TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP.HCM

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN CÁ NHÂN

ỨNG DỤNG CÁC THUẬT TOÁN TÌM KIẾM GIẢI BÀI TOÁN 8PUZZLE

MÔN HỌC: TRÍ TUỆ NHÂN TẠO

HỌC KỲ 2 - NĂM HỌC 2024 -2025

Sinh viên thực hiện: Nguyễn Hồ Phát – 23133054

GVHD: TS. Phan Thị Huyền Trang

TP.HCM, 25 THÁNG 04 NĂM 2025

MỤC LỤC

1. MỤC TIÊU

Mục tiêu của chương trình là xây dựng một hệ thống giải bài toán 8-Puzzle bằng cách áp dụng nhiều nhóm thuật toán khác nhau trong lĩnh vực Trí Tuệ Nhân Tạo. Việc này nhằm so sánh hiệu quả giữa các thuật toán, hiểu rõ đặc điểm, ưu nhược điểm của từng phương pháp trong việc giải quyết bài toán tìm kiếm, từ đó nâng cao khả năng chọn lựa giải pháp phù hợp cho các vấn đề tương tự trong thực tế.

2. NỘI DUNG

Chương trình được tổ chức thành nhiều nhóm thuật toán dựa theo phân loại trong AI:

1.Uniformed Search: BFS, DFS, UCS, IDDFS

2.Informed Search: Greedy, A*, IDA*, Genetic Algorithm (GA), Beam Search

3.Local Search: Simple Hill Climbing, Steepest Hill Climbing, Stochastic Hill Climbing, Simulated Annealing (SA)

4.Complex Environments: Nondeterministic search, No observation search, Partially observable search

5.Constraint Satisfaction Problems (CSPs): Backtracking, Min-conflict, Backtracking with Forward Checking

6.Reinforcement Learning: Q-Learning

2.1. Các thuật toán tìm kiếm có thông tin (Informed Search)

IDA* – Iterative Deepening A*

Thuật toán IDA* kết hợp hai thuật toán:

A*: Tìm kiếm theo chi phí tổng f(n) = g(n) + h(n) (chi phí đến hiện tại + ước lượng đến đích).

Iterative Deepening (tìm kiếm sâu dần): Tìm theo mức giới hạn chi phí và mở rộng dần mức giới hạn này.

IDA* tận dụng bộ nhớ thấp của DFS và sức mạnh định hướng của heuristic trong A*. Đây là một trong những thuật toán hiệu quả nhất cho các bài toán như 8-Puzzle.

Các thành phần của bài toán tìm kiếm trong 8-Puzzle

Trạng thái ban đầu (Initial State): Ma trận 3x3 gồm các ô số từ 1 đến 8 và một ô trống (0). Đảm bảo trạng thái có lời giải.

Tập hành động (Actions): Di chuyển ô trống theo các hướng trái, phải, lên, xuống.

Hàm chuyển trạng thái (Transition Model): Hoán đổi vị trí ô trống với ô lân cận theo hướng hành động.

Kiểm tra mục tiêu (Goal Test): So sánh trạng thái hiện tại với trạng thái đích (ví dụ: 1–2–3 | 4–5–6 | 7–8–0).

Hàm chi phí (Path Cost): Mỗi bước di chuyển có chi phí 1. Tổng chi phí = tổng số bước.

Heuristic (Hàm ước lượng h(n)): Dùng để định hướng tìm kiếm. Phổ biến: Tổng khoảng cách Manhattan (Manhattan Distance).

Hoạt động của thuật toán IDA*

Khởi tạo threshold bằng giá trị f(start) = g(start) + h(start) của trạng thái ban đầu.

Thực hiện DFS với giới hạn chi phí không vượt quá threshold.

Nếu tìm thấy đích → trả về lời giải.

Nếu không, tăng threshold lên giá trị nhỏ nhất của f(n) đã vượt giới hạn trong lần tìm trước và lặp lại.

Quá trình tiếp tục cho đến khi tìm được lời giải hoặc không còn trạng thái nào để mở rộng.

IDA* sử dụng DFS có giới hạn chi phí, không phải giới hạn độ sâu.

Solution:

Trả về chuỗi hành động từ trạng thái ban đầu đến trạng thái đích, được định hướng bởi hàm f(n) = g(n) + h(n).

Đảm bảo lời giải tối ưu nếu heuristic thỏa mãn tính nhất quán (consistent).

Hiệu suất

Ưu điểm:

Tối ưu hóa bộ nhớ: chỉ cần dùng không gian giống DFS.

Tìm lời giải tối ưu nếu heuristic phù hợp.

Thích hợp cho bài toán có không gian trạng thái lớn.

Nhược điểm:

Phải duyệt lại nhiều trạng thái trong mỗi vòng lặp do đặc trưng của DFS.

Hiệu suất phụ thuộc lớn vào chất lượng của heuristic.

Không hiệu quả nếu heuristic kém hoặc không nhất quán.

Greedy Search

Thuật toán Greedy Search là một chiến lược tìm kiếm dựa hoàn toàn vào hàm heuristic h(n) để định hướng. Tại mỗi bước, chỉ mở rộng nút có giá trị heuristic nhỏ nhất (ước lượng gần đích nhất), mà bỏ qua chi phí đã đi (g(n)).

Các thành phần của bài toán tìm kiếm trong 8-Puzzle

Trạng thái ban đầu (Initial State):

Ma trận 3x3 gồm các ô số 1–8 và một ô trống (0).

Tập hành động (Actions):

Di chuyển ô trống theo bốn hướng: trái, phải, lên, xuống.

Hàm chuyển trạng thái (Transition Model):

Hoán đổi vị trí ô trống với ô lân cận.

Kiểm tra mục tiêu (Goal Test):

Kiểm tra xem trạng thái hiện tại có giống trạng thái đích không.

Hàm đánh giá:

• Heuristic h(n) → ước lượng chi phí từ n đến đích.

• Ví dụ: số ô sai vị trí, khoảng cách Manhattan.

Lưu ý: Greedy Search chỉ dùng h(n), không dùng f(n) = g(n) + h(n).

Hoạt động của thuật toán Greedy Search

Khởi tạo: Đặt trạng thái ban đầu vào tập open list.

Lặp lại

• Chọn nút n có h(n) nhỏ nhất trong open list.

• Nếu n là trạng thái đích → trả về lời giải.

• Nếu không, mở rộng n, thêm các trạng thái con vào open list.

• Loại bỏ n khỏi open list, thêm vào closed list (nếu dùng).

Kết thúc

Dừng khi tìm thấy đích hoặc không còn nút nào để mở rộng.

Solution

Trả về một chuỗi hành động từ trạng thái ban đầu đến đích dựa trên hướng đi "ngắn nhất theo ước lượng".

Lưu ý: Greedy Search không đảm bảo tìm được lời giải tối ưu. Nó có thể rơi vào bẫy cục bộ hoặc vòng lặp nếu không cẩn thận.

