**Solving the n-Queens Problem using Local Search**

Kết quả học tập

* Triển khai nhiều biến thể khác nhau của thuật toán leo đồi (Hill Climbing) để giải quyết bài toán n-Queens.
* Áp dụng phương pháp mô phỏng tôi luyện (Simulated Annealing) cùng với lịch trình nhiệt độ (temperature schedule) phù hợp nhằm vượt qua các cực trị cục bộ (local optima).
* So sánh hiệu suất của các thuật toán dựa trên các chỉ số: thời gian chạy (runtime), chất lượng nghiệm (solution quality), và tỉ lệ thành công (success rate).
* Phân tích và trực quan hóa hiệu suất của thuật toán trên các kích thước bàn cờ khác nhau.
* Đối với học viên cao học: Thiết kế và thử nghiệm các phép di chuyển cục bộ thay thế (alternative local move operators) nhằm nâng cao hiệu quả tìm kiếm.

Hướng dẫn

* Tổng điểm: Sinh viên đại học: 100 điểm(+5 điểm thưởng) / Học viên cao học: 110 điểm.
* Hoàn thiện toàn bộ nội dung trong notebook. Sử dụng các ô (cells) được cung cấp sẵn và bổ sung thêm các ô mã (code cells) hoặc ghi chú (markdown cells) khi cần thiết.
* Sau khi hoàn tất, xuất (render) toàn bộ notebook và nộp dưới định dạng tệp HTML.

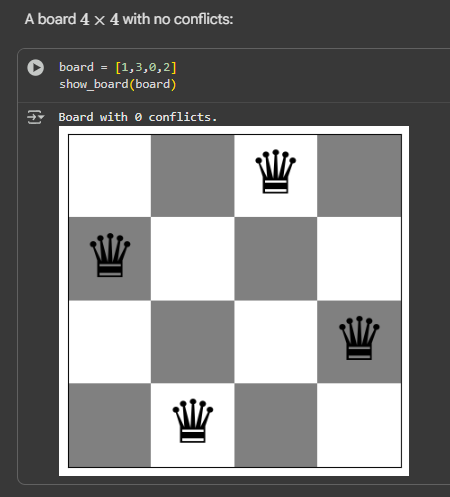
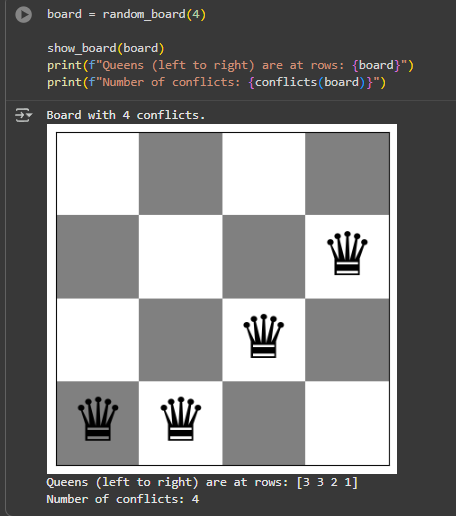
**Vấn đề N-Queens**

* **Mục tiêu:** Tìm một cách sắp xếp *n* quân hậu trên bàn cờ *n × n* sao cho không có hai quân hậu nào cùng nằm trên cùng một hàng, cột hoặc đường chéo.
* **Không gian trạng thái (State space):** Một trạng thái là một cách sắp xếp các quân hậu trên bàn cờ. Chúng ta giới hạn không gian trạng thái ở những sắp xếp chỉ có **một quân hậu trên mỗi cột**. Mỗi trạng thái được biểu diễn bằng một **vector số nguyên** q={q1,q2,…,qn}, trong đó mỗi phần tử qi biểu thị **vị trí hàng** của quân hậu trong cột thứ i (từ trái sang phải). Ta sẽ gọi mỗi trạng thái là một “bàn cờ”.
* **Hàm mục tiêu (Objective function):** Là **số cặp xung đột** giữa các quân hậu (ví dụ: hai quân hậu nằm trên cùng hàng, cột hoặc đường chéo). Bài toán tối ưu là tìm một sắp xếp tối ưu q∗ của n quân hậu sao cho: minimize: conflicts(q) với điều kiện rằng: q chỉ chứa một quân hậu trên mỗi cột.

Lưu ý: điều kiện ràng buộc này đã được đảm bảo bởi cách định nghĩa của không gian trạng thái.

