UỶ BAN NHÂN DÂN

THÀNH PHỐ HỒ CHÍ MINH

**TRƯỜNG ĐẠI HỌC SÀI GÒN**



**BÁO CÁO PHÂN TÍCH  
MÔ PHỎNG VÀ ĐÁNH GIÁ TÁC NHÂN THÔNG MINH,  
SO SÁNH HIỆU SUẤT CÁC THUẬT TOÁN TÌM KIẾM TRONG MÊ CUNG**

Học phần: Trí tuệ nhân tạo nâng cao

Giảng viên hướng dẫn: TS. Đỗ Như Tài

Sinh viên tham gia:

|  |  |
| --- | --- |
| Cao Tấn Đạt | 3122410069 |
| Lê Nguyễn Anh Đức | 3122410087 |
| Nguyễn Trọng Tấn Sang | 3121560076 |
|  |  |

**Thành phố Hồ Chí Minh, 10/2025**

**Mục lục**

[I. N-QUEENS PROBLEM 1](#_Toc212394530)

[1. Mô tả bài toán 1](#_Toc212394531)

[1.1. Bài toán n-Queens 1](#_Toc212394532)

[1.2. Phương pháp tiếp cận 1](#_Toc212394533)

[2. Các thuật toán được triển khai 1](#_Toc212394534)

[2.1. Steepest\_Ascend Hill Climbing 1](#_Toc212394535)

[2.2. Stochastic Hill Climbing 1 1](#_Toc212394536)

[2.3. Stochastic Hill Climbing 2 (First-Choice) 2](#_Toc212394537)

[2.4. Random Restarts 2](#_Toc212394538)

[2.5. Simulated Annealing 2](#_Toc212394539)

[3. Kết quả thực nghiệm 2](#_Toc212394540)

[3.1. So sánh hiệu suất 2](#_Toc212394541)

[3.2. So sánh hiệu suất 3](#_Toc212394542)

[3.3. Phân tích khả năng mở rộng 4](#_Toc212394543)

[II. GRAPH COLORING 4](#_Toc212394544)

[1. ĐỊNH NGHĨA 4](#_Toc212394545)

[2. CÁC HÀM HỖ TRỢ CHO TÌM KIẾM QUAY LUI 5](#_Toc212394546)

[3. THUẬT TOÁN TÌM KIẾM QUAY LUI 6](#_Toc212394547)

[4. KẾT QUẢ THỰC THI 7](#_Toc212394548)

[III. KẾT LUẬN 8](#_Toc212394549)

**BẢNG PHÂN CÔNG**

|  |  |
| --- | --- |
| **Sinh viên** | **Công việc** |
| Cao Tấn Đạt | Làm silde |
| Lê Nguyễn Anh Đức | Làm word |
| Nguyễn Trọng Tấn Sang | Làm assigment |

1. **N-QUEENS PROBLEM**
2. **Mô tả bài toán**
   1. **Bài toán n-Queens**

Mục tiêu: Tìm cách sắp xếp n quân hậu trên bàn cờ n×n sao cho không có quân hậu nào nằm trên cùng hàng, cột hoặc đường chéo với bất kỳ quân hậu nào khác.

Không gian trạng thái: Một cách sắp xếp các quân hậu trên bàn cờ, giới hạn một quân hậu mỗi cột. Chúng ta biểu diễn trạng thái dưới dạng vector số nguyên q = {q₁, q₂, ..., qₙ}, trong đó mỗi số đại diện cho vị trí hàng của quân hậu từ trái sang phải.

Hàm mục tiêu: Số lượng xung đột theo cặp (các quân hậu đe dọa lẫn nhau).

Bài toán tối ưu: Minimize conflicts(q) với điều kiện q chỉ chứa một quân hậu mỗi cột.

Bước di chuyển cục bộ: Di chuyển một quân hậu sang hàng khác trong cùng cột của nó.

Điều kiện dừng: Luôn tồn tại nghiệm tối ưu với conflicts(q\*) = 0, nhưng các thuật toán tìm kiếm cục bộ có thể bị kẹt ở cực tiểu địa phương.

