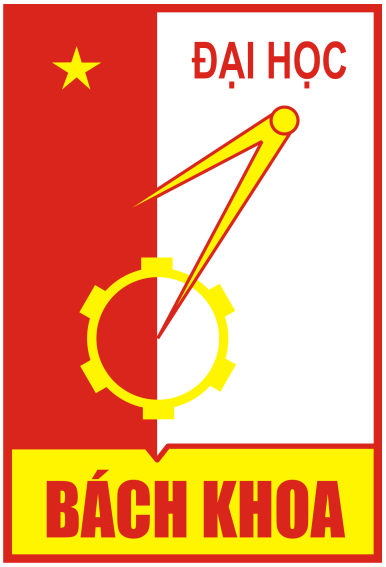
Đại học Bách khoa Hà Nội
Trường công nghệ Thông tin và Truyền thông



Bài tập lớn

Môn: Nhập môn Trí tuệ nhân tạo

Tên đề tài: Ứng dụng Trí tuệ nhân tạo tìm số bước đi tối thiểu thắng trò chơi N-puzzle

Giáo viên hướng dẫn: TS. Nguyễn Nhật Quang

Nhóm sinh viên thực hiện:

Tạ Hữu Bình	20190094
Trần Trọng Hiệp	20190051

Mã lớp: 131403

Mã HP: IT3160

Hà Nội, 1 tháng 7 năm 2022

Mục lục

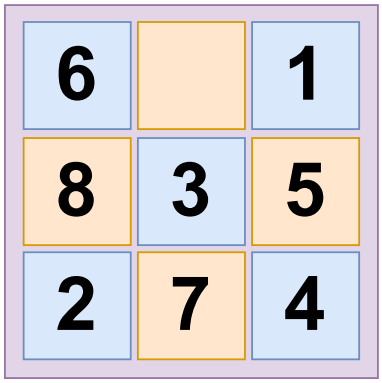
Lời mở đầu

Những năm gần đây, Trí tuệ nhân tạo hay viết tắt là AI (lĩnh vực khoa học và công nghệ nhằm mang lại sự thông minh cho các máy tính, đặc biệt là các chương trình máy tính thông minh) đang có sự phát triển vượt bậc. Không chỉ ở các nước có nền công nghiệp, kinh tế phát triển mà ngay tại Việt Nam, ứng dụng của AI cũng đang xuất hiện ngày càng nhiều, đem đến những tác động tích cực to lớn cho xã hội. Vậy nên, trong bài tập lớn của học phần "Nhập môn Trí tuệ nhân tạo", chúng em chọn đề tài **"Ứng dụng Trí tuệ nhân tạo tìm số bước đi tối thiểu thắng trò chơi N-puzzle"**, gọi tắt là ***bài toán N-puzzle***. Một số thuật toán tìm kiếm, bao gồm cả chiến lược *tìm kiếm cơ bản (Uninformed Search)* và *tìm kiếm với tri thức bổ sung (Informed Search)*, sẽ được áp dụng để giải bài toán và kiểm tra độ hiệu quả thông qua các thực nghiệm.

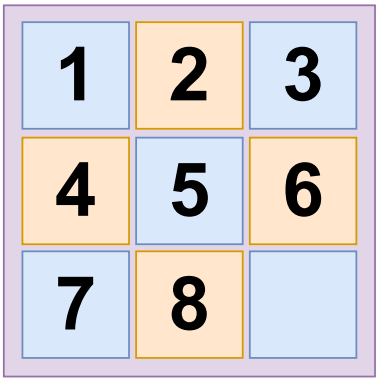
1 Giới thiệu bài toán

1.1 Trò chơi N-puzzle

Trò chơi N-puzzle (N là hiệu của một số chính phương và 1) là một bài toán nổi tiếng, phát biểu đơn giản, dễ hiểu nhưng không dễ để tìm ra lời giải có số bước đi ít nhất (đặc biệt khi xét độ phức tạp tính toán, hoặc thời gian tiêu tốn của các giải thuật ứng dụng). Bài toán đưa ra một bảng vuông, chia thành $(N + 1)$ vùng vuông, $\sqrt{N+1}$ hàng và $\sqrt{N+1}$ cột đều nhau, được phủ bởi N ô vuông nhỏ, ví dụ Hình 1. Các ô vuông kề với vùng trống dư lại có thể được di chuyển để trám chính xác vào vị trí trống đó. Người chơi chiến thắng khi toàn bộ ô vuông ở đúng vị trí yêu cầu, ví dụ Hình 2.