* **Phép di chuyển cải thiện cục bộ (Local improvement move):** Di chuyển một quân hậu sang **một hàng khác trong cùng cột**.
* **Điều kiện dừng (Termination):** Đối với bài toán này, luôn tồn tại một sắp xếp q∗ sao cho conflicts(q∗)=0. Tuy nhiên, trong quá trình tối ưu, các bước cải thiện cục bộ có thể khiến ta **rơi vào cực tiểu cục bộ** (local minimum).

**Khởi tạo bàn cờ**

****

**Các Nhiệm vụ**

Chung [10 Điểm]

1. Đảm bảo rằng bạn sử dụng phiên bản mới nhất của sổ tay (notebook) này. Đồng bộ hóa kho lưu trữ đã fork của bạn và kéo (pull) bản sửa đổi mới nhất.
2. Việc triển khai của bạn có thể sử dụng các thư viện như math, numpy, scipy, nhưng không được sử dụng các thư viện triển khai các tác nhân thông minh (intelligent agents) hoặc các thuật toán tìm kiếm hoàn chỉnh. Cố gắng giữ cho mã nguồn đơn giản! Trong khóa học này, chúng ta muốn tìm hiểu về các thuật toán và thường không cần sử dụng thiết kế hướng đối tượng (object-oriented design).
3. Sổ tay của bạn cần được định dạng chuyên nghiệp.

* Thêm các khối markdown bổ sung cho mô tả của bạn, nhận xét trong mã, thêm bảng và sử dụng matplotlib để tạo biểu đồ khi thích hợp.
* Không hiển thị đầu ra gỡ lỗi (debugging output) hoặc bao gồm một lượng đầu ra quá mức.
* Kiểm tra xem tệp bạn gửi có dễ đọc và chứa tất cả các hình ảnh/biểu đồ (figures) hay không.

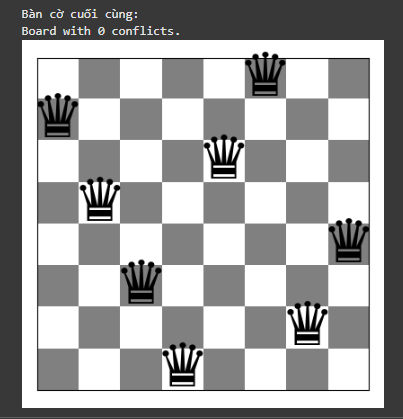
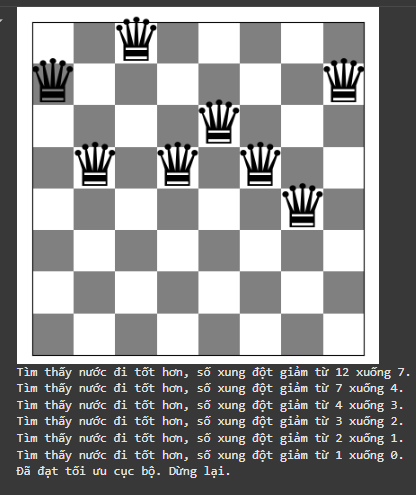
1. Tài liệu hóa mã của bạn. Sử dụng nhận xét trong mã và thêm một bài thảo luận về cách thức triển khai của bạn hoạt động và các lựa chọn thiết kế của bạn.

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

Tính toán hàm mục tiêu cho toàn bộ nước đi cục bộ(xem định nghĩa về các nước đi cục bộ ở trên) và luôn chọn nước đi tốt nhất. Nếu không có bước di chuyển cục bộ nào cải thiện được giá trị mục tiêu, thì bạn đã đạt đến tối ưu cục bộ (local optimum).

Thuật toán **Steepest-Ascent Hill Climbing (Leo đồi dốc nhất)** là một phương pháp tìm kiếm cục bộ (*local search*) trong lĩnh vực **Trí tuệ nhân tạo (Artificial Intelligence)**.  
Ý tưởng của thuật toán là:

* Xuất phát từ một **trạng thái ban đầu (initial state)**.
* Tại mỗi bước, xét tất cả các **trạng thái lân cận (neighbors)** có thể đạt được bằng các phép di chuyển hợp lệ.
* **Chọn trạng thái có giá trị hàm mục tiêu tốt nhất** (ít xung đột nhất) và di chuyển đến đó.
* Tiếp tục lặp lại quá trình này cho đến khi **không thể cải thiện thêm** (đạt tối ưu cục bộ).

