

TrieuHieu842 /
8-puzzle-with-6-group-Algorithms

<> Code

Issues

Pull requests

Actions

Projects

Wiki

Security

Insights

Setting

This project solves the 8-Puzzle problem.

☆ 0 stars

🔗 0 forks

👁 1 watching

🌿 Branches

📈 Activity

🏷 Tags

🌐 Public repository

🔗 1 Branch

🏷 0 Tags

🔗

📄

🔍 Go to file

t

Go to file

+

Add file

<> Code

...

TrieuHieu842

Update README.md

0354308 · 1 minute ago

📄	23110217_TrieuPhucHieu_Re...	them code va bao cao chinh	2 days ago
📄	CE.gif	them hình 3 nhóm cuối	2 days ago
📄	CSPs.gif	them hình 3 nhóm cuối	2 days ago
📄	ISS_1.gif	Hình ảnh các nhóm thuật toán	4 days ago
📄	LCL_1111.gif	Hình ảnh các nhóm thuật toán	4 days ago
📄	LCL_2.gif	Hình ảnh các nhóm thuật toán	4 days ago
📄	README.md	Update README.md	1 minute ago
📄	UNFS_2.gif	Hình ảnh các nhóm thuật toán	4 days ago
📄	main.py	Cập nhật file code	12 minutes ago
📄	q_learning.gif	them hình 3 nhóm cuối	2 days ago

📖 README

Sử dụng các thuật toán tìm kiếm AI để giải bài toán 8-Puzzle

Triệu Phúc Hiếu - 23110217

1. Mục tiêu

Dự án nhằm sử dụng các thuật toán tìm kiếm AI để giải bài toán 8-Puzzle với mục đích triển khai, đánh giá, so sánh mức độ hiệu quả, năng suất của từng thuật toán. Các nhóm thuật toán bao gồm:

- Tìm kiếm không có thông tin (Uniformed Search): Gồm có các thuật toán BFS, DFS, UCS và IDS.
- Tìm kiếm có thông tin (Informed Search): Gồm có các thuật toán GBFS, A*, IDA*.
- Tìm kiếm cục bộ (Local Search): Gồm có các thuật toán Simulated Annealing, Beam Search, Genetic Algorithm và nhóm thuật toán Hill Climbing: Simple, Steepest, Stochastic
- Tìm kiếm trong môi trường phức tạp (Searching in Complex Environments): Gồm có các thuật toán AND-OR Graph Search, Searching for a partially observation, Sensorless.
- Bài toán thỏa mãn ràng buộc (Constraint Satisfaction Problems - CSPs): Gồm có các thuật toán Forward-Checking, Backtracking.
- Học tăng cường (Reinforcement Learning - RL): Gồm có thuật toán Q-Learning.

2. Nội dung

Một bài toán tìm kiếm 8-Puzzle thường có các thành phần chính:

- Trạng thái (State): Ma trận 3x3.
- Trạng thái ban đầu (Initial State): Trạng thái của ma trận lúc ban đầu.
- Trạng thái đích (Goal State): Trạng thái mà bài toán cần đạt được.
- Hành động (Actions): Các hành động có thể có của trạng thái như L (Left), R (Right), U (Up), D (Down).
- Kiểm tra đích (Goal Test): Kiểm tra xem trạng thái đang xét có phải trạng thái đích hay chưa.
- Chi phí đường đi (Path Cost): Tổng chi phí của các bước đi từ trạng thái ban đầu đến trạng thái hiện tại.
- Giải pháp (Solution): Một chuỗi các hành động (LRUD...) để giải bài toán từ trạng thái ban đầu đến trạng thái đích.

2.1. Tìm kiếm không có thông tin (Uniformed Search)

- Không gian trạng thái (State Space): Tập hợp tất cả trạng thái có thể sinh ra từ khởi đầu.
- Trạng thái khởi đầu (Initial State): Ở bài toán này sẽ là bảng 8-puzzle [4, 1, 3], [7, 2, 5], [0, 8, 6].
- Trạng thái đích (Goal State): Ở bài toán này sẽ là [1, 2, 3], [4, 5, 6], [7, 8, 0].

Minh họa:

UNINFORMED SEARCH ALGORITHMS

4	1	3
7	2	5
	8	6

BFS

4	1	3
7	2	5
	8	6

DFS

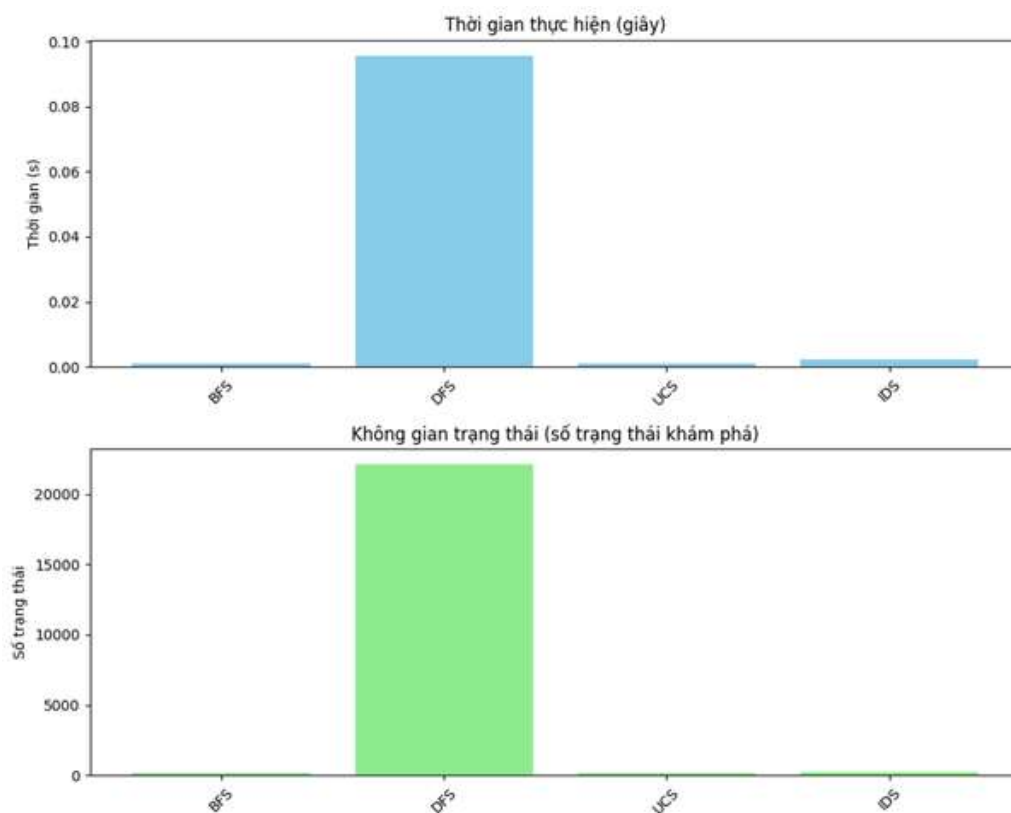
4	1	3
7	2	5
	8	6

UCS

4	1	3
7	2	5
	8	6

IDS

Biểu đồ so sánh khi chạy thuật toán:



Nhận xét:

- Tìm kiếm theo chiều rộng (BFS - Breadth-First Search): Thời gian thực hiện và không gian trạng thái tương đối nhỏ bởi vì thuật toán này tìm kiếm theo lớp, đảm bảo tìm kiếm được lời giải tối ưu nhưng thường tốn bộ nhớ lớn.

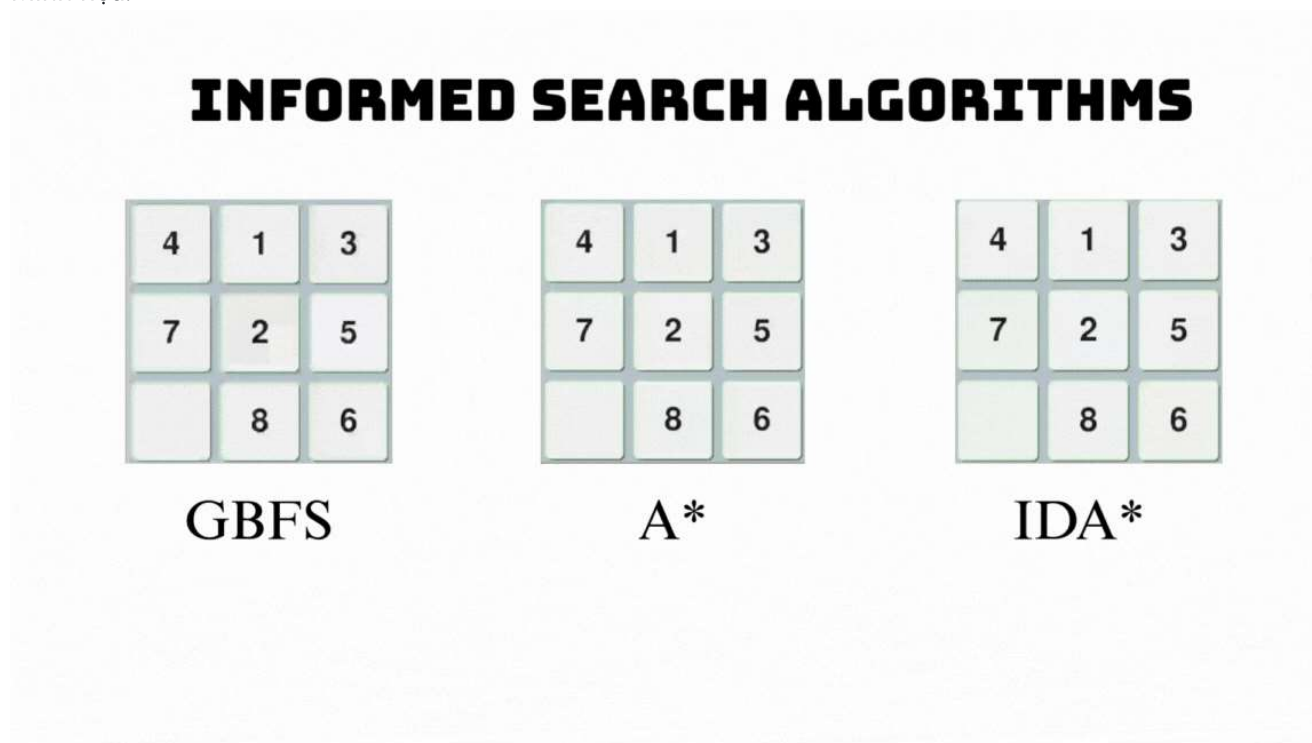
- Tìm kiếm theo chiều sâu (DFS - Depth-First Search): Thời gian thực hiện và không gian trạng thái lớn, vượt trội so với các thuật toán khác bởi vì DFS ưu tiên đi sâu theo nhánh nên dễ rơi vào vòng lặp hoặc đi sai hướng và khám phá các trạng thái không cần thiết.
- Tìm kiếm theo chi phí đồng đều (UCS - Uniform Cost Search): UCS tương tự BFS nhưng có thêm chi phí đường đi, nếu chi phí giữa các trạng thái bằng nhau, UCS sẽ hoạt động tương tự như BFS.
- Tìm kiếm lặp sâu dần (IDS - Iterative Deepening Search): IDS là sự kết hợp của BFS và DFS nhằm tiết kiệm bộ nhớ và tìm được lời giải tối ưu.