Hình 1: Minh họa trạng thái ban đầu của 8-puzzle.

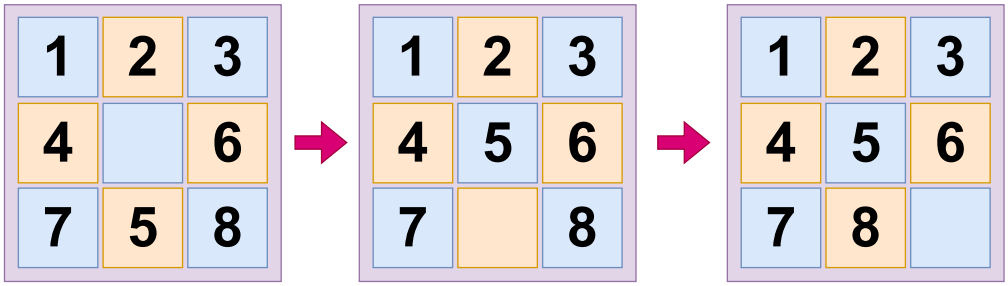


Hình 2: Minh họa trạng thái mục tiêu của 8-puzzle.

1.2 Mô hình bài toán

Việc tím kiếm một lời giải cho bài toán thì tương đối đơn giản và có thể thực hiện trong thời gian đa thức. Báo cáo này sẽ hướng đến mục tiêu tìm kiếm số bước đi ít nhất có thể thực hiện để chiến thắng trò chơi N-puzzle (đã được chứng minh là bài toán NP-khó theo [1]). Bài toán được phát biểu như sau:

- **Đầu vào:** Một bảng vuông, chia thành (N + 1) vùng trống vuông được phủ bởi N ô vuông nhỏ.
- **Đầu ra:** Lời giải để toàn bộ ô vuông ở đúng vị trí yêu cầu.
- **Ràng buộc:** Mỗi bước đi chỉ di chuyển một ô vuông.
- **Mục tiêu:** Tối thiểu hóa số bước đi để chiến thắng, ví dụ Hình 3.



Hình 3: Minh họa chiến thắng với số bước đi tối thiểu.

Không gian tìm kiếm được mô hình dưới dạng một đồ thị. Mỗi đỉnh trong đồ thị ứng với một trạng thái (vị trí của các ô) trong trò chơi. Nếu một trạng thái có thể chuyển đến trạng thái khác trong đúng một bước bước đi, hai đỉnh tương ứng sẽ được nối với nhau. Do đặc tính của trò chơi, đồ thị tạo ra sẽ có chu kỳ và mỗi đỉnh nối được tối đa với 4 đỉnh khác. Với mục tiêu tìm số bước đi ngắn nhất thắng trò chơi N-puzzle, đồ thị xét đến sẽ là đồ thị không trọng số. Phần còn lại của báo cáo, khi đề cập đến **đồ thị** mà không bổ sung thông

tin gì thêm, chúng ta sẽ hiểu với nghĩa **đồ thị không trọng số**.

2 Thuật toán đề xuất

Phần này của báo cáo sẽ trình bày nội dung lý thuyết của hai chiến lược tìm kiếm cơ bản hay *uninformed search* (tìm kiếm theo chiều rộng, tìm kiếm sâu dần) và hai chiến lược tìm kiếm với tri thức bổ sung hay *informed search* (tìm kiếm tham lam, A*) cùng các hàm heuristic dùng trong các thuật toán với tri thức bổ sung.

