GIỚI THIỆU BÀI TOÁN 8-PUZZLE

8-Puzzle là một bài toán cổ điển trong trí tuệ nhân tạo, bao gồm một bảng vuông 3x3 với 8 ô đánh số từ 1 đến 8 và một ô trống (0). Mục tiêu của bài toán là sắp xếp lại các ô sao cho chúng nằm đúng vị trí theo trạng thái mục tiêu đã định, bằng cách di chuyển các ô kề với ô trống vào vị trí của nó

1. MỤC TIÊU

Mục tiêu chính của dự án là xây dựng một phần mềm trực quan, linh hoạt để giải quyết bài toán 8-Puzzle – một trong những bài toán kinh điển trong lĩnh vực trí tuệ nhân tạo và tối ưu hóa

Dự án không chỉ giúp người dùng hiểu rõ cách các thuật toán tìm kiếm hoạt động mà còn so sánh trực tiếp hiệu suất giữa chúng thông qua số bước giải và thời gian thực thi. Giao diện trực quan cho phép hiển thị các bước giải, trạng thái trung gian và các trạng thái belief (niềm tin) trong môi trường không xác định, giúp người học dễ dàng quan sát và phân tích quá trình giải của từng thuật toán

Ngoài ra, dự án còn hướng đến mục tiêu giảng dạy và nghiên cứu, hỗ trợ sinh viên, giảng viên hoặc lập trình viên có thể thử nghiệm, mở rộng hoặc tích hợp các phương pháp AI nâng cao vào bài toán cổ điển này

2. NỘI DUNG

2.1 Nhóm thuật toán tìm kiếm không có thông tin (Uninformed Search Algorithms)

Các thành phần chính của bài toán tìm kiếm:

- Trạng thái ban đầu: Lưới 3x3 chứa 8 số từ 1 đến 8 và một ô trống (ô 0). Trong đó, trạng thái ban đầu là ([[1 2 3], [4 6 0], [7 5 8]])
- Trạng thái mục tiêu: Lưới 3x3 với trạng thái là ([[1 2 3], [4 5 6], [7 8 0]])
- Không gian trạng thái: Tập hợp tất cả các cách sắp xếp cụ thể vị trí các ô của lưới 3x3
- Hành động: Ô trống di chuyển lên, xuống, trái, phải để hoán đổi với ô liền kề dựa trên một thuật toán để tìm trạng thái đích
- Chi phí: Mỗi bước di chuyển có chi phí bằng 1

Giải pháp:

- Từ trạng thái ban đầu, tìm ra trạng thái mục tiêu từ các thuật toán tìm kiếm không có thông tin như BFS, DFS, UCS, IDS

Hình ảnh gif của từng thuật toán:

BFS


1
2
3
4
6
7
5
8
Time: 0.00 ms
Steps: 0

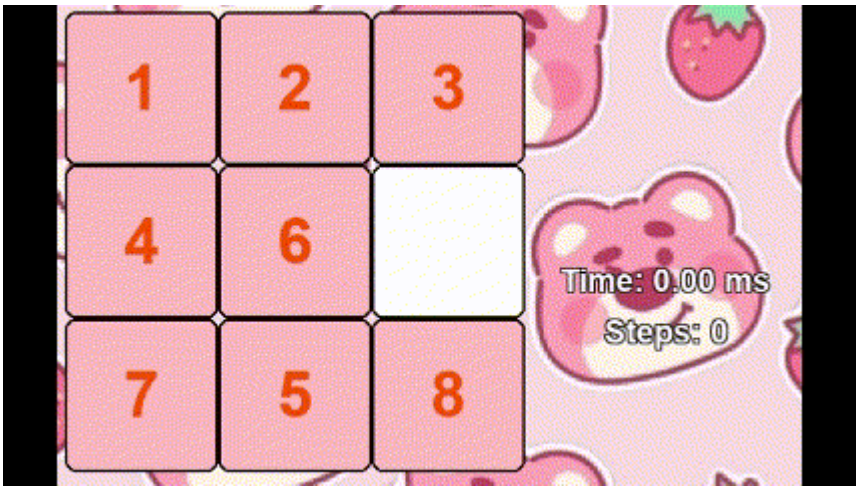
DFS


1
2
3
4
6
7
5
8
Time: 0.00 ms
Steps: 0

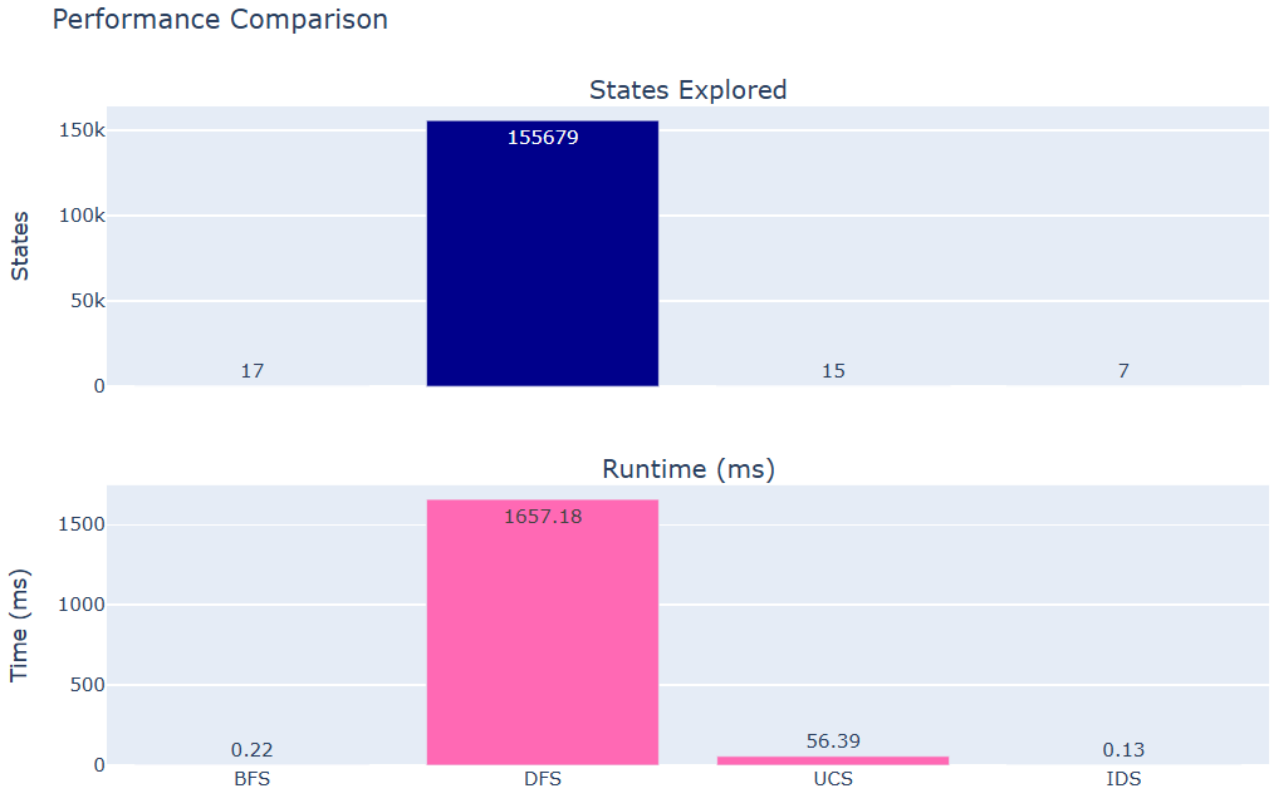
UCS


1
2
3
4
6
7
5
8
Time: 0.00 ms
Steps: 0

IDS



Hình ảnh so sánh hiệu suất của các thuật toán:



Đánh giá các thuật toán:

- Breadth-First Search (BFS): Thuật toán BFS có khả năng tìm ra lời giải ngắn nhất khi mỗi bước đi có cùng chi phí. Tuy nhiên, BFS phải lưu trữ toàn bộ các trạng thái ở từng mức độ, khiến lượng bộ nhớ tiêu tốn tăng nhanh chóng khi độ sâu tăng lên. Điều này trở thành rào cản lớn nếu không gian trạng thái quá rộng.