Hiệu suất

Ưu điểm:

Nhanh, vì chỉ tập trung mở rộng nút hứa hẹn nhất.

Tốn ít bộ nhớ hơn A* (không cần lưu g(n)).

Nhược điểm:

Không đảm bảo tối ưu (dễ rơi vào lời giải không tốt nhất).

Có thể mắc kẹt trong bẫy heuristic hoặc bị lặp vô hạn nếu không kiểm soát.

Phụ thuộc mạnh vào chất lượng heuristic.

A* – A Star Search Algorithm

Thuật toán A* kết hợp hai yếu tố quan trọng:

Tìm kiếm theo chi phí tối ưu: Xem xét tổng chi phí ước lượng để đến đích qua hàm

f(n) = g(n) + h(n)

• g(n): chi phí thực từ trạng thái ban đầu đến n

• h(n): chi phí ước lượng từ n đến đích (heuristic)

Ưu tiên mở rộng nút tiềm năng: Luôn chọn nút có giá trị f(n) nhỏ nhất trong tập mở (open list).

Các thành phần của bài toán tìm kiếm trong 8-Puzzle

Trạng thái ban đầu (Initial State): Ma trận 3x3 gồm các ô số 1–8 và một ô trống (0). Đảm bảo trạng thái có thể giải được.

Tập hành động (Actions):

Di chuyển ô trống theo bốn hướng: trái, phải, lên, xuống.

Hàm chuyển trạng thái (Transition Model):

Hoán đổi vị trí ô trống với ô kề theo hướng di chuyển.

Kiểm tra mục tiêu (Goal Test):

So sánh trạng thái hiện tại với trạng thái đích (thường là 1–2–3 | 4–5–6 | 7–8–0).

Hàm chi phí (Path Cost): Mỗi bước di chuyển có chi phí 1. Tổng chi phí = tổng số bước từ đầu đến trạng thái hiện tại.

Heuristic (Hàm ước lượng h(n)): Khoảng cách Manhattan

Hoạt động của thuật toán A*

Khởi tạo

Tập mở (open list) chứa trạng thái ban đầu.

Tập đóng (closed list) rỗng.

Lặp lại

• Lấy nút n có f(n) nhỏ nhất từ open list.

• Nếu n là đích → trả về lời giải.

• Nếu không, thêm n vào closed list.

• Mở rộng n: sinh các trạng thái kế cận.

• Với mỗi trạng thái kế cận:

– Nếu chưa nằm trong closed list hoặc có đường đi tốt hơn, tính f(n), thêm hoặc cập nhật vào open list.

Kết thúc: Thuật toán dừng khi tìm thấy đích hoặc không còn nút nào trong open list (→ không có lời giải).

Solution: Trả về chuỗi hành động tối ưu từ trạng thái ban đầu đến trạng thái đích, được xác định bởi đường đi có chi phí f(n) nhỏ nhất.

Đảm bảo lời giải tối ưu nếu heuristic thỏa mãn:

Admissible (không đánh giá quá thấp chi phí thực).

Consistent (thỏa mãn bất đẳng thức tam giác).

Hiệu suất

Ưu điểm:

Tìm được lời giải tối ưu nếu heuristic phù hợp.

Thường mở rộng ít nút hơn so với tìm kiếm không heuristic (như Uniform Cost Search).

Có thể dễ dàng tùy biến heuristic để tăng tốc tìm kiếm.

Nhược điểm:

Tốn bộ nhớ lớn: lưu toàn bộ open list và closed list.

Hiệu suất phụ thuộc mạnh vào chất lượng heuristic.

Không phù hợp cho không gian trạng thái cực lớn nếu không có heuristic tốt.

2.2 Các thuật toán tìm kiếm không có thông tin

BFS

Thuật toán Breadth-First Search (BFS) là một trong những phương pháp tìm kiếm không sử dụng thông tin bổ sung về trạng thái đích. BFS hoạt động theo nguyên lý mở rộng tất cả các trạng thái ở mức hiện tại trước khi chuyển sang mức tiếp theo, đảm bảo tìm ra lời giải có số bước ít nhất.

Các thành phần của bài toán tìm kiếm trong 8-Puzzle

Trạng thái ban đầu (Initial State) : Một mảng 3x3 gồm 8 ô số (1–8) và một ô trống (thường ký hiệu là 0). Vị trí các ô được sắp xếp ngẫu nhiên nhưng đảm bảo bài toán có lời giải.

Tập hành động (Actions): Tại mỗi trạng thái, ô trống có thể được di chuyển theo 4 hướng: trái, phải, lên, xuống, tùy vào vị trí của nó trong lưới.

Hàm chuyển trạng thái (Transition Model): Một hành động sẽ tạo ra một trạng thái mới bằng cách hoán đổi vị trí của ô trống với ô liền kề theo hướng di chuyển.

Kiểm tra mục tiêu (Goal Test): Kiểm tra xem trạng thái hiện tại có giống với trạng thái mục tiêu hay không. Trạng thái mục tiêu thường là:

Hàm chi phí (Path Cost): Mỗi hành động được gán chi phí là 1. Tổng chi phí bằng tổng số bước thực hiện từ trạng thái đầu đến trạng thái đích.

Hoạt động của thuật toán BFS

Sử dụng một hàng đợi (queue) để lưu các trạng thái chờ mở rộng.

Khởi tạo bằng trạng thái ban đầu và đưa nó vào hàng đợi.

Lặp lại:

Lấy trạng thái đầu tiên khỏi hàng đợi.

Nếu là trạng thái đích → trả về chuỗi hành động dẫn đến nó.

Nếu không, tạo tất cả các trạng thái con (sau khi thực hiện các hành động hợp lệ).

Thêm các trạng thái con chưa từng được duyệt vào hàng đợi.

Quá trình dừng khi tìm thấy trạng thái đích hoặc không còn trạng thái nào để mở rộng.

Thuật toán đảm bảo rằng trạng thái đích sẽ được tìm thấy với số bước ít nhất vì nó mở rộng theo từng mức.

Solution:

Lời giải là một chuỗi các hành động (ví dụ: ["Right", "Down", "Left", "Up"]) biến trạng thái ban đầu thành trạng thái đích. Với BFS, chuỗi này luôn là lời giải ngắn nhất (ít bước nhất).

Hiệu suất:

Ưu điểm

Đảm bảo tìm lời giải ngắn nhất về số bước.

Thực thi đơn giản, dễ kiểm chứng đúng sai.

Hoàn tất trong không gian trạng thái hữu hạn như 8-Puzzle.

Nhược điểm

Tốn bộ nhớ nghiêm trọng nếu lời giải sâu.

Không hiệu quả với các bài toán mở rộng như 15-Puzzle.

Không tận dụng bất kỳ thông tin nào về mục tiêu (không có heuristic).