Như vậy, các thuật toán BFS, UCS, IDS trong nhóm thuật toán này tối ưu hơn DFS

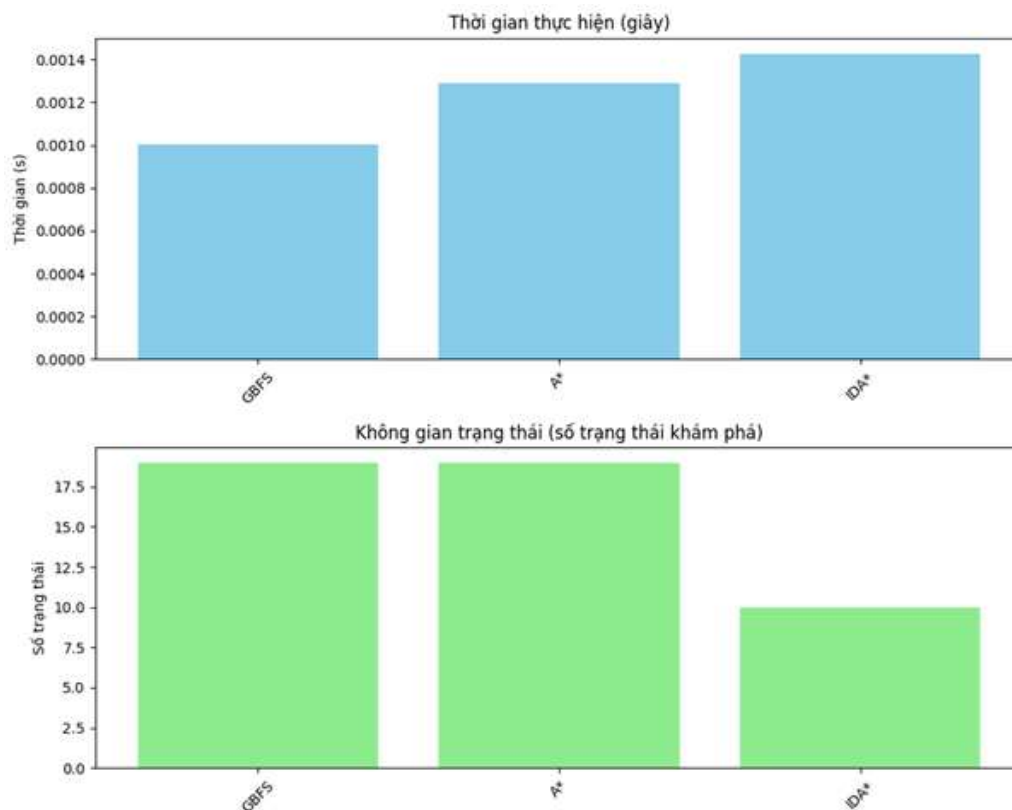
2.2. Tìm kiếm có thông tin (Informed Search)

- Không gian trạng thái (State Space): Tập hợp tất cả trạng thái có thể sinh ra từ khởi đầu.
- Trạng thái khởi đầu (Initial State): Ở bài toán này sẽ là bảng 8-puzzle [4, 1, 3], [7, 2, 5], [0, 8, 6].
- Trạng thái đích (Goal State): Ở bài toán này sẽ là [1, 2, 3], [4, 5, 6], [7, 8, 0].
- Hàm chi phí thực ($g(n)$): Chi phí từ nút gốc tới n .
- Hàm đánh giá (heuristic $h(n)$): Ước lượng chi phí từ n tới đích.

Minh họa:



Biểu đồ so sánh khi chạy thuật toán:



Nhận xét

- Tìm kiếm tham lam theo chiều tốt nhất (GBFS - Greedy Best-First Search): Thời gian thực hiện nhỏ nhất nhưng không gian trạng thái lại lớn nhất trong nhóm thuật toán bởi vì GBFS chỉ dựa vào hàm heuristic (ước lượng đến đích) mà không xét chi phí đã đi qua, nên đôi khi không tối ưu và khám phá nhiều trạng thái sai hướng.
- Tìm kiếm A* (A-Star): A* sử dụng kết hợp giữa chi phí đã đi và ước lượng còn lại ($f(n) = g(n) + h(n)$) nên thường tìm được lời giải tối ưu nếu heuristic tốt, cân bằng được giữa tốc độ và độ chính xác.
- Tìm kiếm lặp sâu theo heuristic (IDA - Iterative Deepening A): IDA* kết hợp giữa A* và IDS, hoạt động theo ngưỡng (threshold) và tái sử dụng DFS có kiểm soát. Tuy thực hiện nhiều lần nhưng mỗi lần dùng ít bộ nhớ.

Như vậy, các thuật toán trong nhóm thuật toán này đều có ưu điểm và nhược điểm, nếu như tìm kiếm thời gian nhanh thì không gian trạng thái lại nhiều (GBFS), tìm kiếm thời gian chậm thì không gian lại ít (IDA) và A-Star cân bằng tốt cả hai.