* 1. **Phương pháp tiếp cận**

Chúng tôi tiếp cận bài toán này bằng các thuật toán tìm kiếm cục bộ, lặp đi lặp lại cải thiện nghiệm ứng viên bằng cách thực hiện các sửa đổi nhỏ. Thách thức chính là thoát khỏi cực tiểu địa phương - các cấu hình mà không có bước di chuyển một quân hậu nào cải thiện nghiệm, nhưng chưa đạt được cực tiểu toàn cục.

1. **Các thuật toán được triển khai**
   1. **Steepest\_Ascend Hill Climbing**

Thuật toán này đánh giá TẤT CẢ các bước di chuyển cục bộ có thể (n × (n-1) bước) ở mỗi lần lặp và luôn chọn bước di chuyển tạo ra sự giảm xung đột lớn nhất.  
Đặc điểm:

* Xác định - luôn chọn bước di chuyển tốt nhất
* Toàn diện - đánh giá tất cả các bước di chuyển có thể
* Độ phức tạp thời gian: O(n²) đánh giá mỗi lần lặp
* Đảm bảo giảm đơn điệu cho đến khi đạt cực tiểu địa phương
  1. **Stochastic Hill Climbing 1**

Biến thể này đánh giá tất cả các bước di chuyển có thể nhưng chọn ngẫu nhiên từ tất cả các bước di chuyển cải thiện (bước đi lên) thay vì luôn chọn bước tốt nhất.  
Đặc điểm:

* Ngẫu nhiên - đưa tính ngẫu nhiên vào việc lựa chọn bước di chuyển
* Chi phí tính toán giống Steepest-Ascend (đánh giá tất cả bước di chuyển)
* Có thể khám phá các đường dẫn nghiệm khác nhau trong các lần chạy khác nhau
  1. **Stochastic Hill Climbing 2 (First-Choice)**

Còn được gọi là First-Choice Hill Climbing, thuật toán này chỉ tạo ra MỘT láng giềng ngẫu nhiên tại một thời điểm và chấp nhận nó nếu nó cải thiện hàm mục tiêu.  
  
Đặc điểm:

* Rất hiệu quả - chỉ một đánh giá cho mỗi bước di chuyển được chấp nhận
* Thực thi nhanh cho không gian trạng thái lớn
* Sử dụng ngưỡng (max\_tries = 100) để phát hiện cực tiểu địa phương
* Có thể bỏ lỡ bước di chuyển cải thiện tốt nhất nhưng được tốc độ đáng kể
  1. **Random Restarts**

Một chiến lược meta khởi động lại bất kỳ thuật toán leo đồi nào với trạng thái ban đầu ngẫu nhiên mới khi bị kẹt ở cực tiểu địa phương.  
Đặc điểm:

* Hoạt động với bất kỳ thuật toán leo đồi cơ sở nào
* Cải thiện đáng kể tỉ lệ thành công
* Chạy tối đa 100 lần khởi động lại hoặc cho đến khi tìm được nghiệm tối ưu
* Đánh đổi: tăng thời gian chạy để có chất lượng nghiệm tốt hơn
  1. **Simulated Annealing**

Lấy cảm hứng từ quá trình ủ luyện kim loại, thuật toán này chấp nhận các bước di chuyển tồi hơn với xác suất giảm dần theo thời gian theo lịch trình nhiệt độ.  
Xác suất chấp nhận: P(chấp nhận) = e^(-Δ/T) cho các bước tăng xung đột thêm Δ  
Lịch trình nhiệt độ: Làm lạnh hình học T(i+1) = α × T(i) với α = 0.95  
Tham số:

* Nhiệt độ ban đầu: 100
* Tỉ lệ làm lạnh: 0.95
* Nhiệt độ tối thiểu: 0.01

Điều này cho phép thuật toán thoát khỏi cực tiểu địa phương bằng cách thỉnh thoảng chấp nhận các bước di chuyển tồi hơn, với việc khám phá giảm dần khi nhiệt độ nguội đi.