DFS

Thuật toán Depth-First Search (DFS) là một phương pháp tìm kiếm không sử dụng thông tin bổ sung về trạng thái đích. DFS hoạt động theo nguyên lý đi sâu vào nhánh con đầu tiên cho đến khi không thể đi tiếp (chạm nút lá hoặc gặp trạng thái đã duyệt), sau đó mới quay lui để tiếp tục nhánh kế tiếp. DFS không đảm bảo tìm ra lời giải tối ưu về số bước, nhưng có thể tiết kiệm bộ nhớ hơn so với BFS.

Các thành phần của bài toán tìm kiếm trong 8-Puzzle

Trạng thái ban đầu (Initial State): Một ma trận 3x3 gồm 8 ô số (1–8) và một ô trống (0). Trạng thái phải đảm bảo là khả giải (có lời giải tồn tại).

Tập hành động (Actions): Di chuyển ô trống theo các hướng: lên, xuống, trái, phải, tùy theo vị trí hiện tại trong lưới.

Hàm chuyển trạng thái (Transition Model): Thực hiện một hành động sẽ tạo ra một trạng thái mới bằng cách hoán đổi ô trống với ô liền kề theo hướng tương ứng.

Kiểm tra mục tiêu (Goal Test): Kiểm tra xem trạng thái hiện tại có khớp với trạng thái đích đã cho không. Trạng thái mục tiêu thường là:

Hàm chi phí (Path Cost): Mỗi hành động có chi phí là 1. Tổng chi phí là tổng số hành động đã thực hiện.

Hoạt động của thuật toán DFS

Sử dụng một ngăn xếp (stack) để lưu các trạng thái chờ mở rộng.

Khởi tạo bằng trạng thái ban đầu và đưa nó vào ngăn xếp.

Lặp lại:

Lấy trạng thái ở đỉnh ngăn xếp ra.

Nếu là trạng thái đích → trả về chuỗi hành động dẫn đến nó.

Nếu không, tạo các trạng thái con hợp lệ.

Thêm các trạng thái con chưa từng được duyệt vào ngăn xếp.

Quá trình kết thúc khi tìm thấy trạng thái đích hoặc ngăn xếp trống.

DFS ưu tiên đi sâu trước, nên có thể tìm ra lời giải nhanh ở những trường hợp đích nằm gần rễ theo một nhánh cụ thể. Tuy nhiên, do không duyệt theo mức, nên không đảm bảo tìm ra lời giải ngắn nhất.

Solution:

Lời giải là một chuỗi hành động như ["Right", "Right", "Down", "Left", "Down", "Left"] biến trạng thái ban đầu thành trạng thái đích.

DFS không đảm bảo lời giải ngắn nhất.

Hiệu suất

Ưu điểm:

Bộ nhớ ít hơn BFS, vì chỉ cần lưu các trạng thái theo chiều sâu (ngăn xếp).

Có thể nhanh chóng tìm thấy lời giải nếu nó nằm ở nhánh đầu tiên.

Dễ cài đặt.

Nhược điểm:

Không đảm bảo lời giải ngắn nhất.

Có thể rơi vào vòng lặp vô hạn nếu không kiểm tra trạng thái đã duyệt.

Không hiệu quả nếu lời giải nằm ở mức rất nông nhưng DFS lại đi lạc quá sâu ở nhánh sai.

UCS – Uniform Cost Search

Thuật toán Uniform Cost Search (UCS) là một thuật toán tìm kiếm không có thông tin (uninformed search) nhưng có xét đến chi phí của đường đi. UCS mở

rộng các trạng thái theo tổng chi phí đường đi thấp nhất từ trạng thái ban đầu đến trạng thái hiện tại. UCS sử dụng một hàng đợi ưu tiên (priority queue), đảm bảo tìm được lời giải tối ưu về tổng chi phí, không chỉ về số bước.

Các thành phần của bài toán tìm kiếm trong 8-Puzzle

Trạng thái ban đầu (Initial State): Ma trận 3x3 chứa 8 ô số (1–8) và một ô trống (0), có thể sắp xếp ngẫu nhiên nhưng phải khả giải.

Tập hành động (Actions): Ô trống có thể di chuyển lên, xuống, trái, phải – tùy thuộc vào vị trí của nó trong lưới.

Hàm chuyển trạng thái (Transition Model): Tạo trạng thái mới bằng cách hoán đổi ô trống với ô liền kề theo hướng di chuyển.

Kiểm tra mục tiêu (Goal Test): Kiểm tra xem trạng thái hiện tại có khớp với trạng thái đích hay không.

Hàm chi phí (Path Cost): Mỗi hành động có thể gán chi phí là 1 (nếu đồng đều) hoặc một giá trị khác nếu mô hình mở rộng. UCS ưu tiên mở rộng trạng thái có tổng chi phí nhỏ nhất.

Hoạt động của thuật toán UCS

Sử dụng một hàng đợi ưu tiên (priority queue) với ưu tiên là tổng chi phí đường đi (g(n)).

Khởi tạo: Đưa trạng thái ban đầu vào hàng đợi với chi phí 0.

Lặp lại:

Lấy trạng thái có chi phí nhỏ nhất khỏi hàng đợi.

Nếu đó là trạng thái đích → trả về chuỗi hành động tương ứng.

Nếu không, tạo các trạng thái con hợp lệ.

Thêm hoặc cập nhật các trạng thái con vào hàng đợi nếu:

Chúng chưa được duyệt.

Hoặc tìm thấy đường đi tốt hơn đến chúng (chi phí thấp hơn).

Quá trình dừng khi tìm được trạng thái đích hoặc hàng đợi rỗng.

UCS tương đương với Dijkstra’s algorithm nếu áp dụng cho tìm đường đi ngắn nhất trên đồ thị.

Solution:

Trả về chuỗi hành động biến trạng thái ban đầu thành trạng thái đích, với tổng chi phí thấp nhất (dù số bước có thể không ít nhất nếu chi phí mỗi bước khác nhau).

Trong bài toán 8-Puzzle với chi phí mỗi bước = 1, UCS sẽ giống với BFS trong kết quả (vì tổng chi phí = số bước).

Hiệu suất

Ưu điểm:

Luôn tìm được lời giải tối ưu (chi phí thấp nhất).

Ứng dụng rộng rãi trong các bài toán có chi phí hành động không đồng đều.

Tổng quát hơn BFS.

Nhược điểm:

Chi phí tính toán và bộ nhớ cao nếu không có giới hạn độ sâu.

Nếu tất cả chi phí đều bằng nhau thì hiệu suất kém hơn BFS vì tốn công duy trì hàng đợi ưu tiên.