- Depth-First Search (DFS): DFS đi theo chiều sâu, ưu tiên khám phá hết một nhánh trước khi quay lại. Cách làm này giúp giảm đáng kể lượng bộ nhớ sử dụng hơn BFS, vì chỉ cần theo dõi một nhánh duy nhất tại một thời điểm. Tuy vậy, DFS dễ rơi vào các nhánh sâu không lối thoát, hoặc bỏ qua lời giải gần hơn, đặc biệt nếu không có cơ chế kiểm tra trạng thái đã duyệt.

- Uniform-Cost Search (UCS): UCS ưu tiên mở rộng các trạng thái có tổng chi phí thấp nhất từ đầu đến hiện tại. Trong trường hợp chi phí từng bước là như nhau, UCS hoạt động tương tự như BFS nhưng vẫn

bảo đảm tìm được lời giải tối ưu nếu tồn tại. Điểm mạnh của UCS thể hiện rõ hơn khi áp dụng vào các bài toán có trọng số không đồng đều.

- Iterative Deepening Search (IDS): IDS là sự kết hợp hiệu quả giữa BFS và DFS: nó thực hiện nhiều lần tìm kiếm theo chiều sâu với giới hạn độ sâu tăng dần. Mặc dù mỗi lần lặp lại sẽ tái duyệt lại các node đã duyệt ở độ sâu trước đó, nhưng chi phí bộ nhớ cực thấp là một lợi thế lớn.

2.2 Nhóm thuật toán tìm kiếm có thông tin (Informed Search Algorithms)

Các thành phần chính của bài toán tìm kiếm:

- Trạng thái ban đầu: Lưới 3x3 chứa 8 số từ 1 đến 8 và một ô trống (ô 0). Trong đó, trạng thái ban đầu là ([[1 2 3], [0 4 6], [7 5 8]]).
- Trạng thái mục tiêu: Lưới 3x3 với trạng thái là ([[1 2 3], [4 5 6], [7 8 0]]).
- Không gian trạng thái: Tập hợp tất cả các cách sắp xếp cụ thể vị trí các ô của lưới 3x3.
- Hành động: Ô trống di chuyển lên, xuống, trái, phải để hoán đổi với ô liền kề dựa trên một thuật toán để tìm trạng thái đích
- Chi phí: Mỗi bước di chuyển có chi phí bằng 1

Giải pháp:

- Từ trạng thái ban đầu, tìm ra trạng thái mục tiêu từ các thuật toán tìm kiếm có thông tin như Greedy Search, A*, IDA*

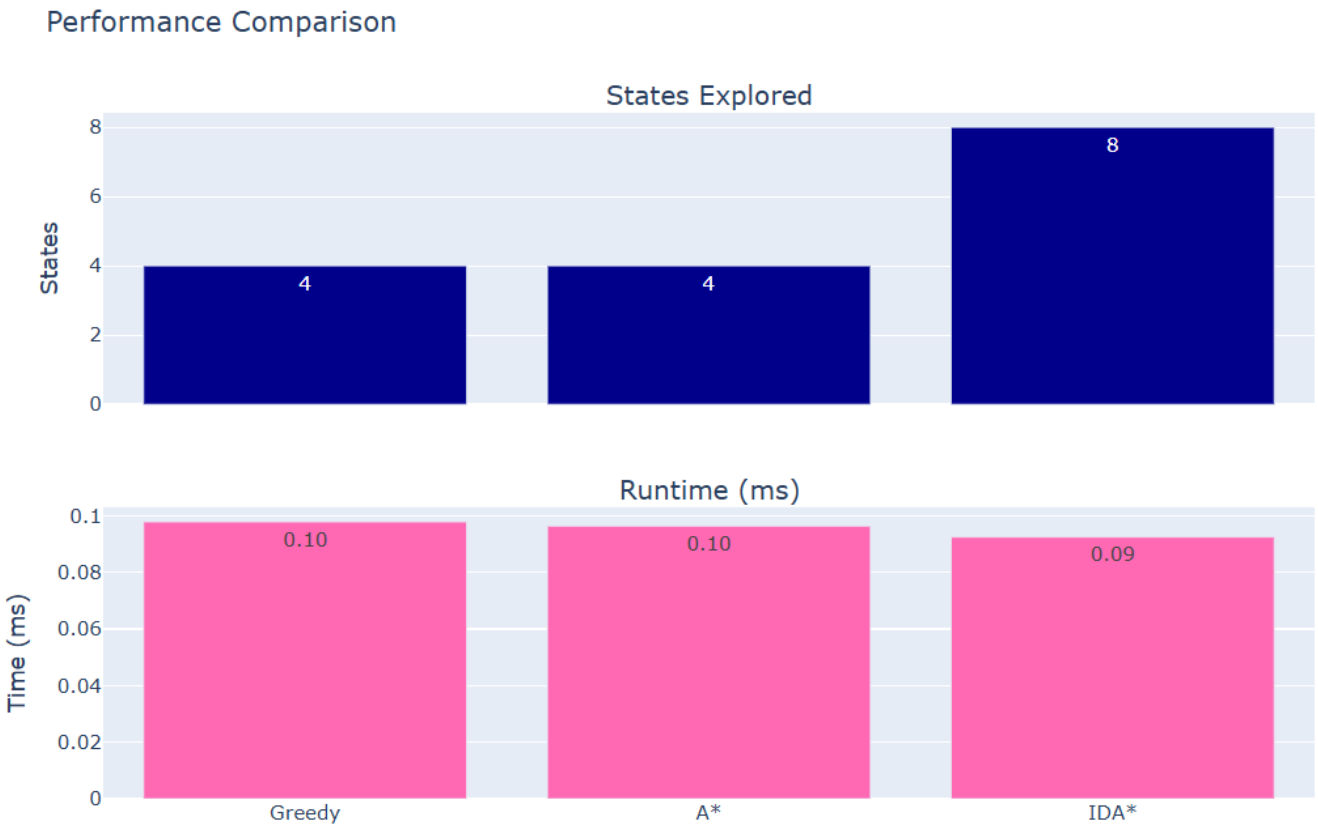
Hình ảnh gif của từng thuật toán:

Greedy Search

A*

IDA*

Hình ảnh so sánh hiệu suất của các thuật toán:



Đánh giá các thuật toán:

- Greedy Search: Greedy Search ưu tiên mở rộng những trạng thái được đánh giá là gần đích nhất dựa trên giá trị heuristic (h(n)) mà không quan tâm đến chi phí đã đi qua. Nhờ đó, Greedy Search thường có tốc độ giải nhanh và lượng trạng thái duyệt tương đối thấp. Tuy nhiên, điểm yếu lớn của nó là dễ bị rơi vào các điểm tối ưu cục bộ (local optima)
- A*: A* là thuật toán tìm kiếm sử dụng hàm đánh giá tổng hợp f(n) = g(n) + h(n), trong đó g(n) là chi phí đã đi, còn h(n) là ước lượng đến đích. Nhờ kết hợp cả hai yếu tố này, A* đảm bảo tìm ra lời giải ngắn nhất nếu heuristic là chấp nhận được (admissible). Tuy nhiên, cái giá phải trả là bộ nhớ tiêu tốn khá lớn, do cần quản lý một hàng đợi ưu tiên với rất nhiều trạng thái
- IDA*: IDA* kế thừa tính đúng đắn của A* và tính tiết kiệm bộ nhớ của DFS. Thay vì giữ tất cả trạng thái trong hàng đợi, IDA* lặp lại tìm kiếm theo độ sâu giới hạn, mỗi lần tăng giới hạn theo giá trị f(n). Điều này giúp giảm áp lực bộ nhớ đáng kể, thích hợp cho các thiết bị hạn chế tài nguyên. Tuy nhiên, nhược điểm là IDA* phải duyệt lại nhiều trạng thái qua từng vòng lặp, dẫn đến tổng số trạng thái duyệt cao