1. **Kết quả thực nghiệm**
   1. **So sánh hiệu suất**

Mỗi thuật toán được kiểm tra 100 lần trên các bàn cờ được tạo ngẫu nhiên có kích thước 4×4 và 8×8. Bảng sau tóm tắt kết quả:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Thuật toán | Kích thước | T.gian TB (s) | Xung đột TB | Tỉ lệ T.công |
| Steepest asc. HC | 4×4 | 0.000186 | 0.73 | 38.0% |
| Stochastic HC 1 | 4×4 | 0.000166 | 0.76 | 35.0% |
| Stochastic HC 2 | 4×4 | 0.000889 | 0.81 | 32.0% |
| Simulated Annealing | 4×4 | 0.001153 | 0.07 | 93.0% |
| Steepest asc. HC | 8×8 | 0.002120 | 1.22 | 12.0% |
| Stochastic HC 1 | 8×8 | 0.003021 | 1.44 | 7.0% |
| Stochastic HC 2 | 8×8 | 0.002150 | 1.42 | 13.0% |

Nhận xét chính:

* Với bàn cờ 4×4: Simulated Annealing đạt tỉ lệ thành công 93%, vượt trội hơn đáng kể so với các biến thể leo đồi (32-38%).
* Với bàn cờ 8×8: Tất cả các thuật toán đều gặp khó khăn hơn, với tỉ lệ thành công giảm xuống còn 7-13%.
* Thời gian chạy: Simulated Annealing chậm hơn một chút nhưng tìm được nghiệm tốt hơn đáng kể.
* Xung đột trung bình: Simulated Annealing liên tục đạt được xung đột trung bình thấp hơn trên cả hai kích thước bàn cờ.
* Tác động của Random Restarts: Khi kết hợp với random restarts (tối đa 100 lần), tỉ lệ thành công tiến gần 100% cho tất cả các thuật toán.
  1. **So sánh hiệu suất**

Phân tích các mẫu hội tụ cho thấy các đặc điểm khác biệt cho từng thuật toán:

Steepest-Ascend Hill Climbing:

* Giảm đơn điệu (luôn cải thiện hoặc dừng lại)
* Cải thiện nhanh ban đầu sau đó đạt ngưỡng ở cực tiểu địa phương
* Trung bình 92% số lần chạy bị kẹt ở cực tiểu địa phương với bàn cờ 8×8

Các biến thể Stochastic Hill Climbing:

* Mẫu giảm đơn điệu tương tự
* Lựa chọn ngẫu nhiên tạo ra sự khác biệt trong các đường dẫn hội tụ
* Stochastic HC 2 cho thấy ít bước cải thiện hơn nhưng lớn hơn
* Tần suất cực tiểu địa phương: 76-90% với bàn cờ 8×8

Simulated Annealing:

* Hội tụ không đơn điệu (có thể tạm thời tăng xung đột)
* Khám phá rộng ở giai đoạn đầu, hội tụ khi nhiệt độ giảm
* Mạnh mẽ hơn trong việc tránh cực tiểu địa phương
* Đặc điểm hội tụ tổng thể tốt nhất
  1. **Phân tích khả năng mở rộng**

Kiểm tra khả năng mở rộng trên các bàn cờ kích thước n = 4, 8, 12, 16, 20 cho thấy:  
Độ phức tạp thời gian thực nghiệm:

* Steepest-Ascend HC: O(n^4.18) - Bậc ba hoặc tệ hơn
* Simulated Annealing: O(n^1.01) - Dưới bậc hai (gần như tuyến tính!)

Kết luận về khả năng mở rộng:

* Simulated Annealing mở rộng tốt hơn đáng kể cho các bài toán lớn
* Tính toán đơn giản hơn mỗi lần lặp mặc dù có nhiều lần lặp hơn
* Tốt hơn trong việc tìm nghiệm mà không cần khởi động lại
* Steepest-Ascend HC có chi phí tính toán cao
  + Đánh giá tất cả O(n²) bước di chuyển có thể mỗi lần lặp
  + Nhiều lần lặp làm tăng thêm chi phí
* Khuyến nghị cho bài toán n-Queens lớn (n > 20):
  + Simulated Annealing: Hiệu suất một lần chạy tốt nhất
  + Stochastic HC 2 với Restarts: Lựa chọn thay thế nhanh nhất
  + Tránh Steepest-Ascend HC cho các bàn cờ rất lớn

1. **GRAPH COLORING**

**1. ĐỊNH NGHĨA**

Mục tiêu: gán màu cho các đỉnh của một đồ thị sao cho không có hai đỉnh kề nhau nào có cùng màu, sử dụng một tập hợp màu cho trước.