2.3. Tìm kiếm cục bộ (Local Search)

- Không gian trạng thái (State Space): Tập hợp tất cả trạng thái có thể sinh ra từ khởi đầu.
- Trạng thái khởi đầu (Initial State): Ở bài toán này là bảng 8-puzzle [1, 2, 3], [4, 0, 6], [7, 5, 8].
- Trạng thái đích (Goal State): Ở bài toán này sẽ là [1, 2, 3], [4, 5, 6], [7, 8, 0].
- Láng giềng (Neighbors): Tập các trạng thái sinh ra từ trạng thái hiện tại.
- Hàm đánh giá (Objective/Evaluation Function): Giá trị số mô tả chất lượng trạng thái.
- Chiến lược cập nhật: Quy tắc chọn trạng thái tiếp theo (vd. xác suất, số lượng beam, phép lai/đột biến).
- Điều kiện dừng: Số vòng lặp tối đa, không cải thiện, đạt ngưỡng chất lượng.

Minh họa:

LOCAL SEARCH

1	2	3
4		6
7	5	8

Simple

1	2	3
4		6
7	5	8

Steepest

1	2	3
4		6
7	5	8

Stochastic

1	2	3
4		6
7	5	8

Simulated

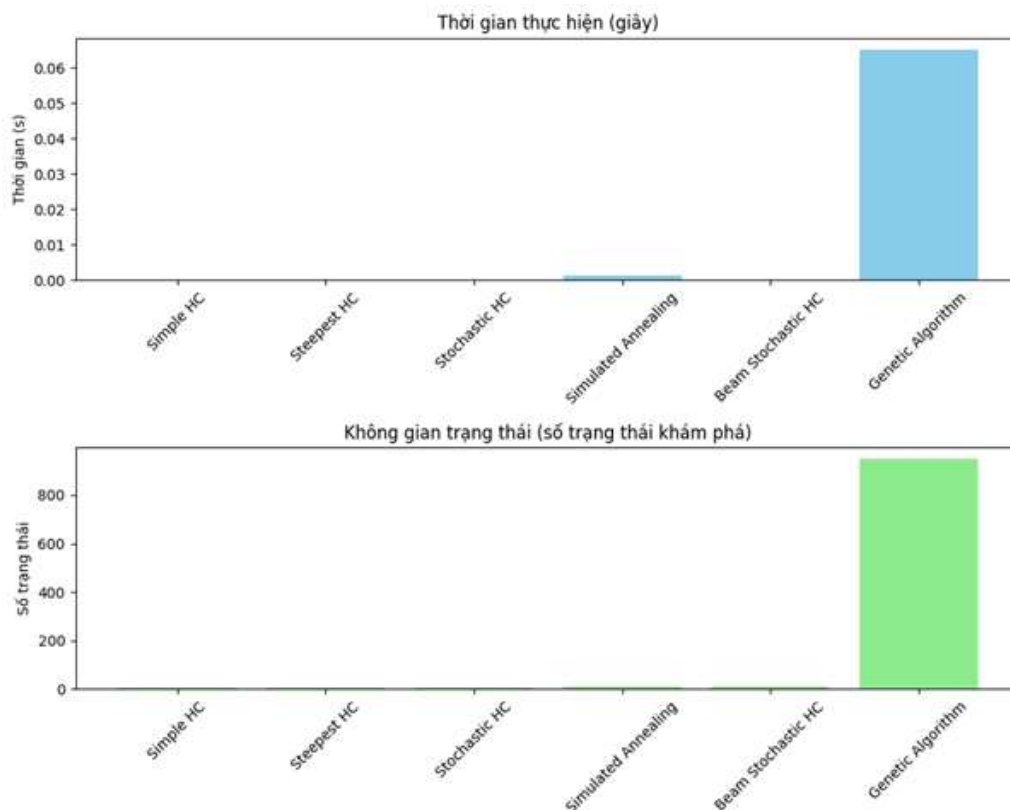
1	2	3
4		6
7	5	8

Beam

1	2	3
4		6
7	5	8

Genetic

Biểu đồ so sánh khi chạy thuật toán:



Nhận xét

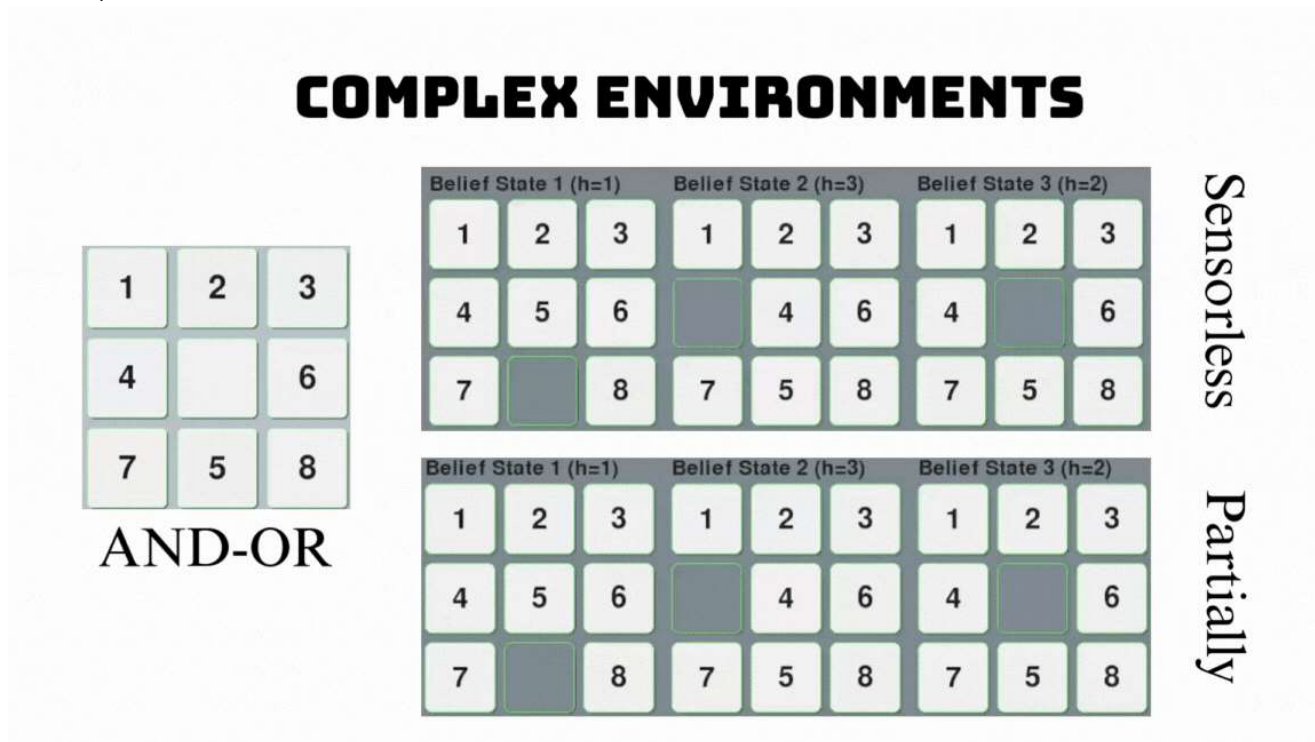
- Leo đồi đơn giản (Simple Hill Climbing): Thuật toán đơn giản, chỉ đi theo hướng tăng mà không lùi, nên có thể nhanh chóng bị kẹt ở cực trị địa phương. Hiệu suất tốt nhưng khả năng tìm lời giải toàn cục thấp.
- Leo đồi dốc nhất (Steepest Hill Climbing): So với Simple Hill Climbing, Steepest Hill Climbing đánh giá tất cả các hàng xóm để chọn hướng tốt nhất, nhưng vẫn dễ bị kẹt ở cực trị. Cải tiến hơn Simple nhưng chưa đủ.
- Leo đồi ngẫu nhiên (Stochastic Hill Climbing): Chọn ngẫu nhiên các bước tăng tốt, nên có thể thoát khỏi một số cực trị địa phương so với Hill Climbing truyền thống, tối ưu hơn.
- Luyện thép mô phỏng (Simulated Annealing): Cho phép chấp nhận lời giải kém hơn với xác suất giảm dần, từ đó tránh được cực trị địa phương tốt hơn Hill Climbing. Chậm hơn nhưng chính xác hơn.
- Tìm kiếm Chùm tia Cục bộ (Local Beam Search): Duy trì nhiều trạng thái đồng thời và chọn hướng đi tốt nhất, nên có khả năng khám phá không gian rộng hơn Hill Climbing. Cân bằng giữa hiệu suất và tối ưu.
- Thuật toán Di truyền (Genetic Algorithm): Dựa vào quần thể, lai ghép, đột biến nên mất thời gian và bộ nhớ hơn nhưng tăng khả năng tìm lời giải toàn cục. Mặc dù chậm và tốn tài nguyên nhưng rất tối ưu.

Như vậy, các thuật toán tìm kiếm cục bộ (Local Search) đều có ưu và nhược điểm riêng: những thuật toán như Hill Climbing có thời gian thực hiện rất nhanh nhưng dễ mắc kẹt tại cực trị địa phương, trong khi các thuật toán như Genetic Algorithm, Beam Stochastic, Simulated Annealing tốn nhiều thời gian và không gian hơn nhưng lại có khả năng tìm ra lời giải tối ưu cao hơn.

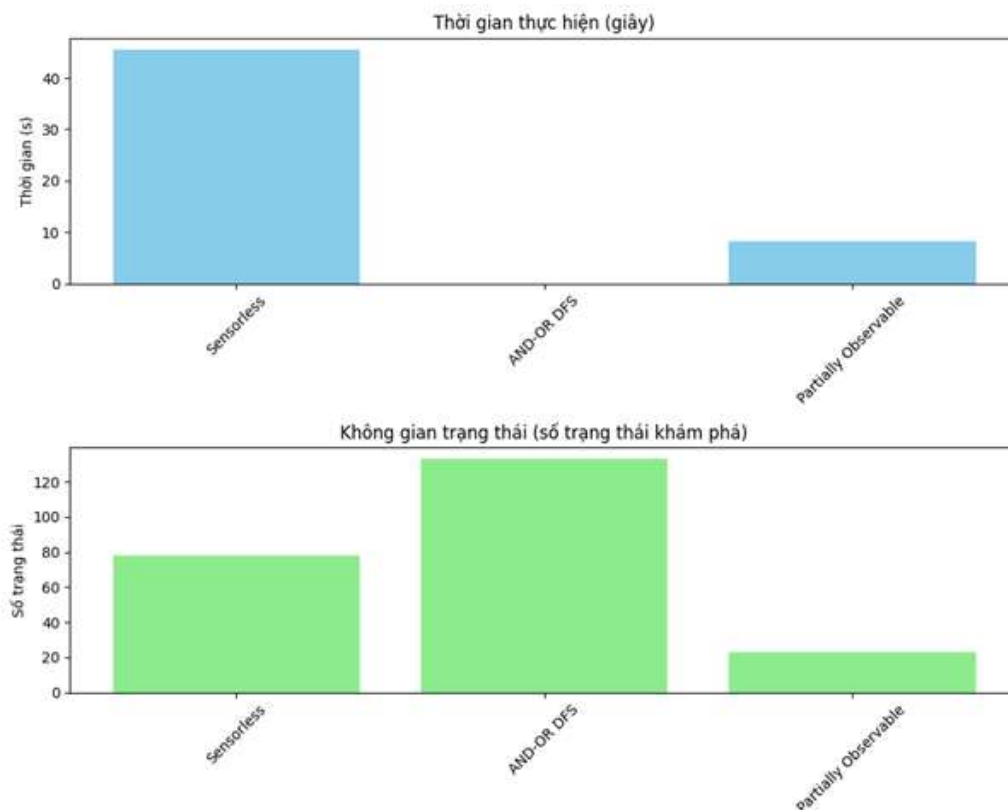
2.4. Các Thuật Toán Cho Môi Trường Phức Tạp (Complex Environments)