Trong bài toán N-Queens, mỗi **trạng thái** là một cách sắp xếp N quân hậu trên bàn cờ, và **hàm mục tiêu** chính là số lượng xung đột (số cặp quân hậu tấn công nhau).  
Thuật toán sẽ tìm cách **giảm dần số xung đột** cho đến khi không thể giảm thêm được nữa. 

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

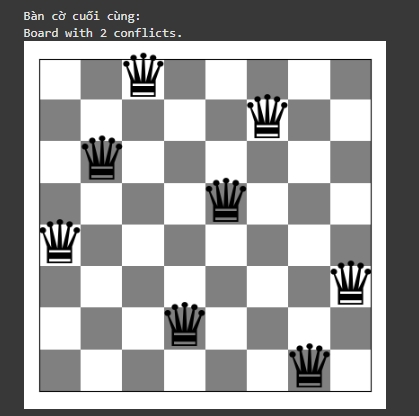
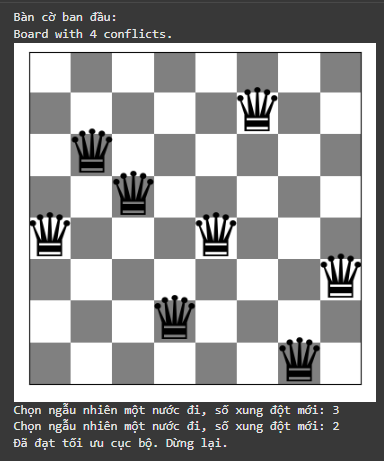
Xác định tất cả các nước đi giúp cải thiện tình hình (gọi là các "uphill moves" - nước đi lên dốc).

Sau đó, chọn ngẫu nhiên một nước đi từ trong danh sách các nước đi tốt đó.

Thuật toán **Stochastic Hill Climbing (Leo đồi ngẫu nhiên)** là một biến thể của phương pháp **Hill Climbing (Leo đồi)**, thuộc nhóm **tìm kiếm cục bộ (local search algorithms)**.  
Trong khi thuật toán Steepest-Ascent Hill Climbing luôn **chọn nước đi tốt nhất trong tất cả các lựa chọn**, thì Stochastic Hill Climbing lại **chọn ngẫu nhiên một nước đi tốt hơn** trong số những lựa chọn khả thi.

**Ý tưởng chính:**

* Xuất phát từ một trạng thái ban đầu.
* Xét các trạng thái lân cận và **chỉ giữ lại các trạng thái tốt hơn (giảm xung đột)**.
* Thay vì chọn “tốt nhất trong số tốt hơn”, thuật toán **chọn ngẫu nhiên một trạng thái cải thiện** để tiếp tục tìm kiếm.  
  Cách tiếp cận này giúp thuật toán **tránh bị kẹt ở các điểm tối ưu cục bộ** và tăng khả năng tìm được nghiệm tối ưu toàn cục.



**Task 3: Stochastic Hill Climbing 2 [20 Points]**

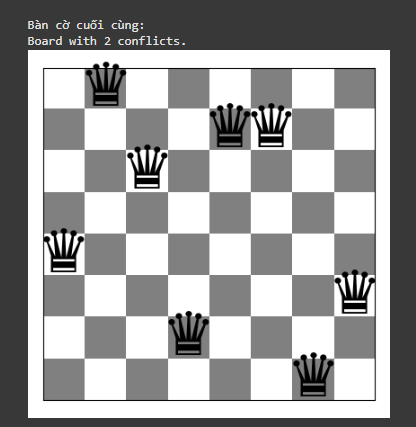
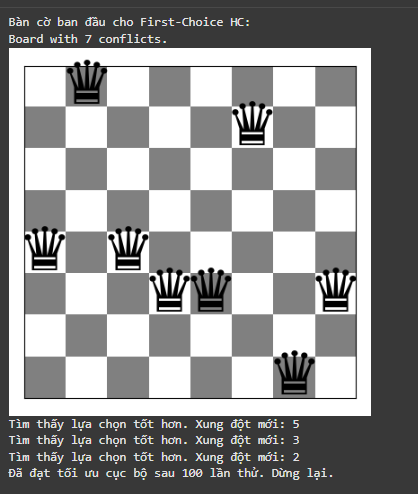
Một phiên bản phổ biến của phương pháp leo núi ngẫu nhiên (stochastic hill climbing) chỉ tạo ra một người láng giềng địa phương ngẫu nhiên duy nhất tại một thời điểm và chấp nhận người đó nếu nó có giá trị hàm mục tiêu tốt hơn trạng thái hiện tại. Điều này rất hiệu quả nếu mỗi trạng thái có nhiều trạng thái kế tiếp khả thi. Phương pháp này được gọi là "First-choice hill climbing" trong textbook.

