớn hơn**, nhờ vào cơ chế chấp nhận bước đi tệ một cách có kiểm soát.
* **Steepest-Ascent** dễ bị **kẹt tại local optimum** khi n lớn.
* **Stochastic** và **First-Choice** cũng gặp vấn đề tương tự ở n = 12.

5. Kết luận & So sánh

| **Thuật toán** | |  | | --- |  | **Ưu điểm** | | --- | | |  | | --- |  | **Nhược điểm** | | --- | |
| --- | --- | --- | --- | --- | --- | --- |
| Steepest-Ascent | Rất nhanh, đơn giản | Dễ kẹt local optimum |
| Stochastic | Đa dạng hướng đi | Hiệu quả không ổn định |
| First-Choice | Chạy nhanh hơn Stochastic | Hiệu quả giảm khi n tăng |
| Simulated Annealing | Giải được bài toán lớn, thoát local optimum tốt | Chạy chậm hơn, cần điều chỉnh tham số |