Iterative Deepening Depth-First Search (IDDFS)

Iterative Deepening Depth-First Search (IDDFS) là một kỹ thuật kết hợp giữa DFS (Depth-First Search) và BFS (Breadth-First Search) nhằm khai thác ưu điểm của cả hai: bộ nhớ tiết kiệm như DFS nhưng vẫn đảm bảo tìm được lời giải tối ưu như BFS. IDDFS đặc biệt phù hợp với các bài toán tìm kiếm có không

gian trạng thái rộng nhưng độ sâu lời giải không quá lớn, ví dụ như bài toán 8-Puzzle.

Các thành phần của bài toán tìm kiếm trong 8-Puzzle

Trạng thái ban đầu (Initial State)

Cấu hình lưới 3x3 với 8 ô số (1–8) và 1 ô trống (0), xuất phát từ một trạng thái bất kỳ nhưng hợp lệ (có thể giải được).

Tập hành động (Actions)

Tại mỗi trạng thái, ô trống có thể được di chuyển lên, xuống, trái hoặc phải, tùy vào vị trí hiện tại.

Hàm chuyển trạng thái (Transition Model)

Thực hiện một hành động → tạo trạng thái mới bằng cách đổi chỗ ô trống với ô liền kề tương ứng.

Kiểm tra mục tiêu (Goal Test)

Kiểm tra xem trạng thái hiện tại có giống với trạng thái đích (thường là trạng thái đã sắp xếp đúng theo thứ tự từ 1 đến 8, với 0 ở cuối) hay không.

Hàm chi phí (Path Cost)

Mỗi hành động có chi phí bằng 1 → tổng chi phí là số bước di chuyển.

Hoạt động của thuật toán IDDFS

Gọi một hàm tìm kiếm theo chiều sâu (DFS) với giới hạn độ sâu d.

Bắt đầu với d = 0, sau đó tăng dần (d = 1, 2, 3,...) cho đến khi tìm được lời giải hoặc đạt giới hạn cụ thể.

Ở mỗi vòng lặp, thuật toán sẽ tìm kiếm lại từ đầu đến độ sâu d hiện tại.

Trong quá trình này, thuật toán không lưu toàn bộ cây trạng thái, giúp tiết kiệm đáng kể bộ nhớ.

Solution

Lời giải là chuỗi các hành động hợp lệ từ trạng thái đầu đến trạng thái đích. IDDFS đảm bảo lời giải ngắn nhất về số bước (giống BFS), vì nó kiểm tra các độ sâu nhỏ trước.

Ưu điểm và nhược điểm của IDDFS trong 8-Puzzle

Ưu điểm

Đảm bảo tìm được lời giải ngắn nhất (nếu chi phí bằng nhau).

Bộ nhớ tiêu tốn rất ít → phù hợp với không gian trạng thái lớn.

Kết hợp được điểm mạnh của BFS và DFS.

Hiệu quả hơn DFS thuần vì tránh vòng lặp vô tận và thiếu tối ưu.

Nhược điểm

Phải lặp lại việc duyệt ở các độ sâu nhỏ → chi phí thời gian tăng.

Không sử dụng thông tin heuristic → tốc độ chậm hơn các thuật toán có thông tin (A*, IDA*).

2.3 Các thuật toán local search

Stochastic Hill Climbing – Leo đồi ngẫu nhiên

Thuật toán Stochastic Hill Climbing là một biến thể của Hill Climbing, thuộc nhóm tìm kiếm cục bộ (local search), hoạt động bằng cách: Lựa chọn ngẫu nhiên một trong số các hàng xóm tốt hơn thay vì luôn chọn hàng xóm tốt nhất.

Các thành phần của bài toán tìm kiếm trong 8-Puzzle

Trạng thái ban đầu (Initial State): Ma trận 3x3 gồm các ô số 1–8 và một ô trống (0).

Tập hành động (Actions):

Di chuyển ô trống theo các hướng trái, phải, lên, xuống.

Hàm chuyển trạng thái (Transition Model):

Hoán đổi ô trống với ô lân cận.

Kiểm tra mục tiêu (Goal Test):

Kiểm tra trạng thái hiện tại có giống trạng thái đích không.

Hàm đánh giá:

Heuristic h(n) – đo lường mức "gần đúng" của trạng thái so với đích (ví dụ: số ô sai vị trí, khoảng cách Manhattan).

Hoạt động của thuật toán Stochastic Hill Climbing

Khởi tạo

Bắt đầu từ trạng thái ban đầu.

Lặp lại

• Sinh tập các hàng xóm hợp lệ của trạng thái hiện tại.

• Chọn ngẫu nhiên một trong các hàng xóm có giá trị heuristic tốt hơn (nhỏ hơn) so với trạng thái hiện tại.

• Cập nhật trạng thái hiện tại thành hàng xóm được chọn.

• Nếu đạt đích → trả về lời giải.

Kết thúc

Dừng khi tìm thấy đích hoặc không còn hàng xóm nào tốt hơn (tụt vào đỉnh cục bộ).

Solution

Trả về chuỗi hành động từ trạng thái ban đầu đến trạng thái đích, đi theo các bước leo đồi ngẫu nhiên.

Lưu ý: Thuật toán này có thể:

Kẹt tại đỉnh cục bộ (local maxima).

Lặp vô hạn trong vùng bằng phẳng (plateau) nếu không kiểm soát.

Không đảm bảo tìm được lời giải tối ưu.

Hiệu suất

Ưu điểm:

Đơn giản, dễ cài đặt.

Ít tốn bộ nhớ (chỉ cần lưu trạng thái hiện tại).

Tránh được một số bẫy mà Hill Climbing tham lam gặp phải (vì chọn ngẫu nhiên, không quá cứng nhắc).

Nhược điểm:

Dễ kẹt tại đỉnh cục bộ.

Không đảm bảo tìm được lời giải tối ưu.

Không đảm bảo tìm thấy đích, đặc biệt nếu không cho phép quay lại hoặc restart.

Steepest-Ascent Hill Climbing – Leo đồi dốc nhất

Steepest-Ascent Hill Climbing là một thuật toán tìm kiếm cục bộ (local search) chọn hàng xóm tốt nhất tại mỗi bước. Đây là biến thể "tham lam nhất" của Hill Climbing, vì luôn chọn bước đi có cải thiện lớn nhất theo hàm đánh giá.

Các thành phần của bài toán tìm kiếm trong 8-Puzzle

Trạng thái ban đầu (Initial State):

Ma trận 3x3 gồm các ô số từ 1 đến 8 và một ô trống (0).

Tập hành động (Actions):

Di chuyển ô trống theo 4 hướng: trái, phải, lên, xuống.

Hàm chuyển trạng thái (Transition Model):

Hoán đổi vị trí ô trống với ô kề.

Kiểm tra mục tiêu (Goal Test):

Trạng thái hiện tại có giống trạng thái đích không?

Hàm đánh giá:

Heuristic

h(n): đo mức gần đích: Tổng khoảng cách Manhattan.

Hoạt động của thuật toán Steepest-Ascent Hill Climbing

Khởi tạo

Bắt đầu từ trạng thái ban đầu.