Ghi chú:Việc phát hiện cực trị cục bộ là khó khăn! Ví dụ, bạn có thể dừng lại nếu không thể cải thiện hàm mục tiêu trong x lần thử gần đây nhất.

Thuật toán **First-Choice Hill Climbing** là một biến thể ngẫu nhiên của Hill Climbing truyền thống, được thiết kế nhằm giảm thời gian tìm kiếm và tránh bị kẹt sớm trong tối ưu cục bộ.  
Thay vì duyệt qua toàn bộ không gian lân cận như Steepest-Ascent hay chọn ngẫu nhiên từ tập các nước đi tốt hơn như Stochastic Hill Climbing, First-Choice Hill Climbing chỉ thử một **số lượng giới hạn các nước đi ngẫu nhiên (max\_tries)** và **chấp nhận ngay** nước đi đầu tiên tốt hơn trạng thái hiện tại.

**Ý tưởng chính**

* Xuất phát từ một trạng thái ngẫu nhiên (bàn cờ khởi tạo).
* Tại mỗi vòng lặp, chọn **một nước đi ngẫu nhiên** (tức là thay đổi vị trí của một quân hậu trên cột bất kỳ).
* Nếu nước đi đó giúp **giảm số xung đột**, thuật toán **chấp nhận ngay** mà không cần xem xét thêm các nước đi khác.
* Nếu sau max\_tries lần thử mà **không có nước đi nào tốt hơn**, ta xem như đã đạt **tối ưu cục bộ** và dừng lại.
* Quá trình này được lặp lại cho đến khi tìm được lời giải (số xung đột = 0) hoặc không thể cải thiện thêm.



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

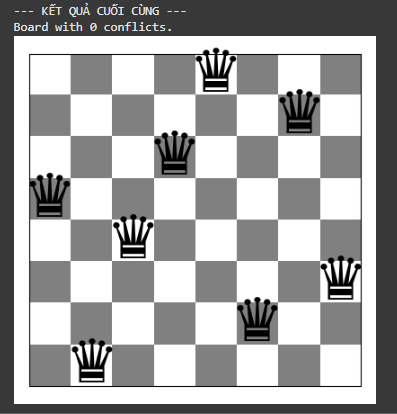
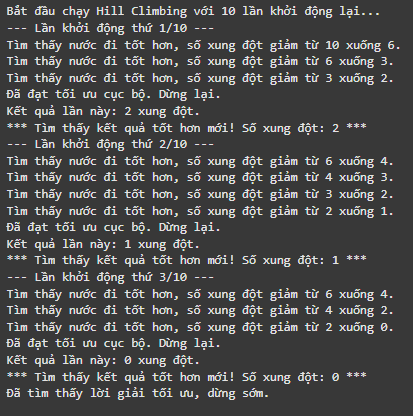
Leo núi thường sẽ kết thúc ở cực trị cục bộ. Khởi động lại từng thuật toán leo núi ba lần lên đến 100 lần với một bảng ngẫu nhiên để tìm giải pháp tốt hơn (hy vọng là tối ưu). Lưu ý rằng khởi động lại chỉ có nghĩa là chạy thuật toán nhiều lần bắt đầu với một bảng ngẫu nhiên mới.

Một trong những hạn chế lớn nhất của các thuật toán Hill Climbing (bao gồm **Steepest-Ascent**, **Stochastic**, và **First-Choice**) là khả năng **mắc kẹt trong cực trị cục bộ (local optimum)**.  
Để khắc phục vấn đề này, kỹ thuật **Random Restart Hill Climbing** được đề xuất: thay vì chỉ chạy một lần từ một trạng thái khởi tạo duy nhất, thuật toán **chạy nhiều lần** từ **nhiều trạng thái ngẫu nhiên khác nhau**, giúp tăng khả năng tìm được **lời giải toàn cục (global optimum)**.