Phương pháp: Đồ thị được tạo ra bằng cách đặt ngẫu nhiên 10 điểm (n=10) trong không gian 2 chiều, sau đó thực hiện phép đạc tam giác Delaunay (scipy.spatial.Delaunay) để xác định các cạnh nối giữa các điểm (đỉnh). Phép đạc tam giác này đảm bảo tạo ra một đồ thị phẳng.

Biểu diễn: Các điểm (đỉnh) được sắp xếp theo tọa độ x và đánh số từ 0 đến 9. Đồ thị được trực quan hóa bằng matplotlib, hiển thị các đỉnh và cạnh.

A screen shot of a computer code

AI-generated content may be incorrect.Trích xuất quan hệ kề: Từ kết quả của phép đạc tam giác, một danh sách neighbors được tạo ra. Mỗi phần tử neighbors[k] là một mảng NumPy chứa chỉ số của các đỉnh kề với đỉnh k. Ví dụ, neighbors[0] là array([1, 2, 4, 5], dtype=int32), nghĩa là đỉnh 0 kề với các đỉnh 1, 2, 4, và 5.

Biến (Variables): Là các đỉnh của đồ thị, được biểu diễn dưới dạng chuỗi: ['0', '1', ..., '9'].

Miền giá trị (Domains): Là tập hợp các màu có thể sử dụng để tô. Trong ví dụ này, miền giá trị là ['red', 'blue', 'green', 'orange'] (4 màu). Một dictionary domains được tạo, trong đó mỗi biến (đỉnh) đều có cùng miền giá trị này. (Code cũng có tùy chọn sử dụng 3 màu, hiện đang được comment lại).

Ràng buộc (Constraints): Là các ràng buộc nhị phân (binary constraints) yêu cầu hai đỉnh kề nhau phải có màu khác nhau. Các ràng buộc này được tạo từ danh sách neighbors dưới dạng các tuple (str(i), str(j)) với i < j và j là hàng xóm của i. Tập hợp các ràng buộc này được lưu trữ trong csp['constraints'].

Cấu trúc dữ liệu: Toàn bộ CSP được gói gọn trong dictionary csp bao gồm các khóa 'variables', 'domains', và 'constraints'.

**2. CÁC HÀM HỖ TRỢ CHO TÌM KIẾM QUAY LUI**

Trước khi triển khai thuật toán chính, hai hàm kiểm tra cơ bản được định nghĩa:

* complete(assignment, csp): Kiểm tra xem phép gán assignment (một dictionary) đã hoàn chỉnh hay chưa, tức là tất cả các biến trong csp['variables'] đã có trong khóa của assignment hay chưa. Kết quả cho thấy {} (phép gán rỗng) là không hoàn chỉnh, trong khi một phép gán đủ 10 biến là hoàn chỉnh.

A computer code with many different colored text

AI-generated content may be incorrect.

* A computer code with colorful text

  AI-generated content may be incorrect.consistent(assignment, csp): Kiểm tra xem phép gán hiện tại (có thể là một phần) có vi phạm ràng buộc nào không. Hàm này lặp qua tất cả các ràng buộc (var1, var2) trong csp['constraints']. Nếu cả var1 và var2 đều đã được gán giá trị trong assignment và giá trị của chúng giống nhau (assignment[var1] == assignment[var2]), thì phép gán là không nhất quán (trả về False). Phép gán rỗng {} là nhất quán. Phép gán {'0': 'red', '1': 'red', '2': 'blue'} là không nhất quán vì đỉnh 0 và 1 kề nhau và cùng màu đỏ.

**3. THUẬT TOÁN TÌM KIẾM QUAY LUI**

Thuật toán được triển khai thông qua các hàm sau:

* select\_unassigned\_var(assignment, csp): Chọn biến tiếp theo chưa được gán giá trị. Phiên bản đơn giản này chỉ chọn biến đầu tiên trong danh sách csp['variables'] mà chưa có trong assignment. Kết quả ví dụ: với assignment = {'0': 'red', '1': 'blue'}, biến tiếp theo được chọn là '2'. Lưu ý: Code có ghi chú TODO về việc cải tiến hàm này bằng heuristic Minimum-Remaining-Values (MRV), tức là ưu tiên chọn biến có ít giá trị hợp lệ còn lại nhất trong miền giá trị của nó.
* backtrack(assignment, csp): Hàm đệ quy thực hiện tìm kiếm.