2.1 Thuật toán tìm kiếm theo chiều rộng - BFS

Tìm kiếm theo chiều rộng là thuật toán tìm kiếm không dựa trên kinh nghiệm trong đồ thị. Bắt đầu từ một điểm trong đồ thị (điểm nguồn), thuật toán sẽ duyệt qua tất cả các đỉnh, trong đó ưu tiên những đỉnh có đường đi ngắn nhất (theo nghĩa là số điểm ở trên đường nối ít nhất) đến đỉnh nguồn. Vì vậy, nếu tồn tại đường đi ngắn nhất giữa điểm nguồn và điểm đích trong đồ thị không trọng số, thuật toán *tìm kiếm theo chiều rộng* luôn chỉ ra được đường đi đó khi không bị giới hạn bởi thời gian hay bộ nhớ lưu trữ. Với bài toán N-puzzle, khi lần đầu duyệt đến trạng thái đích của trò chơi, chúng ta cũng đồng thời có được số bước đi ngắn nhất để chiến thắng. Nói cách khác, thuật toán đảm bảo cả chính hoàn chỉnh và tính tối ưu.

Thuật toán cài đặt trong báo cáo được mô tả như sau:

- Bước 1: Khởi tạo.
  - Các đỉnh đều chưa được đánh dầu, trừ đỉnh nguồn.
  - Một hàng đợi ban đầu chỉ chứa đỉnh nguồn.
  - Một tập để lưu trữ các đỉnh đã thăm
- Bước 2: Duyệt qua các đỉnh.
  - Lấy một đỉnh u ra khỏi hàng đợi.
  - Thêm đỉnh u vào tập lưu các điểm đã thăm.
  - Đỉnh u sẽ mang thông tin đỉnh cha (đỉnh duyệt trước đó nối đến nó), bước đi đã thực hiện để từ đỉnh cha (lên, xuống, trái, phải) đến được nó và số bước từ đỉnh nguồn đến nó.
  - Thêm các đỉnh kề với u và chưa được thăm (không có trong tập lưu các điểm đã thăm) vào hàng đợi.

  - Nếu hàng đợi rông, chuyển đến bước 3. Ngược lại, vòng lại bước 2.

- Bước 3: Truy vết đường đi (Tùy chọn).

2.2 Thuật toán tìm kiếm sâu dần - IDS

Thuật toán *tìm kiếm sâu dần* được phát triển từ thuật toán *tìm kiếm theo chiều sâu* (DFS) và *tìm kiếm giới hạn độ sâu* (DLS).

Thuật toán DFS bắt đầu từ một điểm trong đồ thị (điểm nguồn) và sau đó phát triển sâu và xa nhất của mỗi nhánh. Tuy nhiên nếu lời giải ở rất sâu và vì số cấu hình rất lớn theo kích thước của đồ thị nên sẽ tốn rất nhiều thời gian để có thể tìm được lời giải tối ưu theo thuật toán. Bên cạnh đó đồ thị là độ thị liên thông nên có khả năng lời giải nằm ngay bên cạnh, tuy nhiên do thuật toán DFS nên ta phải phát triển việc tìm kiếm lời giải xuống điểm sâu nhất khiến cho việc tìm kiếm lời giải rất khó khăn. Thuật toán DLS có thể khắc phục điều này, thay vì mình tìm kiếm đến độ sâu cuối cùng của lời giải, thì thuật toán sẽ tìm kiếm đến một độ sâu nhất định và khi đến độ sau đó thuật toán sẽ không tìm kiếm xuống dưới nữa mà sẽ tìm kiếm sang nhánh còn lại. Tuy nhiên nếu lời giải mà năm sâu hơn giới hạn của thuật toán DLS, thuật toán sẽ không có lời giải. Để giải quyết điều này, thuật toán IDS sẽ tăng dần độ sâu lên cho đến khi có thể tìm được lời giải tối ưu. Thuật toán này khi đạt đến độ sâu nhất định thì sẽ tìm kiếm sang ngang nên với lời giải về số bước thực hiện để đến được trạng thái mục tiêu của thuật toán BFS thì sẽ luôn tìm được lời giải cho thuật toán IDS ở độ sâu đấy, nên lời giải của thuật toán toán IDS cho số bước đến được trạng thái mục tiêu sẽ bằng so với thuật toán BFS. Từ đây ta có thể thấy được thuật toán IDS luôncó tính hoàn chỉn và tối ưu.