**Ý tưởng chính**

* Thực hiện **nhiều lần khởi tạo ngẫu nhiên** (mỗi lần bắt đầu từ một bàn cờ khác nhau).
* Mỗi lần khởi tạo, chạy một thuật toán Hill Climbing cụ thể (ví dụ: Steepest-Ascent, Stochastic, hoặc First-Choice).
* Ghi nhận **kết quả tốt nhất** trong số tất cả các lần chạy.
* Dừng sớm nếu tìm thấy lời giải tối ưu (không còn xung đột).

Mục tiêu là tận dụng **đa dạng điểm khởi đầu** để tránh việc thuật toán bị kẹt mãi trong một vùng nghiệm xấu.



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

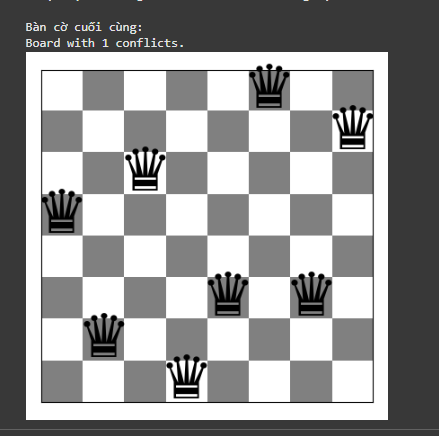
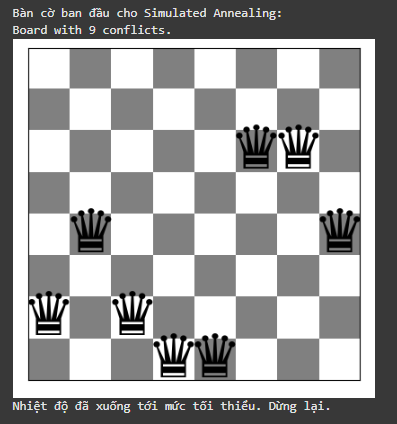
Thuật toán nung giả lập (simulated annealing) là một dạng leo đồi ngẫu nhiên (stochastic hill climbing) giúp tránh cực trị cục bộ bằng cách cho phép các bước đi xuống với xác suất tỉ lệ thuận với nhiệt độ. Nhiệt độ được giảm dần ở mỗi vòng lặp theo một lịch trình nung. Bạn cần thử nghiệm với lịch trình nung (tìm hướng dẫn trên Google về vấn đề này).

* 1. Hãy triển khai thuật toán nung giả lập cho bài toán n-Queens.
  2. Tạo một trực quan hóa quá trình tìm kiếm (biểu đồ đường thể hiện số xung đột thay đổi như thế nào khi thuật toán tiến triển).
  3. Sử dụng trực quan hóa này cho các thí nghiệm với các lựa chọn khác nhau về lịch trình nung và thảo luận những gì bạn đã học được.

Thuật toán **Simulated Annealing (SA)** là một kỹ thuật tìm kiếm ngẫu nhiên bắt nguồn từ **quá trình luyện kim trong vật lý** — khi kim loại được nung nóng rồi làm nguội dần để đạt trạng thái năng lượng thấp nhất.  
Trong lĩnh vực tối ưu hóa, SA mô phỏng quá trình này bằng cách **cho phép chấp nhận tạm thời các nghiệm tệ hơn** với một xác suất giảm dần theo thời gian.  
Nhờ đó, thuật toán **có khả năng thoát khỏi các cực trị cục bộ (local minima)** mà các phương pháp như Hill Climbing thường bị mắc kẹt.

**Ý tưởng chính**

* Bắt đầu từ một trạng thái khởi tạo ngẫu nhiên.
* Ở mỗi bước, chọn **một hàng xóm (neighbor)** của trạng thái hiện tại bằng cách di chuyển ngẫu nhiên một quân hậu.
* Nếu hàng xóm có **số xung đột ít hơn** → chấp nhận ngay.
* Nếu hàng xóm **tệ hơn**, vẫn có thể chấp nhận **với xác suất giảm dần theo nhiệt độ**.
* Nhiệt độ được giảm dần (theo “làm nguội”), khiến thuật toán ngày càng trở nên "bảo thủ" — ít chấp nhận nghiệm tệ hơn.
* Quá trình dừng khi nhiệt độ đạt ngưỡng cực thấp hoặc không còn cải thiện.