1. Kiểm tra hoàn thành: Nếu assignment đã complete, trả về assignment đó (tìm thấy lời giải).
2. Chọn biến: Gọi select\_unassigned\_var để chọn var tiếp theo.
3. Lặp qua giá trị: Duyệt qua từng giá trị val trong csp['domains'][var]. Lưu ý: Code có ghi chú TODO về việc cải tiến vòng lặp này bằng heuristic Least-Constraining-Value (LCV), tức là ưu tiên thử giá trị ít gây ảnh hưởng (ít loại bỏ giá trị khả dĩ của các biến khác) nhất.
4. Gán và Kiểm tra: Gán assignment[var] = val. Lưu ý: Code có ghi chú TODO về việc thêm bước suy luận (inference) như Forward Checking hoặc Arc Consistency tại đây để phát hiện sớm các thất bại (pruning). Nếu consistent(assignment, csp) là True:

* Gọi đệ quy result = backtrack(assignment, csp).
* Nếu result không phải None (tìm thấy lời giải từ nhánh con), trả về result.

1. Quay lui (Backtrack): Nếu giá trị val không dẫn đến lời giải (hoặc không nhất quán), hủy bỏ phép gán (del assignment[var]) và thử giá trị val tiếp theo.
2. Nếu đã thử hết các giá trị cho var mà không tìm thấy lời giải, trả về None (nhánh tìm kiếm này thất bại).

* Biến toàn cục VERBOSE để bật/tắt in các bước kiểm tra, biến COUNT để đếm số nút đã duyệt.
* backtrack\_search(csp): Hàm khởi tạo, gọi backtrack với phép gán rỗng và in ra số nút đã kiểm tra (COUNT) sau khi tìm kiếm xong.

**4. KẾT QUẢ THỰC THI**

Chạy thuật toán: Thuật toán backtrack\_search được chạy trên CSP đã định nghĩa (10 đỉnh, 4 màu).

Số nút đã kiểm tra: COUNT = 11. Điều này cho thấy thuật toán tìm thấy lời giải rất nhanh chóng mà không cần quay lui nhiều với cấu hình bài toán này.

Lời giải: Phép gán màu hoàn chỉnh và nhất quán được tìm thấy là: '0': 'red', '1': 'blue', '2': 'blue', '3': 'red', '4': 'green', '5': 'orange', '6': 'red', '7': 'green', '8': 'blue', '9': 'orange'.

A network of colored circles and lines

AI-generated content may be incorrect.

Thời gian: Lệnh %timeit -n1 -r1 cho thấy thời gian thực thi là khoảng 9.27 ms (trên môi trường thử nghiệm).

Trực quan hóa: Kết quả tô màu được hiển thị lại trên đồ thị bằng matplotlib, sử dụng màu tương ứng từ lời giải res cho từng đỉnh.

1. **KẾT LUẬN**

**Tìm kiếm Quay lui (Graph Coloring):** Là thuật toán hoàn chỉnh (systematic), đảm bảo tìm ra giải pháp nếu tồn tại (trong không gian trạng thái hữu hạn). Tuy nhiên, nó có thể rất chậm đối với các bài toán lớn do phải khám phá không gian tìm kiếm theo cấp số nhân. Các kỹ thuật như heuristic chọn biến/giá trị và ràng buộc lan truyền (không được triển khai trong ví dụ cơ bản) là cần thiết để cải thiện hiệu năng.

**Tìm kiếm Cục bộ (N-Queens):** Là thuật toán không hoàn chỉnh, thường nhanh hơn quay lui nhưng không đảm bảo tìm thấy giải pháp tối ưu và dễ bị kẹt ở cực tiểu cục bộ. Các biến thể ngẫu nhiên (Stochastic HC, SA) và chiến lược meta (Random Restarts) được sử dụng để cải thiện khả năng thoát khỏi cực tiểu cục bộ và tìm giải pháp tốt hơn. SA thường cân bằng tốt giữa khám phá và khai thác, trong khi Random Restarts là cách đơn giản để tăng tỷ lệ thành công của các thuật toán leo đồi cơ bản. Khả năng mở rộng của các thuật toán tìm kiếm cục bộ phụ thuộc vào chi phí đánh giá lân cận và số bước lặp cần thiết.