Thuật toán IDS trong báo cáo được được mô tả như sau:

- Bước 1: Khởi tạo giới hạn độ sâu ban đầu bằng khoảng cách Manhattan của đồ thị hiện tại so với đồ thị mục tiêu

- Bước 2: Với mỗi giới hạn độ sâu:

  - Khởi tạo giới hạn độ sâu ban đầu bằng khoảng cách Manhattan của đồ thị hiện tại so với đồ thị mục tiêu

  - Một hàng đợi ngăn xếp ban đầu chỉ chứa đỉnh nguồn

  - Một tập lưu trữ các đỉnh đã thăm và số bước đi từ đỉnh nguôn tới đỉnh đấy, ban đầu chỉ chứa điểm nguồn cùng với số bước đi từ đỉnh nguồn tới đỉnh đấy là 0.

- Bước 3: Duyệt qua các đỉnh

- Lấy một đỉnh u ra khỏi ngăn xếp.
- Đỉnh u sẽ mang thông tin đỉnh cha (đỉnh duyệt trước đó nối đến nó), bước đi đã thực hiện để từ đỉnh nguồn (lên, xuống, trái, phải) đến được nó và số bước từ đỉnh nguồn đến nó.
- Thêm các đỉnh kề với u và chưa được thăm (không có trong tập lưu các điểm đã thăm và số bước từ đỉnh nguồn đến nó, nếu có đỉnh trùng nhưng số bước đến đỉnh đó khác nhau thì đỉnh đó chưa được thăm) cùng với số bước được thực hiện từ đỉnh cha đến u vào ngăn xếp.
- nếu ngăn xếp rỗng, tăng giới hạn độ sâu thêm 1 vào chuyển đến bước 2, ngược lại vòng lại bước 3

- Bước 4: Truy vết đường đi (tùy chọn)

Với mỗi trạng thái, khi lưu trạng thái đó cùng với số bước đi từ nguồn đến trạng thái đó sẽ giúp ta tráng được tình trạng đường đi của ta sẽ đến 2 trạng thái này ở 2 nhánh khác nhau, nhánh thứ nhất của độ sâu sâu hơn nhánh thứ 2 tuy nhiên trạng thái ở nhánh thứ nhất chưa tìm được lời giải tối ưu.

2.3 Thuật toán tìm kiếm tham lam (Greedy Best-first search)

Các chiến lược tìm kiếm cơ bản có tính tổng quát hóa cao, áp dụng được cho nhiều bài toán tìm kiếm miễn là mô hình hóa được dưới dạng đồ thị. Tuy nhiên, "tối ưu các định lý không có bữa trưa miễn phí" (no free lunch theorems for optimization), ứng với từng vấn đề cụ thể sẽ có thể có nhiều cách tiếp cận tiết kiệm hơn về mặt thời gian hoặc bộ nhớ. Bài toán tìm lời giải có số bước đi ngắn nhất cho bài toán N-puzzle cũng không phải một ngoại lệ. Các chiến lược **tìm kiếm với tri thức bổ sung**, khi sử dụng hợp lý các thông tin về đặc tính của vấn đề cần giải quyết, có thể đem lại kết quả tìm kiếm hiệu quả hơn nhiều lần. Một trong những cách tiếp cận cơ bản nhất là thuật toán *tìm kiếm tham lam* (Greedy Best-first search).

Ý tưởng của cách tiếp cận tham lam này là sử dụng một hàm đánh giá $f(n)$ ước lượng độ "phù hợp" của từng đỉnh tương ứng. Trong quá trình tìm kiếm, ưu tiên duyệt các đỉnh có giá trị hàm đánh giá cao nhất. Cụ thể hơn, đây là hàm heuristic $h(n)$ - ước lượng chi phí để đi từ đỉnh đang xét đến đỉnh mục tiêu. Trong bài toán N-puzzle, hàm heuristic sẽ ước lượng số bước đi ít nhất để thắng được trò chơi. Thuật toán tham lam không đảm bảo được cả tính hoàn chỉnh lẫn tính tối ưu.