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

So sánh

So sánh các thuật toán dựa trên thời gian chạy và giá trị của hàm mục tiêu. Sử dụng các bàn cờ có kích thước 4 và 8 để khám phá cách các thuật toán khác nhau hoạt động. Hãy chắc chắn rằng bạn chạy các thuật toán này nhiều lần cho mỗi kích thước bàn cờ (ít nhất 100 lần) với các bàn cờ khởi đầu khác nhau và báo cáo giá trị trung bình.



**Mục tiêu thực nghiệm**

Mục tiêu của phần này là đánh giá và so sánh hiệu suất của bốn thuật toán tìm kiếm cục bộ trong việc giải bài toán N-Queens, bao gồm:

1. **Steepest-Ascent Hill Climbing**
2. **Stochastic Hill Climbing**
3. **First-Choice Hill Climbing**
4. **Simulated Annealing**

Các thuật toán đều được triển khai bằng Python và chạy trên cùng một môi trường tính toán để đảm bảo tính công bằng khi so sánh.

**Phương pháp thực nghiệm**

Hàm run\_experiment() được sử dụng để thực hiện quá trình đánh giá. Hàm này chạy mỗi thuật toán nhiều lần (100 lần) trên các kích thước bàn cờ khác nhau và ghi nhận các chỉ số:

* **Thời gian trung bình chạy (Average Run Time)** – đo bằng giây.
* **Số xung đột trung bình (Average Conflicts)** – số cặp quân hậu tấn công nhau sau khi thuật toán kết thúc.
* **Tỷ lệ thành công (Success Rate)** – phần trăm số lần thuật toán tìm được nghiệm hợp lệ (không có xung đột).

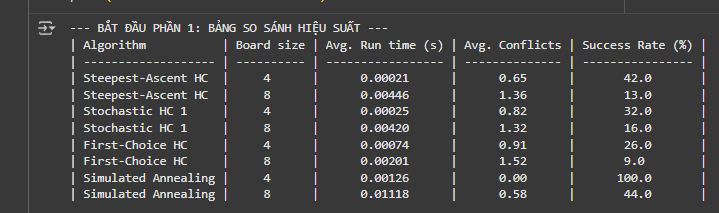
**Cấu hình thuật toán**

Mỗi thuật toán được sử dụng dưới dạng **phiên bản “im lặng” (silent)**, tức là không in kết quả ra màn hình trong mỗi lần chạy, nhằm tránh làm chậm thời gian tính toán.

**Cách so sánh kết quả**

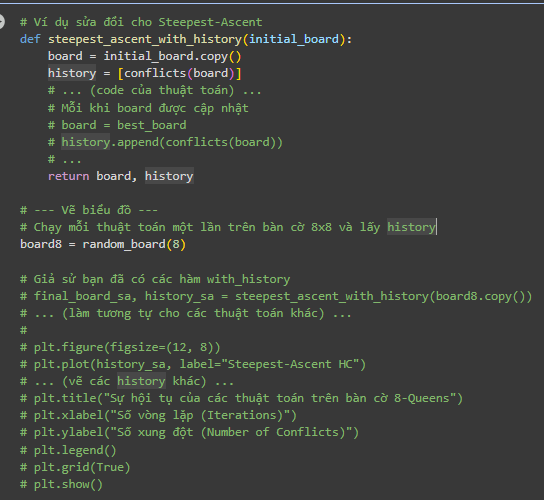
Kết quả được trình bày dưới dạng bảng, với các cột bao gồm:

* **Algorithm**: Tên thuật toán.
* **Board Size**: Kích thước bàn cờ (N).
* **Avg. Run Time (s)**: Thời gian trung bình mỗi lần chạy.
* **Avg. Conflicts**: Số xung đột trung bình còn lại.
* **Success Rate (%)**: Phần trăm số lần tìm được nghiệm đúng.



**Algorithm Convegence**

Đối với mỗi thuật toán được triển khai, hãy mô tả mô hình hội tụ điển hình (cải thiện nhanh ban đầu so với tiến triển đều đặn). Bao gồm một biểu đồ hiển thị giá trị hàm mục tiêu theo số vòng lặp cho một lần chạy đại diện của mỗi thuật toán trên bài toán 8 hậu. Giải thích thuật toán nào thường xuyên gặp tình trạng dừng lại ở các bậc thang hoặc mắc kẹt ở các cực trị cục bộ nhất.