Lặp lại

• Sinh tất cả hàng xóm hợp lệ của trạng thái hiện tại.

• Chọn hàng xóm tốt nhất (có heuristic h(n) nhỏ nhất).

• Nếu hàng xóm tốt nhất không tốt hơn trạng thái hiện tại → dừng (tụt vào đỉnh cục bộ).

• Cập nhật trạng thái hiện tại = hàng xóm tốt nhất.

• Nếu đạt đích → trả về lời giải.

Kết thúc

Trả về chuỗi hành động hoặc báo thất bại nếu kẹt ở đỉnh cục bộ.

Solution

Trả về chuỗi hành động từ trạng thái ban đầu đến trạng thái đích, đi theo đường leo đồi "dốc nhất" – luôn ưu tiên bước cải thiện mạnh nhất.

Lưu ý:

Không backtrack.

Dễ kẹt tại đỉnh cục bộ hoặc vùng bằng phẳng (plateau).

Có thể lặp lại nhiều lần từ trạng thái khởi tạo khác để tăng cơ hội tìm đích.

Hiệu suất

Ưu điểm:

Đơn giản, dễ cài đặt.

Nhanh, ít tốn bộ nhớ (chỉ cần lưu trạng thái hiện tại).

Khi chạy trên bề mặt đánh giá mượt mà, có thể nhanh chóng hội tụ.

Nhược điểm:

Rất dễ kẹt ở đỉnh cục bộ (local maxima).

Không đảm bảo tìm lời giải tối ưu hoặc tìm thấy lời giải.

Không thể thoát khỏi plateau nếu không có cơ chế bổ sung (như random restart).

Simple Hill Climbing – Leo đồi đơn giản

Simple Hill Climbing là dạng cơ bản nhất của thuật toán Hill Climbing, thuộc nhóm tìm kiếm cục bộ. Thuật toán này chỉ xét một hàng xóm tại một thời điểm (thường theo thứ tự), và chấp nhận ngay nếu nó tốt hơn, thay vì tìm hàng xóm tốt nhất như Steepest-Ascent.

Các thành phần của bài toán tìm kiếm trong 8-Puzzle

Trạng thái ban đầu (Initial State):

Ma trận 3x3 gồm các ô số từ 1–8 và một ô trống (0).

Tập hành động (Actions):

Di chuyển ô trống theo 4 hướng: trái, phải, lên, xuống.

Hàm chuyển trạng thái (Transition Model):

Hoán đổi ô trống với ô lân cận.

Kiểm tra mục tiêu (Goal Test):

So sánh trạng thái hiện tại với trạng thái đích.

Hàm đánh giá:

Heuristic h(n): đo mức gần đích (khoảng cách Manhattan).

Hoạt động của thuật toán Simple Hill Climbing

Khởi tạo

Bắt đầu từ trạng thái ban đầu.

Lặp lại

• Xét tuần tự từng hàng xóm của trạng thái hiện tại.

• Nếu gặp một hàng xóm có heuristic tốt hơn (nhỏ hơn) → chấp nhận ngay và cập nhật trạng thái.

• Nếu không có hàng xóm nào tốt hơn → dừng (tụt vào đỉnh cục bộ).

• Nếu đạt đích → trả về lời giải.

Kết thúc

Trả về chuỗi hành động hoặc báo thất bại nếu kẹt.

Solution

Trả về chuỗi hành động từ trạng thái ban đầu đến trạng thái đích, đi theo các bước cải thiện nhỏ, ngay khi có thể, mà không cần tìm hàng xóm tốt nhất toàn cục.

Lưu ý:

Cải thiện dần, nhưng dễ bị kẹt sớm vì chỉ nhìn hàng xóm đầu tiên tốt hơn, không quan tâm các lựa chọn khác.

Dễ bị kẹt ở đỉnh cục bộ, vùng bằng phẳng (plateau) hoặc sườn dốc (ridge).

Hiệu suất

Ưu điểm:

Cực kỳ đơn giản, dễ cài đặt.

Ít tốn bộ nhớ, chỉ cần lưu trạng thái hiện tại.

Nhược điểm:

Dễ dừng sớm nếu gặp hàng xóm không tốt hơn.

Không tìm được hướng đi tốt hơn nếu đi sai lối.

Không đảm bảo tìm lời giải tối ưu hoặc thậm chí tìm ra đích.

Simulated Annealing (SA) – Leo đồi có làm nguội

Simulated Annealing (SA) là một thuật toán tìm kiếm cục bộ lấy cảm hứng từ quá trình ủ nhiệt trong luyện kim. Khác với các thuật toán Hill Climbing khác, SA cho phép thỉnh thoảng chấp nhận bước lùi (move worse) để thoát khỏi đỉnh cục bộ, với xác suất giảm dần theo thời gian (làm nguội dần).

Các thành phần của bài toán tìm kiếm trong 8-Puzzle

Trạng thái ban đầu (Initial State):

Ma trận 3x3 gồm các ô số từ 1–8 và một ô trống (0).

Tập hành động (Actions):

Di chuyển ô trống trái, phải, lên, xuống.

Hàm chuyển trạng thái (Transition Model):

Hoán đổi vị trí ô trống với ô kề.

Kiểm tra mục tiêu (Goal Test):

So sánh trạng thái hiện tại với trạng thái đích.

Hàm đánh giá (Heuristic h(n)): tổng khoảng cách Manhattan.

Hoạt động của thuật toán Simulated Annealing

Khởi tạo

Bắt đầu từ trạng thái ban đầu.

Đặt nhiệt độ ban đầu T.

Lặp lại cho đến khi $T \approx 0$

• Chọn ngẫu nhiên một hàng xóm của trạng thái hiện tại.

• Tính

$\Delta E = h(\text{current}) - h(\text{neighbor})$:

- Nếu $\Delta E > 0$ → chấp nhận (hàng xóm tốt hơn).

- Nếu $\Delta E \leq 0$ → chấp nhận với xác suất

$\Delta E / T$(cho phép bước lùi).

• Giảm nhiệt độ T theo lịch làm nguội (cooling schedule).

• Nếu đạt đích → trả về lời giải.

Kết thúc

Khi nhiệt độ giảm xuống rất thấp, thuật toán gần như dừng lại (không còn chấp nhận bước lùi).

Solution

Trả về chuỗi hành động từ trạng thái ban đầu đến trạng thái đích, với khả năng "thoát kẹt" nhờ chấp nhận bước lùi có kiểm soát.

Hiệu suất

Ưu điểm:

Có thể thoát khỏi đỉnh cục bộ nhờ bước lùi có xác suất.

Không cần nhớ nhiều trạng thái (bộ nhớ thấp).

Hiệu quả cho không gian trạng thái phức tạp, gồ ghề.

Nhược điểm:

Cần thiết lập lịch làm nguội phù hợp (quá nhanh → kẹt, quá chậm → tốn thời gian).