2.4 Thuật toán A*

A* cũng là một thuật toán với chiến lược tìm kiếm theo tri thức bổ sung. Ý tưởng chính ở đây là tránh phát triển các nhánh tìm kiếm đã xác định tính đến hiện thời mà có chi phí cao. Hàm đánh giá $f(n)$ bây giờ sẽ được định nghĩa là tổng của hàm $g(n)$ tính chi phí từ đỉnh nguồn đến đỉnh đang xét và hàm $h(n)$ ước lượng chi phí từ đỉnh đang xét đến đỉnh đích. Do đó, $f(n)$ sẽ ước lượng tổng chi phí của một lời giải mà qua điểm đang xét. Nói chung, thuật toán A* có các ba đặc điểm nổi bật sau:

- Nếu không gian các trạng thái là hữu hạn và có cách tiếp cận xử lý được vấn đề lặp trạng thái, thì thuật toán là hoàn chỉnh, nhưng không đảm bảo được tính tối ưu.
- Nếu không gian các trạng thái là hữu hạn và không thể tránh việc xét lặp, thì thuật toán không đảm bảo được cả việc tìm được lời giải.
- Nếu không gian trạng thái là vô hạn, thì giải thuật không hoàn chỉnh.

Không gian các trạng thái của bài toán N-puzzle mô hình trong báo cáo là hữu hạn. Thuật toán A* cũng sử dụng một tập lưu trữ các đỉnh đã được ước lượng để tránh việc xét lặp. Do đó, chúng ta luôn tìm được chiến lược để chiến thắng trò chơi N-puzzle.

Một ước lượng heuristic được gọi là *chấp nhận được* nếu giá trị ước lượng đưa ra luôn không âm và cũng không vượt quá chi phí thực tế. Do đó, một ước lượng chấp nhận được sẽ không bao giờ gặp phải hiện tượng đánh giá quá cao chi phí để đi đến đích. Định lý sau đã được chứng minh: "Nếu $h(n)$ là đánh giá cháp nhận được thì phương pháp tìm kiếm A* sử dụng giải thuật tìm kiếm theo cấu trúc cây".

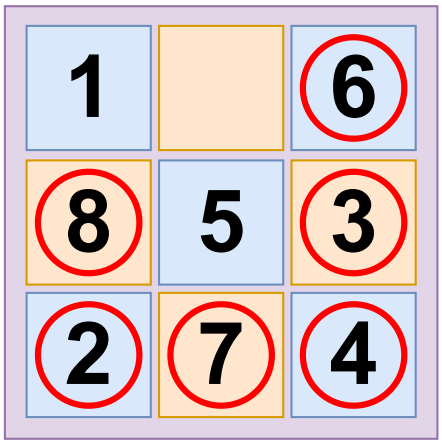
Xét hai ước lượng chấp nhận được, ước lượng nào luôn có giá trị bằng hoặc cao hơn thì được gọi là ưu thế hoặc vượt trội hơn. Một ước lượng được xem là kiên định nếu giá trị ước lượng cho một đỉnh không bao giờ vượt quá tổng của chi phí thật từ đỉnh đó đến đỉnh tiếp theo và giá trị ước lượng của đỉnh tiếp theo. Định lý sau đã được chứng minh: "Nếu $h(n)$ là đánh giá chấp nhận được thì phương pháp tìm kiếm A* sử dụng giải thuật tìm kiếm trên cấu trúc đồ thị là tối ưu".

2.5 Các hàm heuristic

Phần này sẽ trình bày các hàm heuristic được sử dụng trong thuật toán A* của báo cáo.

2.5.1 Số ô sai vị trí (khoảng cách Hamming)

Hàm heuristic "đếm" số ô sai vị trí (hay còn gọi là *khoảng cách Hamming*) là một hàm ước lượng chấp nhận được. Mỗi bước di chuyển, chỉ có duy nhất một ô (không tính khoảng trống) được thay đổi vị trí, do đó số bước di chuyển tối thiểu luôn không ít hơn giá trị của hàm heuristic này.



Hình 4: Minh họa trạng thái có 6 ô bị lệch vị trí so với trạng thái đích.