**Problem Size Scalability**

Tạo một đồ thị log-log để hiển thị cách thời gian chạy thay đổi theo kích thước bàn cờ (n=4,8,12,16,20) cho ít nhất hai thuật toán. Ước lượng độ phức tạp thời gian thực nghiệm (Big O) cho mỗi thuật toán dựa trên kết quả của bạn. Xác định thuật toán nào mở rộng tốt nhất đối với các kích thước vấn đề lớn và giải thích lý do.

**Mục tiêu thí nghiệm**

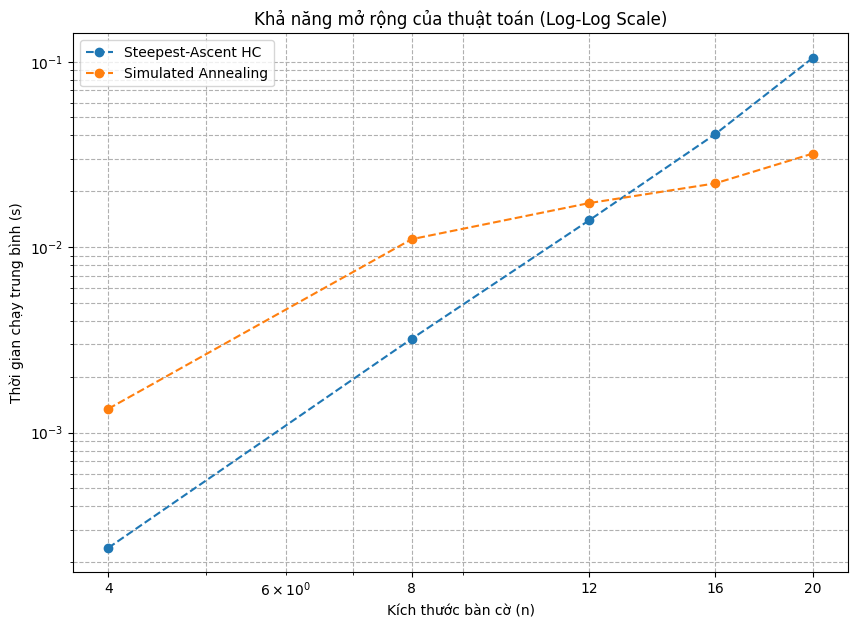
Sau khi so sánh độ chính xác và tốc độ hội tụ của các thuật toán tìm kiếm cục bộ trong bài toán **n-Queens**, nhóm tiếp tục tiến hành **đánh giá khả năng mở rộng (scalability)**.  
Mục tiêu của thí nghiệm này là xác định **mức độ tăng thời gian thực thi** của các thuật toán khi kích thước bàn cờ (n) tăng dần.

**Thiết lập thí nghiệm**

Hai thuật toán được chọn để so sánh là:

* **Steepest-Ascent Hill Climbing** – đại diện cho phương pháp leo đồi thuần túy, luôn chọn lời giải tốt nhất ở mỗi bước.
* **Simulated Annealing** – đại diện cho phương pháp có yếu tố ngẫu nhiên và cơ chế “làm nguội” nhằm thoát khỏi cực trị cục bộ.

Các kích thước bàn cờ được sử dụng lần lượt là **4×4, 8×8, 12×12, 16×16 và 20×20**.  
Mỗi thuật toán được chạy **20 lần độc lập** trên mỗi kích thước để lấy **thời gian chạy trung bình**.  
Để kết quả khách quan, các phiên bản “silent” (không in log ra màn hình) của thuật toán được sử dụng.



**Phân tích kết quả**

Kết quả biểu đồ log-log thể hiện:

* **Steepest-Ascent Hill Climbing** có thời gian tăng **rất nhanh** khi kích thước bàn cờ tăng, do mỗi bước thuật toán cần duyệt **toàn bộ không gian lân cận** để chọn trạng thái tối ưu. Điều này làm chi phí tính toán tăng gần như theo **hàm bậc hai của n**.
* **Simulated Annealing**, mặc dù có nhiều phép thử ngẫu nhiên, lại **mở rộng mượt mà hơn**, do không cần duyệt toàn bộ không gian ứng viên tại mỗi bước. Nhờ vậy, khi n lớn (ví dụ n = 20), thời gian của SA thường **ngắn hơn đáng kể** so với Steepest-Ascent HC.
* Cả hai thuật toán đều có xu hướng **phi tuyến (nonlinear)**, biểu thị rằng bài toán n-Queens trở nên phức tạp hơn đáng kể khi kích thước tăng, đặc biệt với các phương pháp tìm kiếm toàn cục.