Không đảm bảo tìm lời giải tối ưu.

Dễ bị kém ổn định nếu hàm đánh giá không trơn tru.

Beam Search

Thuật toán Beam Search là phiên bản giới hạn bộ nhớ của tìm kiếm theo Best-First Search. Tại mỗi bước mở rộng, chỉ giữ lại k nút có giá trị đánh giá tốt nhất (gọi là beam width) thay vì giữ toàn bộ các nút sinh ra. Điều này giúp giảm bộ nhớ và tập trung vào các nhánh có tiềm năng nhất.

Các thành phần của bài toán :

Trạng thái ban đầu (Initial State): Ma trận 3x3 gồm các ô số 1–8 và một ô trống (0). Đảm bảo trạng thái có lời giải.

Tập hành động (Actions): Di chuyển ô trống theo bốn hướng: trái, phải, lên, xuống.

Hàm chuyển trạng thái (Transition Model): Hoán đổi vị trí ô trống với ô kề.

Kiểm tra mục tiêu (Goal Test): So sánh trạng thái hiện tại với trạng thái đích.

Hàm chi phí / Hàm đánh giá: Trong Beam Search, ta thường chỉ dựa vào heuristic h(n) để chọn nút tốt, bỏ qua chi phí g(n).

Hoạt động của thuật toán Beam Search

Khởi tạo

Tạo danh sách beam (beam list) gồm trạng thái ban đầu.

Lặp lại

• Với mỗi nút trong beam hiện tại: mở rộng các trạng thái con.

• Gom tất cả trạng thái con vào một danh sách tạm.

• Chọn ra k trạng thái con có giá trị heuristic tốt nhất (nhỏ nhất nếu là hàm chi phí) để tạo beam mới.

• Nếu trong beam mới có trạng thái đích → trả về lời giải.

Kết thúc

Thuật toán dừng khi tìm thấy đích hoặc không còn nút nào để mở rộng.

Solution

Trả về chuỗi hành động từ trạng thái ban đầu đến đích, theo một trong k nhánh tiềm năng nhất.

Lưu ý: Beam Search không đảm bảo tìm được lời giải tối ưu vì nó có thể loại bỏ nhánh tối ưu nếu không nằm trong beam tại một bước nào đó.

Hiệu suất

Ưu điểm:

Tối ưu hóa bộ nhớ: chỉ lưu k nút tại mỗi bước.

Nhanh hơn so với các thuật toán tìm kiếm toàn cục (như A*) nếu beam width đủ nhỏ.

Thích hợp cho các bài toán lớn hoặc cần kết quả nhanh.

Nhược điểm:

Không đảm bảo tìm được lời giải tối ưu.

Phụ thuộc mạnh vào giá trị beam width – quá nhỏ có thể bỏ lỡ lời giải, quá lớn lại tăng chi phí tính toán.

Hiệu quả phụ thuộc nhiều vào chất lượng heuristic.

Genetic Algorithm

Sử dụng cơ chế chọn lọc tự nhiên (lai ghép, đột biến) để tiến hóa lời giải qua nhiều thế hệ.

+ Genetic Algorithm: không luôn tìm ra lời giải tối ưu, phụ thuộc nhiều vào cách biểu diễn cá thể và hàm fitness; tuy nhiên có thể hiệu quả trong không gian tìm kiếm lớn hoặc không xác định.

+ Trong GA, crossover (lai ghép) và mutation (đột biến) phải đảm bảo sinh ra cá thể hợp lệ.

Nhưng trong 8-puzzle:

→ Nếu "trộn" hai bàn cờ, rất dễ tạo ra trạng thái không thể đạt được qua các bước hợp pháp (vì chỉ một số hoán đổi là hợp lệ).

→ Mutation (thay đổi ngẫu nhiên) cũng dễ tạo ra trạng thái vi phạm ràng buộc.

→ Việc thiết kế các toán tử GA phù hợp để giữ trạng thái hợp lệ rất khó và phức tạp.

2.4 Các thuật toán tìm kiếm Học tăng cường

Q-Learning – Học tăng cường không mô hình

Q-Learning là một thuật toán học tăng cường (Reinforcement Learning) không có mô hình (model-free), giúp tác nhân học được chính sách tối ưu để đạt mục tiêu thông qua thử và sai. Không giống như BFS, Q-Learning sử dụng thông tin từ quá trình tương tác với môi trường để học dần chính sách tốt mà không cần duyệt toàn bộ không gian trạng thái.

Các thành phần của bài toán tìm kiếm trong 8-Puzzle (áp dụng cho Q-Learning)

Trạng thái (State): Tương tự như BFS, mỗi trạng thái là một mảng 3x3 đại diện cho vị trí của 8 ô số và một ô trống (0).

Tập hành động (Actions): Tại mỗi trạng thái, ô trống có thể di chuyển theo 4 hướng: trái, phải, lên, xuống (nếu hợp lệ).

Hàm chuyển trạng thái (Transition Model): Một hành động sẽ tạo ra một trạng thái mới bằng cách hoán đổi ô trống với ô liền kề.

Hàm thưởng (Reward Function):

Thường đặt phần thưởng là -1 cho mỗi bước đi, để khuyến khích đường đi ngắn nhất.

Phần thưởng lớn (ví dụ: +100) nếu đạt được trạng thái đích.

Mục tiêu là tối đa hóa tổng phần thưởng tích lũy.

Trạng thái mục tiêu (Goal State): Trạng thái đích chuẩn của 8-Puzzle

Hoạt động của thuật toán Q-Learning

1. Khởi tạo:

Bảng Q: Lưu giá trị Q cho mỗi cặp (trạng thái, hành động), ban đầu gán bằng 0.

Tham số: alpha (tốc độ học), gamma (hệ số chiết khấu), epsilon (xác suất chọn hành động ngẫu nhiên để khám phá).

2. Lặp lại nhiều tập (episodes):

Đặt trạng thái hiện tại là trạng thái khởi đầu.

Lặp lại cho đến khi đạt trạng thái mục tiêu:

Chọn hành động:

Với xác suất *epsilon*, chọn hành động ngẫu nhiên (khám phá).

Ngược lại, chọn hành động có Q-value cao nhất (khai thác).

Thực hiện hành động → nhận trạng thái mới và phần thưởng.

Cập nhật giá trị Q bằng công thức:

$Q(s, a) \leftarrow Q(s, a) + \alpha * [r + \gamma * \max Q(s', a') - Q(s, a)]$

Cập nhật trạng thái hiện tại.

3. Lặp lại quá trình học cho đến khi giá trị Q hội tụ.

Solution: Sau khi học xong, tác nhân có thể sử dụng bảng Q để tìm lời giải bằng cách bắt đầu từ trạng thái đầu, tại mỗi bước chọn hành động có giá trị Q cao nhất cho đến khi đạt trạng thái đích.