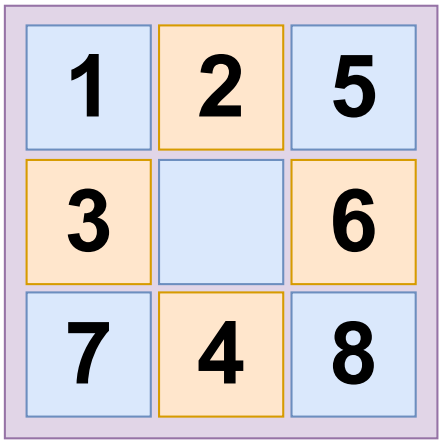
Ví dụ trong Hình 4, có 6 ô bị sai vị trí, là các ô được khoanh đỏ.

2.5.2 Manhattan và Manhattan trọng số

Khoảng cách Manhattan của một ô là số bước di chuyển (số lần trượt) ô đó tối thiểu nếu nó không bị ngáng đường bởi một ô khác. Đây chính là khoảng cách theo chuẩn 1. Hàm heuristic Manhattan sẽ trả về giá trị là tổng khoảng cách Manhattan của mỗi ô đến vị trí đích cần đạt được, trừ ô trống. Mỗi bước di chuyển, chỉ có duy nhất một ô (không tính khoảng trống) được thay đổi vị trí, hàm Manhattan chỉ giảm tối đa được 1 đơn vị. Vậy nên, đây là hàm ước lượng chấp nhận được. Thêm nữa, khoảng cách Manhattan tương ứng của các ô đúng vị trí là 0, của các ô sai vị trí là từ 1 trở lên, do đó hàm Manhattan vượt trội hơn hàm ước lượng heuristic đo khoảng cách Hamming.

Ví dụ trong Hình 5, các ô 5, 3, 4, 8 bị sai vị trí nên ta sẽ tính khoảng cách Manhattan tương ứng của các ô này đến vị trí muốn đạt được. Kết quả lần lượt là 2, 3, 2, 1 dẫn đến giá trị hàm Manhattan trong trạng thái này là 8. Trong khi đó, hàm đo khoảng cách Hamming chỉ trả về giá trị 6.

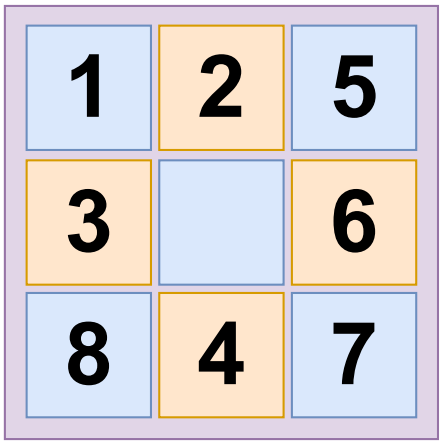
Hàm Manhattan trọng số sẽ nhân giá trị ước lượng Manhattan với một hằng số bất kỳ (trong phần thực nghiệm của báo cáo đều sử dụng hằng số là 1.1). Việc này có thể giúp giá trị ước lượng gần với thực tế hơn, nhưng lại không đảm bảo là chấp nhận được.



Hình 5: Minh họa trạng thái có giá trị hàm Manhattan là 8.

2.5.3 Mâu thuẫn tuyến tính

Hai ô được gọi là mâu thuẫn tuyến tính (linear conflict) nếu chúng và ô đích tương ứng ở cùng hàng hoặc cùng cột, đồng thời muốn di chuyển đến đích, hai ô phải được "di chuyển qua nhau". Ví dụ trong hình 6, ô 7 và ô 8 tạo lên một cặp mâu thuẫn tuyến tính. Trong một cặp mâu thuẫn tuyến tính, bắt buộc có ít nhất một ô phải chuyển sang hàng hoặc cột khác, tức là phải mất thêm ít nhất hai bước di chuyển. Hàm ước lượng heuristic bây giờ sẽ là tổng của hàm Manhattan và 2 lần số cặp mâu thuẫn tuyến tính.