2.3 Nhóm thuật toán tìm kiếm cục bộ (Local Optimization Algorithms)

Các thành phần chính của bài toán tìm kiếm:

- Trạng thái ban đầu: Lưới 3x3 chứa 8 số từ 1 đến 8 và một ô trống (ô 0). Trong đó, trạng thái ban đầu là ([[1 2 3], [7 4 6], [5 0 8]]).
- Trạng thái mục tiêu: Lưới 3x3 với trạng thái là ([[1 2 3], [4 5 6], [7 8 0]]).
- Không gian trạng thái: Tập hợp tất cả các cách sắp xếp cụ thể vị trí các ô của lưới 3x3.
- Hành động: Ô trống di chuyển lên, xuống, trái, phải để hoán đổi với ô liền kề dựa trên một thuật toán để tìm trạng thái đích
- Chi phí: Mỗi bước di chuyển có chi phí bằng 1

Giải pháp:

- Từ trạng thái ban đầu, tìm ra trạng thái mục tiêu từ các thuật toán tìm kiếm cục bộ như Simple Hill Climbing, Steepest Hill Climbing, Random Hill Climbing, Simulated Annealing, Beam Search, Genetic Algorithm

Hình ảnh gif của từng thuật toán:

Simple Hill Climbing

Steepest Hill Climbing

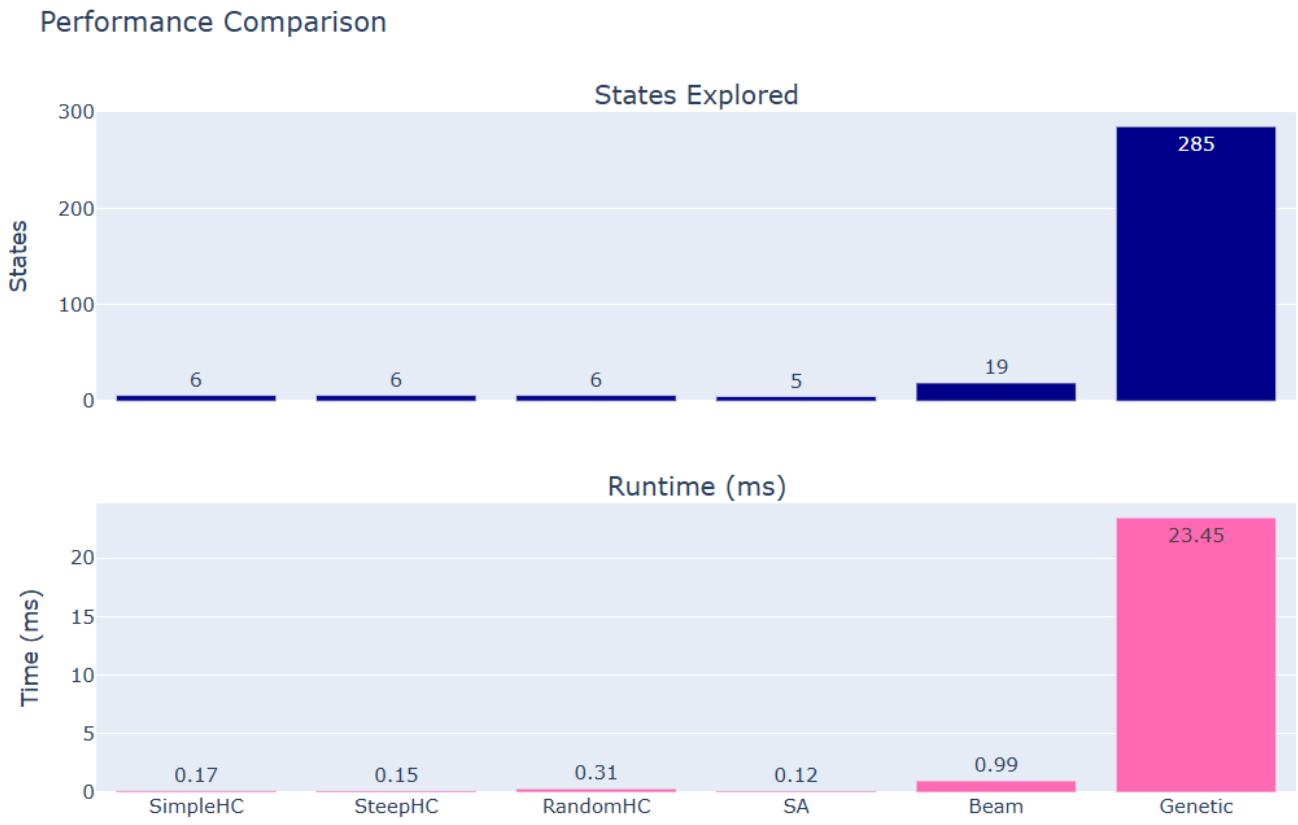
Random Hill Climbing

Simulated Annealing

Beam Search

Genetic Algorithm

Hình ảnh so sánh hiệu suất của các thuật toán:



Đánh giá các thuật toán:

- Simple Hill Climbing: SHC hoạt động theo nguyên tắc đơn giản: từ trạng thái hiện tại, nếu tìm thấy trạng thái lân cận nào tốt hơn, thuật toán chuyển ngay sang đó và tiếp tục. Vì chỉ xét một lựa chọn đầu tiên thỏa mãn, SHC có thời gian chạy rất nhanh và tốn ít tài nguyên. Tuy nhiên, chính sự đơn giản này khiến SHC dễ bị mắc kẹt tại các điểm cực trị cục bộ mà không có cơ chế nào để thoát ra, dẫn đến bỏ lỡ lời giải tốt hơn trong không gian trạng thái rộng lớn
- Steepest Hill Climbing: Khác với SHC, Steepest HC không "dừng chân" với trạng thái tốt đầu tiên mà đánh giá toàn bộ các trạng thái lân cận, sau đó chọn trạng thái có giá trị tốt nhất để tiếp tục. Nhờ vậy, Steepest HC có khả năng tìm ra những đường đi cải thiện tốt hơn, nhưng phải trả giá bằng thời gian xử lý cao hơn và số trạng thái duyệt nhiều hơn SHC
- Random Hill Climbing: RHC dựa vào yếu tố ngẫu nhiên khi chọn trạng thái kế tiếp từ danh sách các lựa chọn tốt hơn hiện tại. Việc không cần duyệt toàn bộ như Steepest HC, cũng không dừng sớm như SHC, khiến thời gian chạy và hiệu suất RHC nằm ở mức trung bình giữa hai thuật toán trên. Yếu tố ngẫu nhiên giúp RHC đôi khi "vượt qua" được một số cực trị cục bộ nhỏ, nhưng vẫn không có cơ chế thoát rõ ràng khỏi các điểm bế tắc lớn
- Simulated Annealing: SA là phiên bản cải tiến của Hill Climbing khi cho phép di chuyển đến trạng thái kém hơn với xác suất giảm dần theo thời gian – mô phỏng quá trình làm nguội trong luyện kim. Cách tiếp cận này giúp SA tránh bị mắc kẹt tại các điểm cực trị cục bộ, với khả năng khám phá rộng hơn các thuật toán Hill Climbing. Tuy nhiên, SA cần thời gian chạy dài hơn để "làm nguội", và lượng trạng thái duyệt thường nhiều hơn đáng kể, đặc biệt khi nhiệt độ giảm chậm
- Beam Search: Beam Search là phương pháp tìm kiếm theo hướng heuristic có kiểm soát, chỉ giữ lại một số trạng thái tốt nhất ở mỗi mức độ (beam width). Với beam width vừa phải (ví dụ = 3), Beam Search cân bằng được giữa tốc độ xử lý và chất lượng kết quả. Số lượng trạng thái duyệt nhiều hơn Hill Climbing, nhưng ít hơn SA hay GA. Nhờ giới hạn trạng thái mở rộng, Beam Search có thời gian xử lý khá ổn định và không quá phụ thuộc vào độ sâu hay độ rộng của không gian trạng thái
- Genetic Algorithm: GA mô phỏng quá trình tiến hóa sinh học: duy trì một quần thể trạng thái, tạo thế hệ mới thông qua lai ghép và đột biến, rồi chọn lọc các cá thể tốt hơn. Thuật toán này rất mạnh về khả năng tìm kiếm toàn cục và có thể tiếp cận lời giải tối ưu, nhưng đánh đổi bằng thời gian xử lý lâu và số lượng trạng thái duyệt rất lớn. GA thường phù hợp khi không gian trạng thái phức tạp, hoặc các thuật toán khác dễ bị mắc kẹt cục bộ