Hiệu suất:

Ưu điểm:

Có khả năng học từ tương tác mà không cần duyệt toàn bộ không gian trạng thái.

Có thể áp dụng cho bài toán lớn hơn (như 15-Puzzle) nếu kết hợp với approximation (deep Q-learning).

Có thể tối ưu theo nhiều mục tiêu khác nhau nhờ hàm thưởng linh hoạt.

Nhược điểm:

Cần rất nhiều tập luyện để học hiệu quả.

Nếu không dùng kỹ thuật biểu diễn trạng thái hiệu quả, bảng Q rất lớn (vì số lượng trạng thái của 8-Puzzle là 9! = 362,880).

Không đảm bảo tìm được lời giải ngắn nhất trừ khi huấn luyện đủ lâu.

2.5 Các thuật toán CSP

Backtracking – Tìm kiếm giải pháp theo ràng buộc

Thuật toán CSP (Problem Satisfaction Problem) tìm kiếm giải pháp cho bài toán bằng cách sử dụng các ràng buộc và backtracking (quay lui). Trong CSP, ta có một tập các biến, mỗi biến có một miền giá trị có thể có, và mục tiêu là tìm một cách gán giá trị cho các biến sao cho tất cả các ràng buộc giữa các biến đều được thỏa mãn.

Các thành phần của bài toán trong 8-Puzzle (CSP)

Biến: Mỗi ô trong ma trận 3x3 của 8-puzzle có thể được coi là một biến. Ví dụ: $x1,x2,...,x9$ tương ứng với các ô trong ma trận.

Miền giá trị (Domain): Mỗi ô có thể chứa một giá trị trong miền {1, 2, 3, ..., 8, 0}, với 0 là ô trống.

Ràng buộc (Constraints): Các ràng buộc này chỉ ra rằng các ô trong ma trận không thể chứa các giá trị trùng nhau, và các phép di chuyển ô trống phải tuân theo các hướng hợp lệ (trái, phải, lên, xuống).

Mục tiêu: Tìm ra một cách gán giá trị cho các biến sao cho trạng thái ban đầu có thể dẫn đến trạng thái đích (ví dụ: 1–2–3 | 4–5–6 | 7–8–0).

Hoạt động của thuật toán CSP – Backtracking

Khởi tạo

Bắt đầu với một trạng thái ban đầu rỗng

Lặp lại

• Chọn một biến: Chọn một ô (biến) chưa có giá trị hợp lệ.

• Giải quyết ràng buộc: Với mỗi giá trị trong miền của ô đó, kiểm tra xem giá trị có vi phạm ràng buộc không (ví dụ: các ô không được trùng nhau).

• Gán giá trị: Nếu không vi phạm, gán giá trị vào ô và tiếp tục.

• Kiểm tra mục tiêu: Nếu tất cả các biến đã được gán giá trị và tất cả các ràng buộc được thỏa mãn, thì ta đã tìm được lời giải.

• Backtracking (Quay lui): Nếu ta không thể gán giá trị hợp lệ cho một ô nào đó, quay lại bước trước và thử các giá trị khác.

Kết thúc

Thuật toán dừng lại khi tìm được lời giải hoặc khi không còn khả năng gán giá trị hợp lệ cho các ô (kết thúc thất bại).

Solution

Trả về một trạng thái trong đó tất cả các ô đều có giá trị hợp lệ, không vi phạm ràng buộc và đạt được mục tiêu (trạng thái đích).

Hiệu suất

Ưu điểm:

Đảm bảo tìm được lời giải nếu tồn tại và thỏa mãn tất cả các ràng buộc.

Phù hợp với các bài toán có ràng buộc phức tạp như 8-puzzle.

Dễ dàng điều chỉnh và mở rộng đối với các bài toán có nhiều ràng buộc khác nhau.

Nhược điểm:

Có thể gặp phải backtracking sâu, dẫn đến thời gian chạy lâu đối với các không gian trạng thái lớn.

Thuật toán có thể bị lặp vô hạn trong các bài toán không có giải pháp hoặc không có chiến lược chọn biến tốt (ví dụ: chọn các ô một cách ngẫu nhiên).

Forward Checking

Forward Checking là một kỹ thuật cải tiến của backtracking giúp tránh việc quay lui không cần thiết bằng cách loại bỏ các giá trị không hợp lệ khỏi miền giá trị của các biến chưa gán. Khi một biến được gán giá trị, thuật toán sẽ kiểm tra trước (forward) các biến còn lại và cập nhật miền giá trị của chúng dựa trên các ràng buộc.

Hoạt động của thuật toán CSP – Forward Checking

Khởi tạo

Khởi tạo trạng thái ban đầu của ma trận 3x3 với tất cả các ô chưa gán giá trị. Miền giá trị ban đầu của mỗi biến là {0, 1, 2, ..., 8}.

Lặp lại

• Chọn một biến: Chọn một ô chưa gán giá trị. Có thể chọn theo thứ tự tuyến tính hoặc sử dụng chiến lược chọn biến tối ưu (như MRV – Minimum Remaining Values).

• Gán giá trị: Thử gán một giá trị từ miền giá trị vào ô đó.

• Kiểm tra ràng buộc:

– Kiểm tra xem giá trị vừa gán có trùng với các giá trị đã được gán trước đó không.

– Cập nhật miền giá trị của các biến chưa gán bằng cách loại bỏ giá trị vừa gán.

– Nếu một biến nào đó còn lại có miền rỗng → quay lui (backtrack).

• Tiếp tục gán biến khác nếu không vi phạm ràng buộc.

• Kiểm tra mục tiêu: Khi tất cả các biến đã được gán và không còn ràng buộc bị vi phạm, trạng thái hợp lệ được tạo ra.

Kết thúc

Thuật toán dừng khi tìm được trạng thái hợp lệ ban đầu (gồm 9 ô với các giá trị duy nhất từ 0 đến 8).

Solution

Trả về một trạng thái 8-puzzle hợp lệ với đầy đủ giá trị và không trùng lặp, có thể tiếp tục dùng làm đầu vào cho thuật toán giải puzzle.

Hiệu suất

Ưu điểm:

Giảm thiểu số bước quay lui nhờ loại bỏ sớm các giá trị không hợp lệ.

Phù hợp với bài toán 8-puzzle có miền nhỏ và ràng buộc đơn giản (giá trị duy nhất).

Nhược điểm:

Tốn tài nguyên để liên tục cập nhật và theo dõi miền giá trị của các biến.

Không tối ưu nếu không kết hợp với chiến lược chọn biến tốt (MRV, Degree heuristic,…).

Min-Conflicts

Min-Conflicts là một thuật toán heuristic được sử dụng để giải bài toán CSP bằng cách bắt đầu với một trạng thái ban đầu ngẫu nhiên và liên tục sửa đổi nó bằng cách giảm số lượng xung đột (conflicts) – tức là các ràng buộc bị vi phạm – cho đến khi không còn xung đột nào.