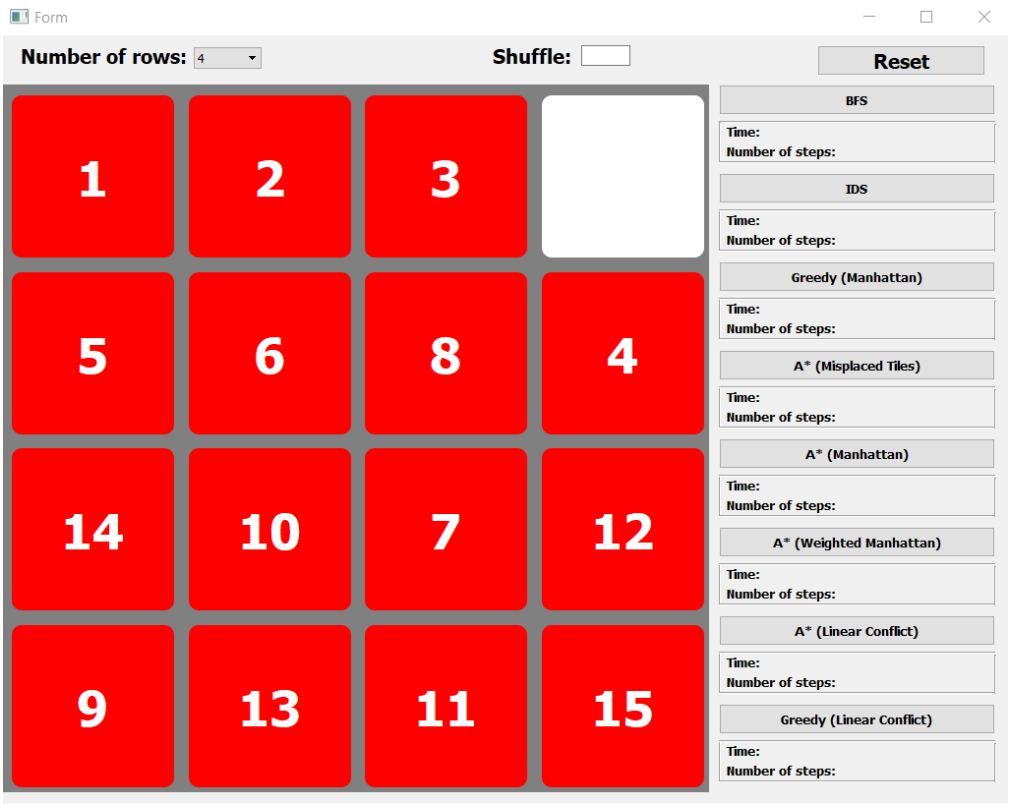
Hình 6: Ô chứa số 7 và 8 tạo thành một cặp mâu thuẫn tuyến tính.

3 Xây dựng chương trình

Ngôn ngữ sử dụng để cài đặt chương trình là Python.

Giao diện chương trình được xây dựng bằng thư viện PyQt5. Trong đó, PyQt nói chung là một liên kết Python cho Qt (một tập hợp các thư viện C++ và các công cụ phát triển

gồm có các bản tóm tắt độc lập nền tảng cho Giao diện người dùng đồ họa cũng như mạng, luồng, biểu thức chính quy,...)



Hình 7: Giao diện chương trình khi mới bật lên.

Để dịch chuyển vị trí của ô trống, các phím được sử dụng là **A** (sang trái), **D** (sang phải), **W** (lên trên) và **S** (xuống dưới). Phím **B** được sử dụng để thực hiện lại các bước đi có được từ thuật toán vừa chạy cho đến đích. Phím **U** được sử dụng để quay lại trạng thái ban đầu.

Hình 8: Các thành phần chính của chương trình.

Các thành phần chính của chương trình được thể hiện trong Hình 8:

- **Number of rows**: Điền số hàng của trò chơi.
- **Shuffle**: Điền số bước tráo từ cấu hình đích.
- Nút **Reset** dùng để tạo trạng thái mới của trò chơi theo số hàng và số bước tráo đã điền.
- Bấm các nút có tên thuật toán mong muốn để tiến hành chạy thuật toán. Khi thuật toán chạy xong sẽ hiển thị kết quả ngay bên dưới:
  - **Time**: Thời gian chạy thuật toán.
  - **Number of steps**: Số bước đi đạt được.