2.4 Nhóm thuật toán tìm kiếm trong môi trường phức tạp (Search in complex environments)

Các thành phần chính của bài toán tìm kiếm:

- Trạng thái ban đầu: AND-OR Search: Xuất phát từ một cấu hình xác định của lưới 3x3, trong đó chứa 8 ô số từ 1 đến 8 và một ô trống (ô 0). Đây là trạng thái gốc để thuật toán khởi động hành trình tìm kiếm, với [[1, 2, 3], [4, 0, 6], [7, 5, 8]]

  Belief State Search: Thay vì một trạng thái đơn, thuật toán này khởi đầu với một tập hợp các trạng thái có thể xảy ra (belief states), mô phỏng sự không chắc chắn trong môi trường. Các belief states ban đầu thường bao gồm trạng thái thực tế cùng với các trạng thái suy đoán được tạo bằng cách hoán đổi ô trống với các ô số lân cận theo luật di chuyển hợp lệ

  Partial Observable Search: Xuất phát điểm là tập hợp các trạng thái có thể có, dựa trên thông tin quan sát không đầy đủ. Ví dụ: nếu chỉ biết ô số 1 nằm ở vị trí (1,1), thuật toán xây dựng tập belief states gồm

tất cả trạng thái 3x3 hợp lệ mà thỏa mãn điều kiện đó

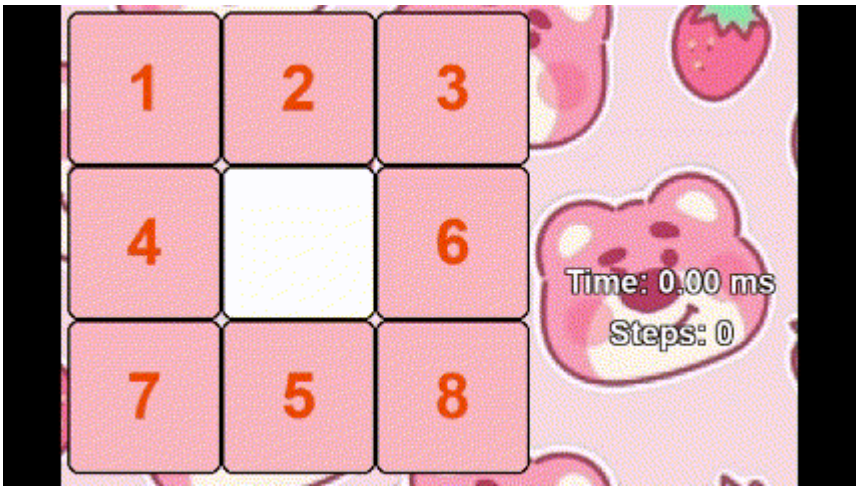
- Trạng thái mục tiêu: Lưới 3x3 với trạng thái là ([[1 2 3], [4 5 6], [7 8 0]]).
- Không gian trạng thái: Tập hợp tất cả các cách sắp xếp cụ thể vị trí các ô của lưới 3x3.Trong trường hợp của Belief State và POS, không gian còn bao gồm các tập trạng thái không xác định – nơi một hành động có thể áp dụng lên nhiều trạng thái cùng lúc
- Hành động: Ô trống di chuyển lên, xuống, trái, phải để hoán đổi với ô liền kề dựa trên một thuật toán để tìm trạng thái đích
- Chi phí: Mỗi bước di chuyển có chi phí bằng 1

Giải pháp:

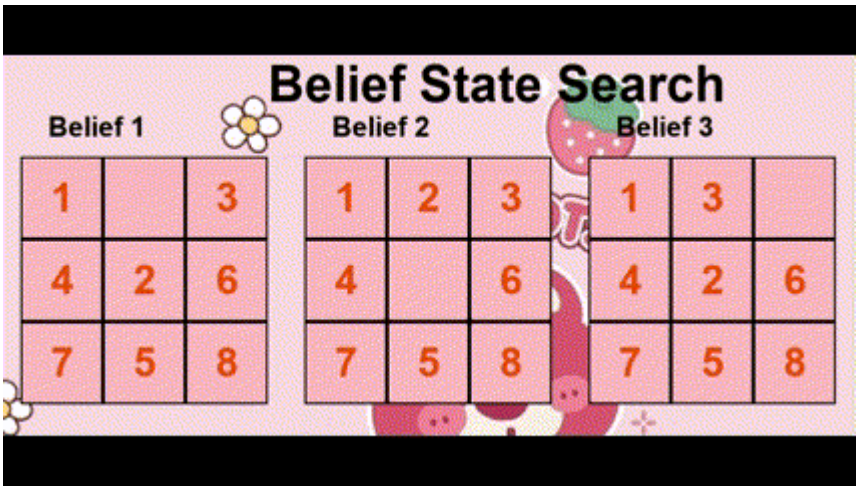
- Một lời giải là dãy hành động hoặc dãy trạng thái dẫn từ trạng thái khởi đầu đến trạng thái mục tiêu.
- Với thuật toán AND-OR Search, lời giải là cây tìm kiếm (search tree) với các nút "AND" và "OR" phản ánh sự rẽ nhánh trong hành động hoặc kết quả
- Với Belief State Search và POS, lời giải là một dãy hành động chung sao cho, khi áp dụng lên toàn bộ các trạng thái trong belief state ban đầu, mọi trạng thái kết thúc đều đạt được trạng thái mục tiêu

Hình ảnh gif của từng thuật toán:

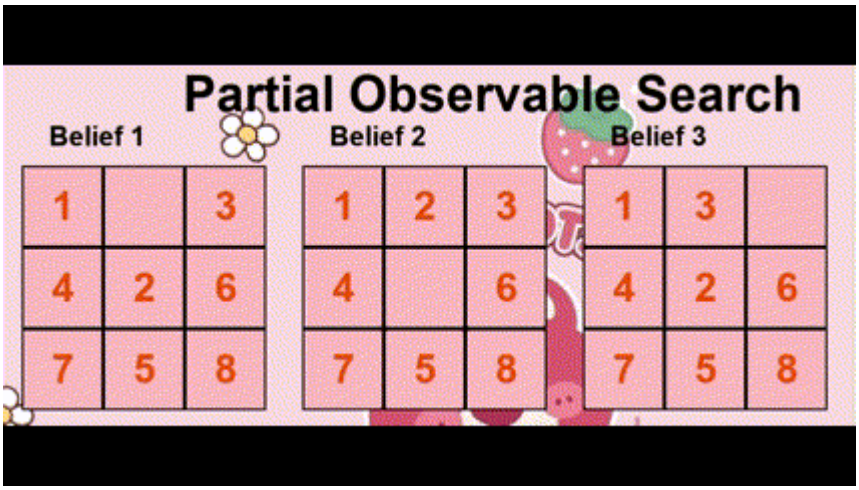
And-Or Search



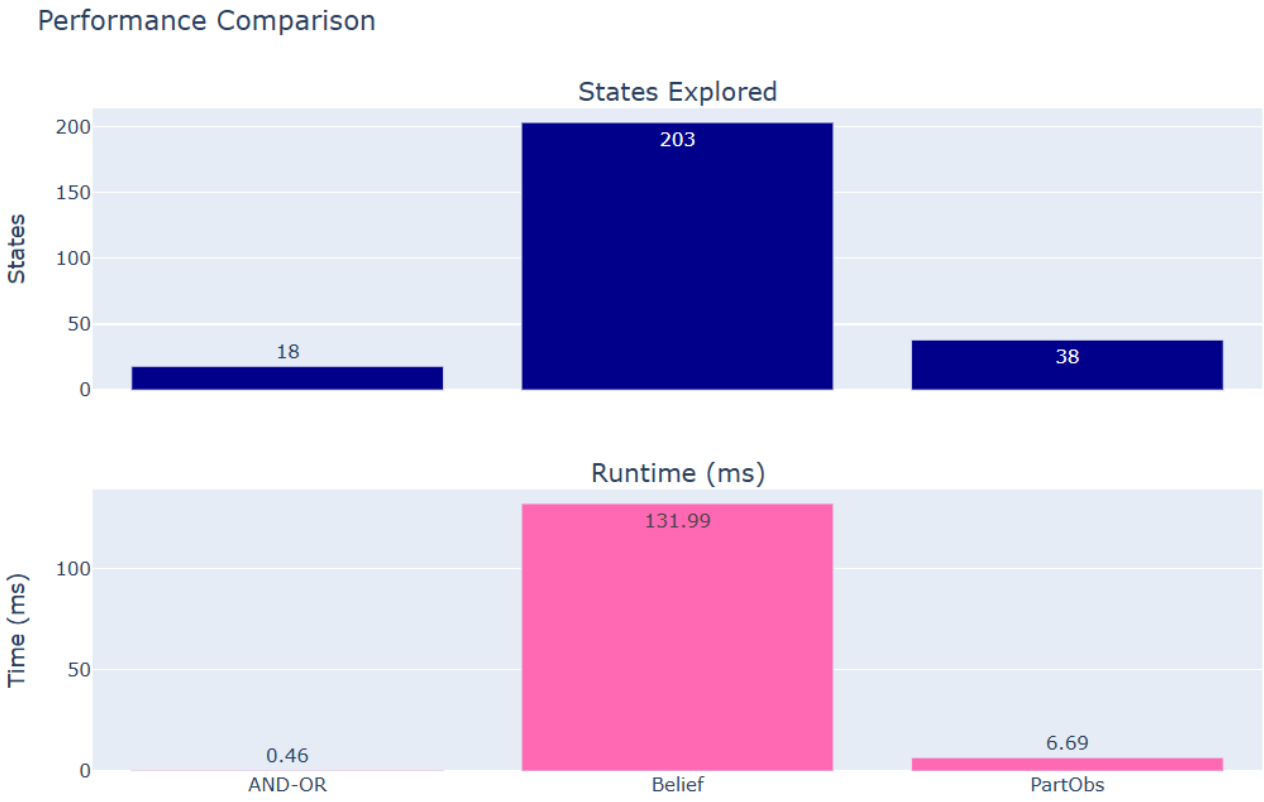
Belief State Search



Partial Observable Search



Hình ảnh so sánh hiệu suất của các thuật toán:



Đánh giá các thuật toán:

- And-Or Search: Thuật toán And-Or Search được thiết kế để hoạt động trong môi trường mà kết quả của hành động có thể không chắc chắn. Tại mỗi bước, nó không sử dụng hàm heuristic để định hướng lựa chọn hành động. Mỗi hành động có thể dẫn đến nhiều kết quả khả dĩ, dẫn đến việc xây dựng cây tìm kiếm AND-OR có kích thước rất lớn. Nhánh AND không bị rút gọn, nên không gian trạng thái là lớn nhất trong ba thuật toán. Tuy nhiên, chi phí tính toán mỗi trạng thái thấp, vì chỉ bao gồm các thao tác đơn giản như kiểm tra trạng thái mục tiêu, thực hiện hoán đổi ô trống, và không gọi các phép đánh giá nâng cao. Do đó, thời gian chạy tổng thể thường thấp, mặc dù số lượng trạng thái cần xử lý nhiều.
- Belief State Search: Trong môi trường không thể quan sát trực tiếp, thuật toán khởi động với tập belief state, đại diện cho tập hợp các trạng thái có thể có. Tại mỗi bước, thuật toán sử dụng hàm heuristic để đánh giá và rút gọn tập belief state, chỉ giữ lại 3 trạng thái tiềm năng nhất, giúp giới hạn không gian tìm kiếm. Tuy nhiên, việc đánh giá liên tục bằng heuristic trên nhiều trạng thái gây ra chi phí xử lý cao cho mỗi bước, dẫn đến thời gian chạy tổng thể dài hơn so với AND-OR Search. Không gian trạng thái giảm đáng kể so với AND-OR nhờ ràng buộc belief state, nhưng vẫn khá lớn do thiếu thông tin quan sát hỗ trợ
- Partial Observable Search: POS là phiên bản cải tiến khi hệ thống có thể quan sát được một phần thông tin (ví dụ: xác định số 1 nằm ở vị trí (1,1)). Tập belief state ban đầu được xây dựng dựa trên điều kiện quan sát này, giúp loại bỏ ngay các trạng thái không phù hợp, làm thu hẹp không gian trạng thái ngay từ đầu. Tương tự Belief State Search, thuật toán giữ lại 3 trạng thái tốt nhất tại mỗi bước theo đánh giá heuristic, nhưng nhờ có thêm thông tin từ quan sát nên việc định hướng trở nên chính xác hơn. Thời gian chạy trung bình vì dù chi phí xử lý mỗi trạng thái vẫn cao, nhưng số trạng thái cần xử lý ít hơn, và số bước trung bình cũng được rút ngắn. POS thể hiện sự cân bằng hiệu quả giữa độ phức tạp và độ chính xác, tận dụng triệt để thông tin quan sát để tập trung vào các hướng đi có khả năng cao đạt mục tiêu

2.5 Nhóm thuật toán tìm kiếm thỏa ràng buộc (Constraint Satisfaction Problem)

Các thành phần chính của bài toán tìm kiếm:

- Trạng thái ban đầu: Bài toán bắt đầu với một lưới 3x3 trống, trong đó mỗi ô chưa có giá trị xác định và được biểu diễn bằng None: [[None, None, None], [None, None, None], [None, None, None]]
- Trạng thái mục tiêu: Mục tiêu là đạt được một cấu hình sắp xếp hợp lệ, có dạng: [[1, 2, 3], [4, 5, 6], [7, 8, 0]]
- Không gian trạng thái: Là tập hợp tất cả các hoán vị hợp lệ của các số từ 0 đến 8 trên lưới 3x3, với các ràng buộc như: - Ràng buộc vị trí cố định: ô (0, 0) phải chứa số 1 theo yêu cầu bài toán - Ràng buộc duy nhất: mỗi số từ 0 đến 8 chỉ xuất hiện một lần duy nhất - Ràng buộc thứ tự hàng ngang: với mọi ô chứa số khác 0, nếu tồn tại ô bên phải (i, j+1), thì grid[i][j+1] = grid[i][j] + 1 - Ràng buộc thứ tự cột dọc: tương tự, nếu tồn tại ô phía dưới (i+1, j), thì grid[i+1][j] = grid[i][j] + 3 - Ràng buộc về khả năng giải được: cấu hình chỉ hợp lệ nếu tổng số cặp nghịch đảo (inversions) là số chẵn, đảm bảo puzzle có lời giải.
- Hành động:
    - Backtracking Search: Duyệt từng ô theo thứ tự, gán giá trị nếu hợp lệ. Quay lui nếu không còn giá trị hợp lệ trong domain.
    - Forward Checking: Tương tự Backtracking, nhưng sau mỗi lần gán sẽ cập nhật domain của các ô còn lại bằng cách loại bỏ các giá trị không còn khả thi. Giúp phát hiện xung đột sớm và tránh rơi