Trong môi trường phức tạp, tác tử không thể quan sát toàn bộ môi trường hoặc không biết chính xác trạng thái hiện tại. Những thuật toán này không trực tiếp tối ưu cho bài toán 8-Puzzle truyền thống (vì bài toán này có thông tin đầy đủ), nhưng có thể mở rộng hoặc áp dụng nếu bài toán 8-Puzzle bị giới hạn thông tin.

Minh họa:



Biểu đồ so sánh khi chạy thuật toán:



Nhận xét

- **Sensorless:** Mặc dù khám phá số lượng trạng thái trung bình (~80 trạng thái), nhưng thời gian thực hiện lại cao nhất (~45 giây). Điều này cho thấy mỗi trạng thái tốn nhiều thời gian xử lý do thiếu thông tin cảm biến, dẫn đến việc thuật toán phải thử nhiều hướng mà không có chỉ dẫn rõ ràng.
- **AND-OR DFS:** Khám phá nhiều trạng thái nhất (~120 trạng thái), cho thấy khả năng bao phủ không gian trạng thái tốt. Biểu đồ thời gian hiển thị 0s (~0.08s) nhanh nhất trong nhóm thuật toán.
- **Partially Observable:** Có số lượng trạng thái khám phá ít nhất (~20 trạng thái) và thời gian thực hiện nhanh nhất (~7 giây). Điều này cho thấy thuật toán được tối ưu tốt, chỉ chọn lọc trạng thái cần thiết để khám phá. Mặc dù không quan sát toàn bộ môi trường, nhưng vẫn đạt được hiệu quả cao.

Như vậy, các phương pháp giải quyết bài toán tìm kiếm đều có ưu và nhược điểm riêng: **Sensorless** cho thấy khả năng xử lý yếu khi thiếu thông tin – thời gian thực hiện lâu, ít hiệu quả. **AND-OR DFS** có khả năng khám phá toàn diện nhưng có thể chưa tối ưu về thời gian hoặc bộ nhớ. **Partially Observable** tuy bị giới hạn về thông tin đầu vào nhưng lại thể hiện tốt nhờ thuật toán tối ưu – khám phá ít trạng thái và thực hiện nhanh.

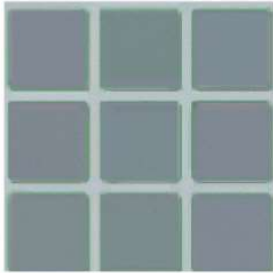
2.5. Bài Toán Thỏa Mãn Ràng Buộc (Constraint Satisfaction Problems - CSPs)

Mô hình CSP:

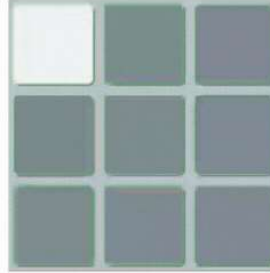
- **Biến (Variables):** Mỗi ô trong ma trận 3x3 là một biến.
- **Miền giá trị (Domains):** Tập các số từ 0 đến 8.
- **Ràng buộc (Constraints):** Mỗi số chỉ xuất hiện một lần trong ma trận (tất cả khác nhau - AllDifferent constraint). Các ràng buộc về quy tắc di chuyển giữa các trạng thái.

Minh họa:

CONSTRAINT SATISFACTION PROBLEMS

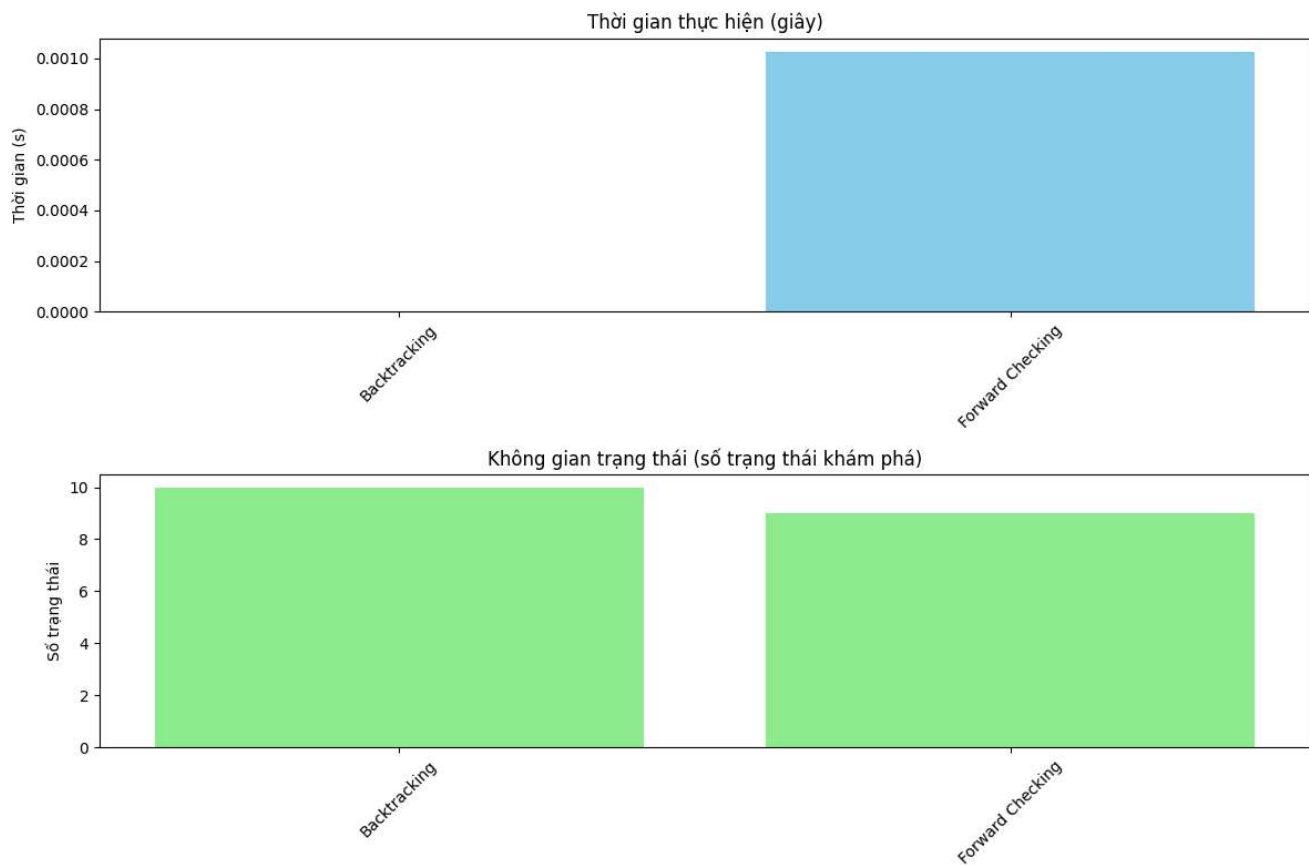


Backtracking



Forward Checking

Biểu đồ so sánh khi chạy thuật toán:



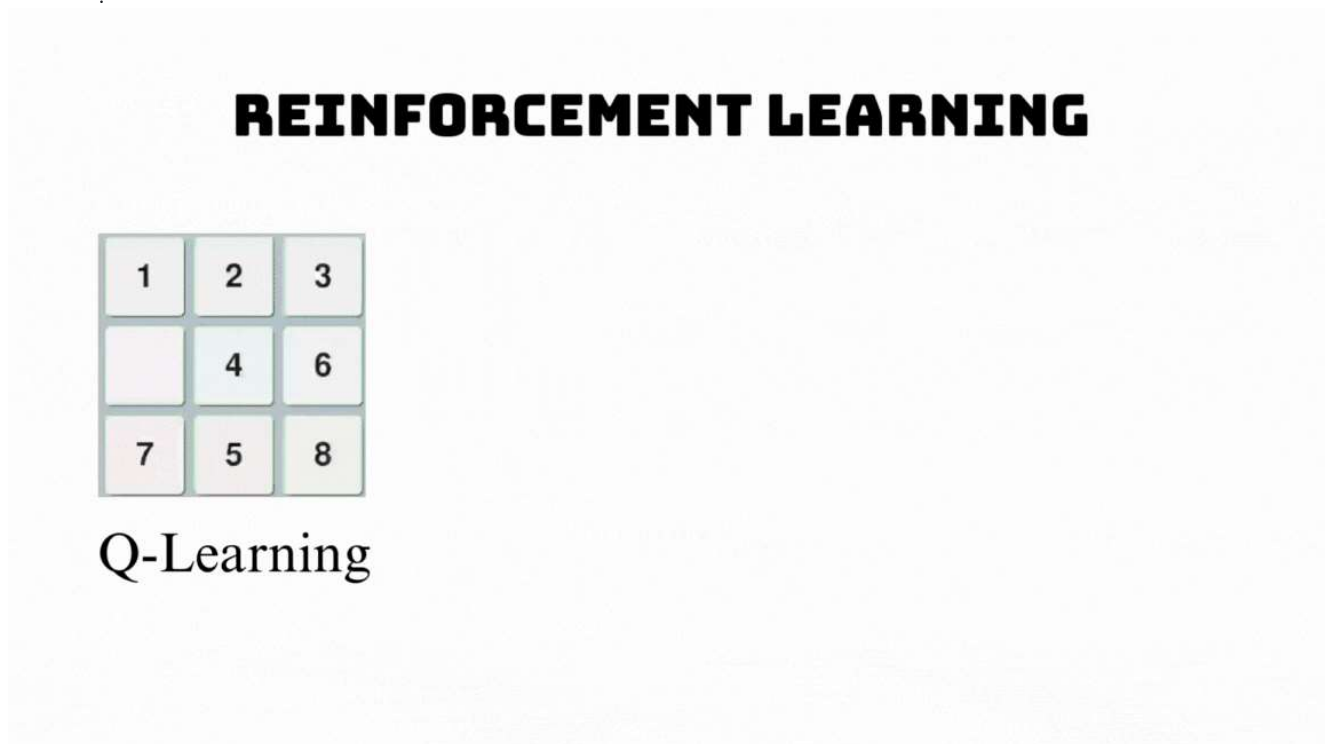
Nhận xét

- Backtracking: Khám phá số lượng trạng thái nhiều hơn nhưng lại có thời gian thực hiện nhanh nhất. Điều này cho thấy mặc dù thuật toán duyệt nhiều hướng hơn, nhưng không tốn nhiều thời gian xử lý mỗi trạng thái. Tuy nhiên, trong bài toán lớn hơn, cách tiếp cận này dễ gặp phải tình trạng phải quay lui nhiều lần, làm tăng chi phí tính toán.
- Forward Checking: Khám phá ít trạng thái hơn, cho thấy khả năng cắt tỉa không gian trạng thái tốt hơn nhờ kiểm tra ràng buộc sớm. Tuy nhiên, thời gian thực hiện lại cao hơn, có thể do chi phí tính toán bổ sung cho việc kiểm tra trước mỗi bước gán giá trị. Điều này hợp lý về mặt lý thuyết, và hiệu quả hơn khi áp dụng cho các bài toán có không gian tìm kiếm lớn và ràng buộc phức tạp.