Hoạt động của thuật toán CSP – Min-Conflicts

Khởi tạo

Bắt đầu với một trạng thái ban đầu ngẫu nhiên, có thể có các giá trị trùng nhau (ví dụ: hai ô cùng chứa số 5).

Lặp lại

• Kiểm tra ràng buộc: Xác định các biến (ô) đang vi phạm ràng buộc (ví dụ: có giá trị trùng).

• Chọn biến xung đột: Chọn một biến đang vi phạm ràng buộc (ví dụ: một ô có giá trị trùng với ô khác).

• Tìm giá trị gây ít xung đột nhất: Gán cho biến một giá trị khác trong miền sao cho số lượng xung đột giảm thiểu. Nếu có nhiều giá trị ngang nhau về xung đột, chọn ngẫu nhiên.

• Lặp lại cho đến khi:

– Tất cả các ô đều có giá trị duy nhất (không còn xung đột).

– Hoặc đạt đến số lần lặp tối đa (kết thúc thất bại).

Kết thúc

Trả về một trạng thái hợp lệ của 8-puzzle nếu tìm được, hoặc báo lỗi nếu không thể sửa hết xung đột sau số lần lặp cho phép.

Solution

Một cấu hình ma trận 3x3 trong đó mỗi giá trị từ 0 đến 8 xuất hiện đúng một lần, có thể dùng làm trạng thái đầu để giải bài toán.

Hiệu suất

Ưu điểm:

Hiệu quả cao trong việc tìm lời giải nhanh khi số lượng ràng buộc không quá phức tạp.

Dễ cài đặt, đặc biệt trong tạo trạng thái đầu hợp lệ.

Nhược điểm:

Không đảm bảo tìm được lời giải nếu trạng thái ban đầu có quá nhiều xung đột.

Có thể rơi vào trạng thái cục bộ (local minima) và lặp lại không hiệu quả.

2.6 Các thuật toán tìm kiếm môi trường phức tạp

No Observation (Không quan sát)

Lý do không áp dụng: Trong bài toán 8-Puzzle, người chơi luôn có thể quan sát toàn bộ trạng thái của ma trận. Mỗi ô trong ma trận có thể được nhìn thấy, và vì vậy, không có môi trường No Observation trong bài toán này. Người chơi hoặc thuật toán luôn có đủ thông tin về trạng thái của game để ra quyết định. Nếu không có khả năng quan sát, người chơi sẽ không thể thực hiện bất kỳ hành động nào có ý nghĩa trong game.

Nondeterministic (Không xác định)

Lý do không áp dụng: Trong 8-Puzzle, mỗi hành động đều dẫn đến một kết quả xác định. Khi bạn di chuyển ô trống theo một hướng (trái, phải, lên, xuống), kết quả luôn là một trạng thái cụ thể và không có sự ngẫu nhiên. Do đó, 8-Puzzle là một môi trường deterministic (xác định), nơi mỗi hành động có một kết quả duy nhất. Nếu có yếu tố ngẫu nhiên trong các hành động (ví dụ, kết quả di chuyển ô trống không xác định), thì bài toán sẽ trở thành môi trường Nondeterministic, nhưng điều này không phải là đặc điểm của bài toán 8-Puzzle truyền thống.

Partial Observation

Partial Observation là môi trường trong đó agent chỉ có thể quan sát một phần của trạng thái môi trường thay vì toàn bộ trạng thái. Điều này tạo ra sự không chắc chắn vì agent phải đưa ra quyết định dựa trên thông tin không đầy đủ. Điều này có thể làm cho việc tìm kiếm giải pháp trở nên khó khăn hơn và đòi hỏi các chiến lược khác nhau để xử lý thiếu sót thông tin.

Các thành phần của bài toán 8-Puzzle:

Trạng thái ban đầu (Initial State): Ma trận 3x3 của 8-Puzzle, trong đó có 8 số và một ô trống (0). Trạng thái ban đầu có thể được tạo ngẫu nhiên hoặc từ một trạng thái đã cho sẵn.

Tập hành động (Actions): Di chuyển ô trống theo bốn hướng: trái, phải, lên, xuống. Mỗi hành động sẽ tạo ra một trạng thái mới.

Hàm chuyển trạng thái (Transition Model): Hoán đổi vị trí của ô trống với ô kề (lên, xuống, trái, phải), tạo ra trạng thái con.

Kiểm tra mục tiêu (Goal Test): So sánh trạng thái hiện tại với trạng thái đích (1-2-3 | 4-5-6 | 7-8-0).

Hàm chi phí / Hàm đánh giá: Thường sử dụng các hàm heuristic như Manhattan Distance để đánh giá mức độ gần với trạng thái đích.

Hoạt động của Partial Observation trong 8-Puzzle:

Khởi tạo:

Tạo một trạng thái ban đầu cho bài toán, có thể là một trạng thái ngẫu nhiên hoặc đã cho.

Lưu trữ thông tin về các ô có thể quan sát được trong trạng thái.

Lặp lại:

Agent chỉ quan sát một phần của trạng thái ma trận (ví dụ, một số ô trong ma trận 3x3).

Dựa trên phần quan sát này, agent đưa ra hành động di chuyển (trái, phải, lên, xuống).

Sau khi thực hiện hành động, trạng thái mới được cập nhật và agent sẽ tiếp tục với quan sát mới.

Nếu trạng thái quan sát đủ thông tin để xác định trạng thái đích, thuật toán dừng lại và trả về lời giải.

Kết thúc: Thuật toán dừng khi trạng thái quan sát đủ thông tin để đạt được trạng thái đích hoặc không còn hành động hợp lệ.

Solution: Trả về chuỗi hành động dẫn từ trạng thái ban đầu đến trạng thái đích. Vì có phần quan sát hạn chế, chính sách tối ưu có thể không được tìm thấy nếu thông tin không đủ để đánh giá đúng các hành động.

Hiệu suất:

Ưu điểm:

Giảm thiểu bộ nhớ: Chỉ cần lưu trữ và tính toán phần trạng thái có thể quan sát, thay vì lưu trữ toàn bộ không gian trạng thái.

Tối ưu hóa tốc độ: Việc chỉ làm việc với một phần của trạng thái giúp giảm bớt khối lượng tính toán.

Nhược điểm:

Không đảm bảo tối ưu: Vì chỉ có thông tin hạn chế, agent có thể đưa ra quyết định sai nếu phần quan sát không đủ.

Khó khăn trong việc khám phá môi trường: Nếu phần quan sát không bao quát đủ không gian trạng thái, agent có thể không tìm ra giải pháp tốt nhất.

Phụ thuộc vào chiến lược quan sát: Nếu cách chọn các ô quan sát không hợp lý, việc tìm kiếm giải pháp có thể không hiệu quả.