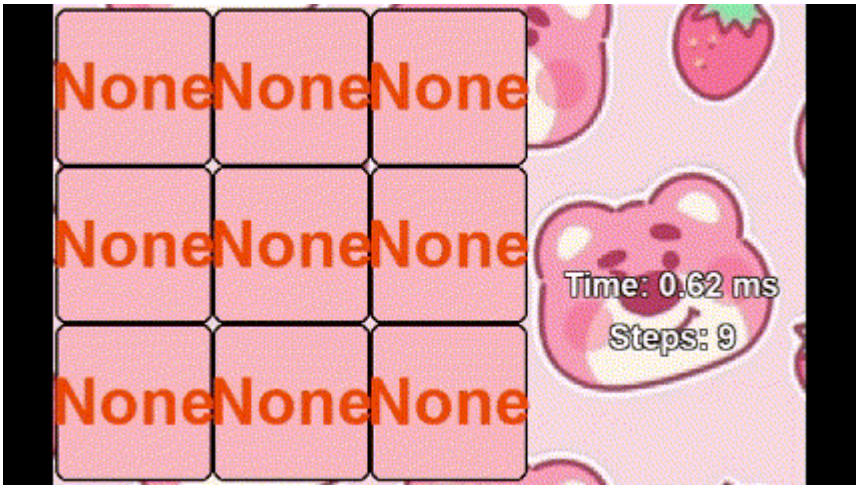
vào nhánh sai
    - Min-Conflicts Search: Khởi đầu với trạng thái đầy đủ, nhưng có thể không hợp lệ (các giá trị được gán ngẫu nhiên). Tìm ô đang gây xung đột ràng buộc, gán lại giá trị sao cho số xung đột giảm nhiều nhất. Lặp lại quá trình sửa lỗi cho đến khi đạt trạng thái thỏa mãn hoặc vượt giới hạn lặp
- Chi phí: Chỉ tập trung vào tìm kiếm trạng thái cuối cùng hợp lệ. Mỗi hành động gán hợp lệ đều có trọng số bằng nhau, nên chi phí không được sử dụng để hướng dẫn tìm kiếm như trong các bài toán tối ưu

Giải pháp:

- Một dãy các bước gán giá trị bắt đầu từ trạng thái trống đến khi hoàn chỉnh lưới 3x3.
- Mỗi chiến lược giải sẽ cho ra một chuỗi trạng thái trung gian khác nhau, nhưng tất cả đều kết thúc bằng trạng thái hợp lệ duy nhất, phù hợp với ràng buộc bài toán.
- Các thuật toán như Backtracking, Forward Checking, và Min-Conflicts cung cấp các cách tiếp cận khác nhau để xử lý không gian ràng buộc, từ duyệt toàn bộ đến tìm kiếm cục bộ có điều chỉnh lỗi.

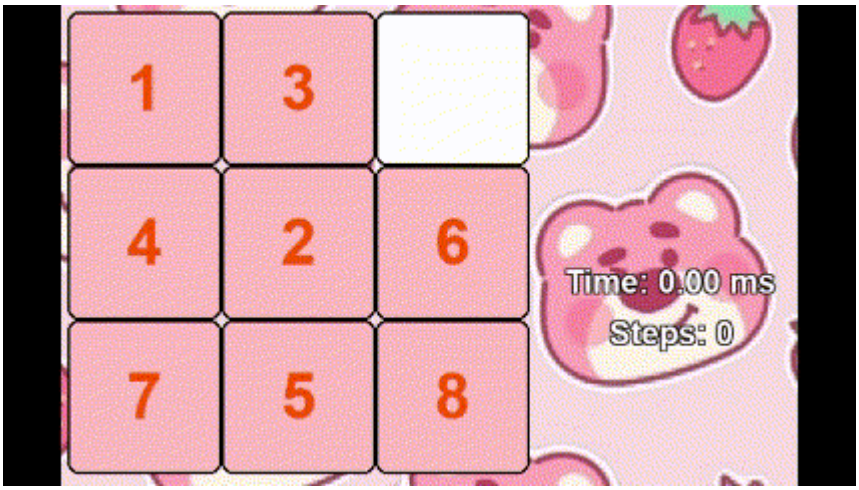
Hình ảnh gif của từng thuật toán:

Backtracking Search

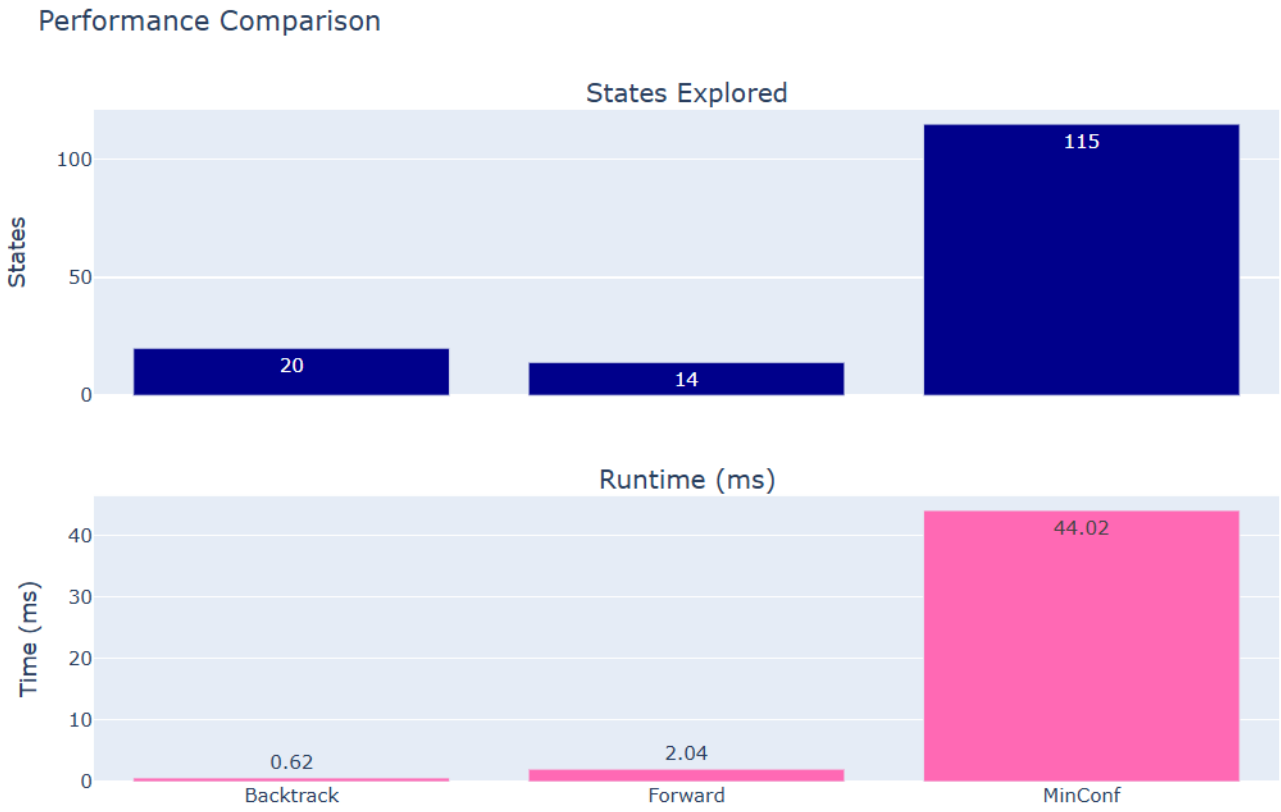


Forward Checking

Min-Conflicts Search



Hình ảnh so sánh hiệu suất của các thuật toán:



Đánh giá các thuật toán:

- Backtracking Search: Thuật toán duyệt theo chiều sâu, khởi đầu từ trạng thái rỗng và tiến hành gán giá trị cho từng ô trong lưới 3x3. Mỗi khi một giá trị được gán, thuật toán kiểm tra ràng buộc ngay lập tức. Nếu phát hiện xung đột, nó sẽ quay lui và thử giá trị khác. Thuật toán không sử dụng thông tin định hướng nào nên khám phá toàn bộ không gian tìm kiếm, phù hợp khi ràng buộc đơn giản hoặc cần liệt kê mọi khả năng. Tuy nhiên, đối với bài toán nhiều biến, ràng buộc phức tạp như 8-Puzzle, số trạng thái duyệt có thể rất lớn do không có cơ chế loại trừ sớm
- Forward Checking Search: Bắt đầu từ trạng thái rỗng và sử dụng Forward Checking để loại bỏ trước các giá trị không hợp lệ khỏi miền (domain) của các biến chưa gán, sau mỗi lần gán giá trị. Thuật toán kết hợp hai chiến lược chọn biến và giá trị:
    - MRV (Minimum Remaining Values): ưu tiên gán cho ô có ít lựa chọn hợp lệ nhất.
    - LCV (Least Constraining Value): chọn giá trị ít làm giảm khả năng gán hợp lệ cho các ô còn lại.

Nhờ vào cơ chế thu hẹp không gian tìm kiếm từ sớm, số trạng thái được mở thấp hơn so với Backtracking. Đây là chiến lược có tính định hướng, giúp tăng hiệu quả khi số ràng buộc nhiều hoặc cấu trúc lời giải chặt chẽ

- Min-Conflicts Search: Thuật toán tìm kiếm cục bộ (local search) bắt đầu từ một trạng thái đầy đủ, tức là mọi ô đã được gán giá trị từ 0 đến 8 một cách ngẫu nhiên. Sau đó, thuật toán đánh giá mức độ xung đột (vi phạm ràng buộc) và chọn biến gây xung đột để gán lại giá trị sao cho giảm xung đột tổng thể.

Quá trình này được lặp lại nhiều lần, có thể kết hợp với Simulated Annealing để chấp nhận tạm thời các bước xấu, giúp tránh rơi vào cực trị địa phương. Min-Conflicts không duyệt theo cây trạng thái mà điều chỉnh trực tiếp trên lời giải hiện tại. Nhờ vậy, số lần cập nhật trạng thái thấp, nhưng mỗi lần cần tính toán toàn cục số xung đột, dẫn đến chi phí xử lý cao hơn. Thuật toán đặc biệt phù hợp với bài toán có ràng buộc mềm, hoặc yêu cầu tìm nghiệm nhanh trong không gian lớn.

2.6 Nhóm thuật toán học tăng cường (Reinforcement Learning)

Các thành phần chính của bài toán tìm kiếm:

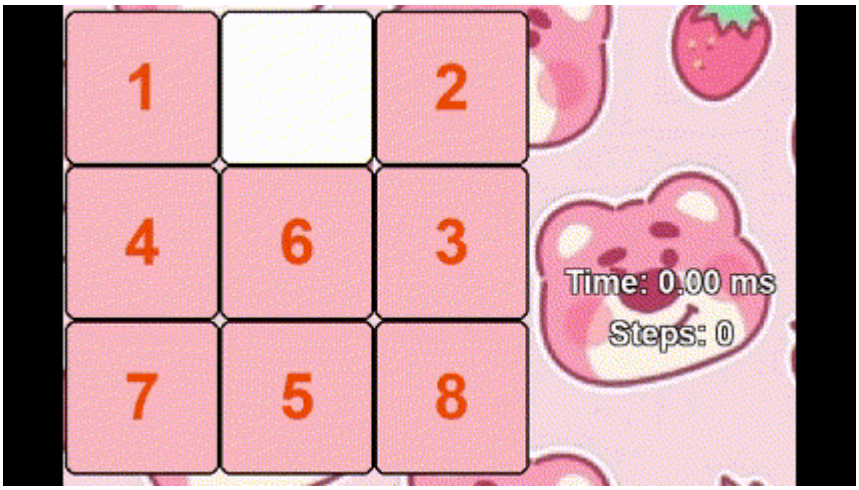
- Trạng thái ban đầu: Lưới 3x3 bao gồm 8 số nguyên từ 1 đến 8 và một ô trống được biểu diễn bằng số 0, thể hiện cấu hình ban đầu của bài toán. Ví dụ: [[1, 0, 2], [4, 6, 3], [7, 5, 8]].
- Trạng thái mục tiêu: Mục tiêu là đạt được một cấu hình sắp xếp hợp lệ, có dạng: [[1, 2, 3], [4, 5, 6], [7, 8, 0]]
- Không gian trạng thái: Tập hợp tất cả các cách sắp xếp cụ thể vị trí các ô của lưới 3x3. Q-Learning không xây dựng toàn bộ không gian này trước mà dần dần khám phá nó thông qua tương tác, cập nhật chính sách dựa trên trải nghiệm
- Hành động: Ô trống di chuyển lên, xuống, trái, phải để hoán đổi với ô liền kề dựa trên một thuật toán để tìm trạng thái đích
- Chi phí: Thay vì tính chi phí theo số bước hoặc khoảng cách, thuật toán Q-Learning sử dụng phần thưởng (reward) để hướng dẫn agent:
    - Mỗi hành động di chuyển thông thường bị phạt bằng một phần thưởng âm nhỏ (ví dụ: -1), để khuyến khích agent tìm đường đi ngắn.
    - Khi đạt trạng thái mục tiêu, agent nhận được phần thưởng dương lớn (ví dụ: +100), củng cố chính sách chọn đường đi hiệu quả.
    - Hệ thống phần thưởng này giúp tối ưu tổng phần thưởng tích lũy chứ không chỉ số bước đi.

Giải pháp:

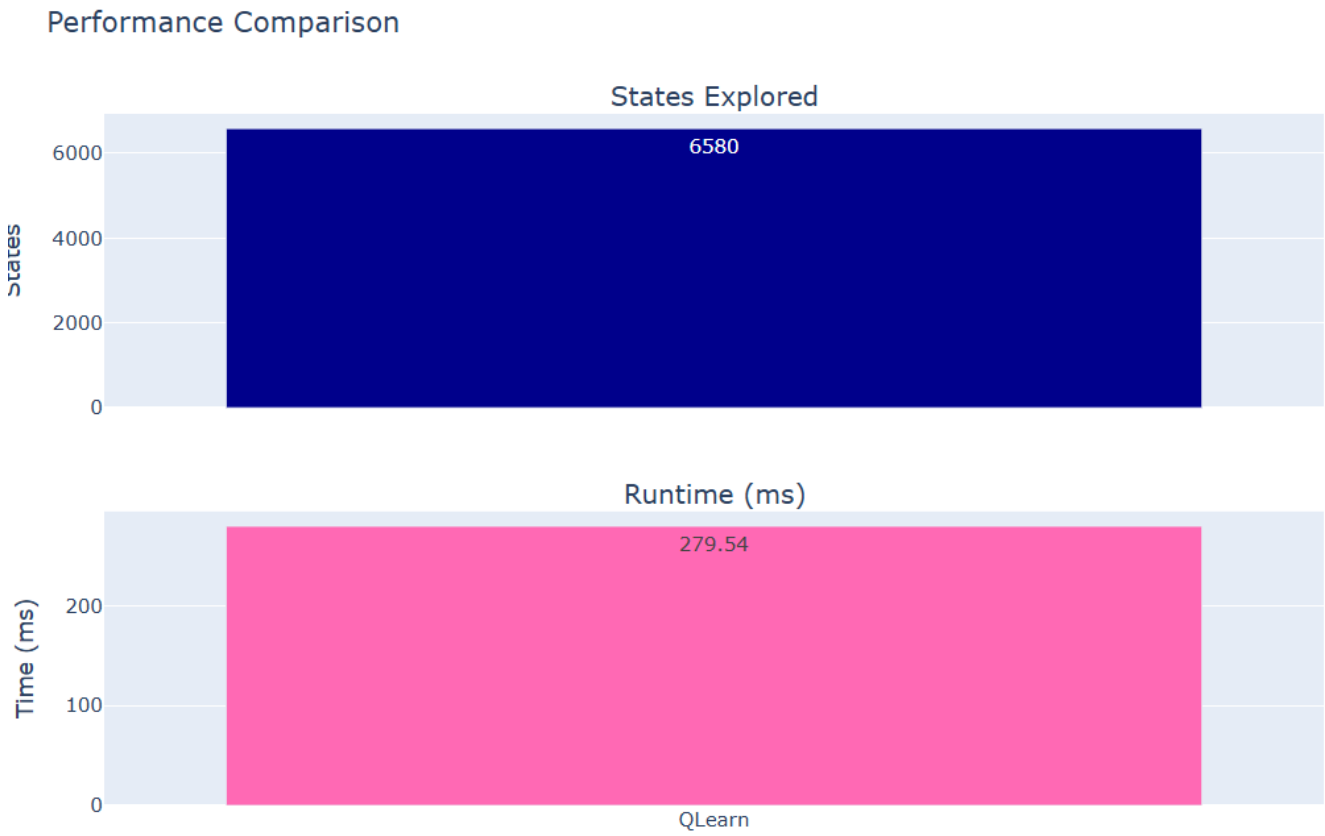
- Chuỗi trạng thái từ trạng thái khởi đầu đến trạng thái mục tiêu, được sinh ra từ chính sách đã học. Q-Learning sử dụng bảng Q (Q-table) để lưu giá trị kỳ vọng của mỗi hành động tại mỗi trạng thái. Trong quá trình học, bảng Q được cập nhật dựa trên công thức Bellman, và sau khi hội tụ, agent sẽ truy xuất đường đi tối ưu bằng cách chọn hành động có Q-value cao nhất tại mỗi bước.

Hình ảnh gif của thuật toán:

Q-Learning



Hình ảnh hiệu suất của thuật toán:



Đánh giá thuật toán:

- Q-Learning là thuật toán học tăng cường không mô hình, cho phép agent học cách hành động thông qua thử nghiệm và cập nhật bảng Q (Q-table) dựa trên phần thưởng. Thuật toán sử dụng chiến lược Epsilon-Greedy để cân bằng giữa khám phá hành động mới và khai thác hành động tốt nhất.
- Tại mỗi bước, agent di chuyển, nhận phần thưởng, cập nhật Q-value và tiếp tục học. Số trạng thái được khám phá lớn vì agent cần trải nghiệm nhiều tình huống để xây dựng chính sách tối ưu.
- Thời gian học tương đối cao do phải tính phần thưởng, tìm hành động hợp lệ và cập nhật Q-value liên tục. Tuy nhiên, nếu được huấn luyện đủ lâu, Q-Learning đảm bảo hội tụ về một chính sách tối ưu, phù hợp với môi trường không xác định và yêu cầu học dài hạn

3. Kết luận

- Dự án đã triển khai thành công một bộ giải 8-puzzle sử dụng nhiều thuật toán tìm kiếm khác nhau, bao gồm BFS, DFS, UCS, IDS, Greedy, A*, IDA*, cùng với các phương pháp tối ưu hóa như Hill Climbing, Simulated Annealing, Beam Search, Genetic Algorithm, AND-OR Search, Belief State Search, Partial Observable Search, Backtracking, Forward Checking, Min-Conflicts, và Q-Learning
- Giao diện Pygame tích hợp cho phép người dùng tùy chỉnh trạng thái ban đầu và trực quan hóa quá trình giải, tăng tính tương tác và hiệu quả học tập
- Phát triển chế độ học thuật (Learning Mode): Thêm tooltip giải thích ngắn gọn cho từng thuật toán khi rê chuột qua. Có file lưu lại các thông tin tìm đường đi của thuật toán
- Nhóm 1: BFS, DFS, UCS, IDS
    - BFS: Tìm lời giải ngắn nhất theo số bước nhưng ngốn bộ nhớ lớn
    - DFS: Bộ nhớ ít, dễ rơi vào vòng lặp hoặc nhánh sai nếu không cắt tỉa
    - UCS: Tối ưu chi phí thực, hoạt động tốt khi không có heuristic
    - IDS: Kết hợp ưu điểm BFS và DFS, tiết kiệm bộ nhớ, nhưng trùng lặp thao tác
- Nhóm 2: Greedy Search, A*, IDA*
    - Greedy Search: Chạy nhanh, nhưng dễ chọn sai hướng nếu heuristic yếu
    - A*: Cân bằng giữa chi phí và heuristic, tìm đường đi tối ưu nếu heuristic tốt
    - IDA*: Giảm bộ nhớ so với A*, nhưng mất thời gian do lặp lại nhiều tầng
- Nhóm 3: SHC, Stee pestHC, RHC, SA, Beam Search, Genetic Algorithm
    - SHC: Đơn giản, nhanh, nhưng dễ kẹt tại cực trị cục bộ
    - Steepest HC: Xét nhiều hướng, cải thiện chất lượng lời giải hơn SHC
    - RHC: Thử nhiều khởi đầu khác nhau, tăng khả năng thoát cực trị
    - Simulated Annealing: Chấp nhận tạm thời bước xấu để khám phá rộng hơn
    - Beam Search: Nhanh, tiết kiệm bộ nhớ, dễ bỏ lỡ lời giải tốt nếu beam nhỏ
    - Genetic Algorithm: Tìm lời giải chất lượng cao qua tiến hóa, tốn thời gian và tài nguyên
- Nhóm 4: AND-OR Search, Belief State Search, Partial Observable Search
    - AND-OR Search: Tìm cây chiến lược cho các tình huống có nhiều phản hồi, phù hợp môi trường không xác định
    - Belief State Search: Đại diện nhiều trạng thái niềm tin, chính xác nhưng nặng tính toán
    - Partial Observable Search (POS): Kết hợp quan sát với belief state, hiệu quả hơn trong môi trường mù mờ
- Nhóm 5: Backtracking, Forward Checking, Min-Conflicts
    - Backtracking: Duyệt sâu theo ràng buộc, đơn giản nhưng tốn thời gian
    - Forward Checking: Loại trừ sớm giá trị không hợp lệ, giảm không gian tìm kiếm
    - Min-Conflicts: Gán toàn bộ trước rồi sửa lỗi dần, phù hợp cho bài toán ràng buộc lớn
- Nhóm 6: Q-Learning
    - Q-Learning: Học chính sách tối ưu qua thử-sai, cần thời gian huấn luyện dài, nhưng mạnh mẽ trong môi trường động và không xác định

4.Sinh viên thực hiện:

Dự án được thực hiện bởi: Dương Quỳnh Như - 23110281

5. Tài liệu tham khảo:

Russell, S. J., & Norvig, P. (2021). Artificial Intelligence: A Modern Approach (4th ed.). Pearson.

Nilsson, N. J. (1980). Principles of Artificial Intelligence. Morgan Kaufmann.

Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning: An Introduction (2nd ed.). MIT Press.

Koenig, S., & Likhachev, M. (2005). Fast Replanning for Navigation in Unknown Terrain. IEEE Transactions on Robotics, 21(3), 354–363.

Tham khảo, sửa lỗi dựa trên một số AI như: ChatGPT, Grok3.