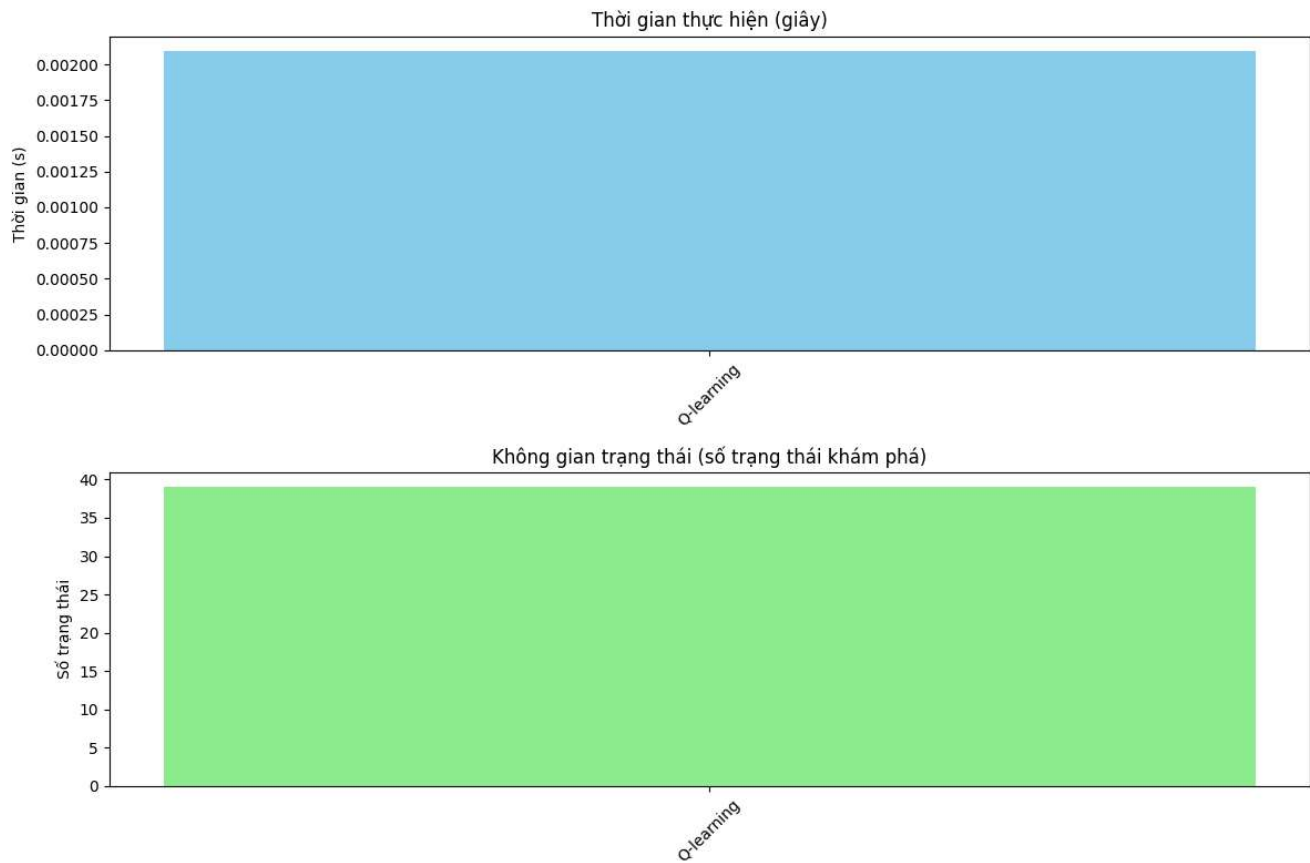
Như vậy, hai phương pháp Backtracking và Forward Checking đều có đặc điểm riêng phù hợp với từng bài toán: Backtracking đơn giản, dễ triển khai, có thể nhanh trên bài toán nhỏ nhưng không hiệu quả trong không gian lớn. Forward Checking tuy có chi phí kiểm tra bổ sung nhưng giúp tiết kiệm không gian tìm kiếm và đặc biệt hiệu quả khi xử lý các bài toán CSP phức tạp với nhiều ràng buộc.

2.6. Học Tăng Cường (Reinforcement Learning - RL)

Minh họa:



Biểu đồ so sánh khi chạy thuật toán:



Nhận xét

- Ưu điểm: Tác tử có thể học mà không cần biết rõ thuật toán giải — học từ kinh nghiệm.
- Nhược điểm: Cần huấn luyện qua nhiều lần chơi thử (episodes), tốn thời gian. Không đảm bảo tìm lời giải tối ưu ngay.

<https://github.com/TrieuHieu842/8-puzzle-with-6-group-Algorithms.git>

Releases

No releases published

[Create a new release](#)

Packages


No packages published
[Publish your first package](#)

Languages

● Python 100.0%

Suggested workflows


Based on your tech stack



Python package

Create and test a Python package on multiple Python versions.


Configure



Pylint

Lint a Python application with pylint.

Configure



Django

Build and Test a Django Project

Configure

[More workflows](#)

Dismiss suggestions