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

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

Mục tiêu:

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

Yêu cầu:

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

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

CODE:

A screen shot of a computer program

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

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

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

Yêu cầu:

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

Ý tưởng:

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

CODE:

A screen shot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

A black screen with white text

AI-generated content may be incorrect.

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

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

Ý tưởng:.

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

**Cách hoạt động**

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

CODE:

A screen shot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

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

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

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

**Điểm quan trọng:**

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

Ý tưởng:

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

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

CODE:

A screen shot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

Kết quả:

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

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

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

Nhận xét:

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

Như vậy:

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

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

Yêu cầu chính:

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

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

1. Vẽ biểu đồ (visualization)

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

1. Thí nghiệm với nhiều annealing schedule khác nhau

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

Cách làm:

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

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

**Công thức chấp nhận**:

Nếu neighbor tốt hơn → chấp nhận.

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

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

Kết quả:

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

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

Trong phần thảo luận, bạn nêu được:

* Schedule nào giúp SA tìm ra nghiệm nhanh nhất.
* Schedule nào ổn định hơn (ít bị mắc kẹt).

CODE:

A computer screen with text and images

AI-generated content may be incorrect.

A computer screen with text and images

AI-generated content may be incorrect.

A computer screen shot of a program code

AI-generated content may be incorrect.

A blue graph with numbers and a white background

AI-generated content may be incorrect.

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

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

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

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

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

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

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

Yêu cầu:

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

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

Ý tưởng code:

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

CODE:

A screen shot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer program

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

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

# **Algorithm Convergence**

Yêu cầu:

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

**Phần mô tả mẫu:**

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

**Phân tích plateau / local optima**

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

Code vẽ **biểu đồ convergence:**

A screen shot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer program

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

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

AI-generated content may be incorrect.

A graph with a red line

AI-generated content may be incorrect.

**Mô tả convergence pattern**

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

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

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

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

**Problem Size Scalability**

**Yêu cầu:**

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

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

**Vẽ đồ thị log-log**:

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

**Phân tích kết quả**:

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

**Ý tưởng**:

**Chạy thử nhiều kích thước bàn cờ**:

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

**Chọn ít nhất 2 thuật toán**:

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

**Vẽ log-log plot**:

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

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

**Ước lượng độ phức tạp Big O**:

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

**Phân tích**:

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

CODE:

A screen shot of a computer program

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

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

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

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

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

AI-generated content may be incorrect.

A graph of a problem size

AI-generated content may be incorrect.

A black screen with white text

AI-generated content may be incorrect.

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

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

Giải thích kết quả:

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

Kết luận:

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

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

 Các Move Operators cần cài đặt:

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

**Code Implementation**

A computer screen shot of text

AI-generated content may be incorrect.

A computer screen shot of a program

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

A computer screen shot of a program

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer program

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

AI-generated content may be incorrect.

A graph of different colored lines

AI-generated content may be incorrect.

A diagram of a diagram

AI-generated content may be incorrect.

A graph of blue rectangular shapes

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

AI-generated content may be incorrect.

A diagram of a diagram

AI-generated content may be incorrect.

A graph with blue squares

AI-generated content may be incorrect.

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

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

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

CODE:

A computer screen shot of a program

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

AI-generated content may be incorrect.

A computer screen shot of a program

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

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

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

AI-generated content may be incorrect.

A graph with a line and a green line

AI-generated content may be incorrect.

A computer screen shot of a program code

AI-generated content may be incorrect.

A screenshot of a game

AI-generated content may be incorrect.