4 Kết quả thực nghiệm

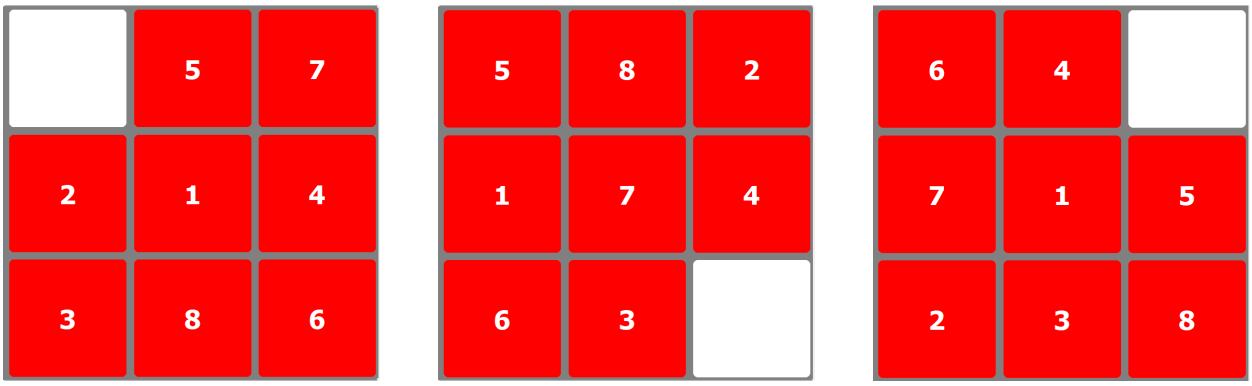
Phần này của báo cáo sẽ tiến hành so sánh độ hiệu quả của các thuật toán đề xuất, bao gồm số bước đi đạt được và thời gian ra kết quả đó. Ba kịch bản chính bao gồm trò chơi 8-puzzle, 15-puzzle, 24-puzzle tương ứng với kích thước 3 x 3, 4 x 4 và 5 x 5. Mỗi thuật toán được chạy 5 lần ứng với từng kịch bản để tính thời gian ra kết quả trung bình. Cấu hình máy tính thực hiện được cho trong bảng dưới đây:

Thông số máy	Giá trị	Đơn vị
Dung lượng RAM	8	GB
Tốc độ đọc ghi RAM	2667	MHz
Tốc độ CPU cơ sở	1.99	GHZ
Số nhân CPU	4	Nhân
L1 cache	256	KB
L2 cache	1.0	MB
L3 cache	8.0	MB

Bảng 1: Cấu hình của máy tiến hành thực nghiệm

4.1 Kịch bản 8-puzzle

. Thực nghiệm được tiến hành trên 3 cấu hình 8-puzzle khác nhau (Hình 9). Các giải thuật sẽ được so sánh và đối chiếu kết quả với thuật toán tìm kiếm theo chiều rộng (đã được chứng minh là luôn đưa ra số bước đi ít nhất để chiến thắng trò chơi).



(a) 1a (b) 1b (c) 1c

Hình 9: Ba trạng thái khởi đầu trò chơi 8-puzzle.

Từ Bảng 2 có thể thấy, các chiến lược tìm kiếm với tri thức bổ sung nói chung có thời gian chạy nhanh hơn nhiều so với hai chiến lược tìm kiếm cơ bản. Trong đó, mặc dù đưa ra kết quả nhanh vượt trội trong đa số các trường hợp, hai thuật toán tham lam không thể đem lại lời giải có số bước đi tối thiểu trong cả 3 kịch bản. Thuật toán IDS cũng luôn có thời gian chạy lâu nhất, mặc dù số bước đi luôn tối ưu giống như BFS nhưng thời gian để ra kết quả có thể cao gấp hơn 10 lần như ở kịch bản 1a. Có thể thấy, BFS là cách tiếp cận theo chiến lược tìm kiếm cơ bản phù hợp hơn, thậm chí thời gian đưa ra kết quả trong kịch bản 1a lẫn 1c còn cao hơn cả thuật toán A* với hàm heuristic là *số ô sai vị trí*. Sử dụng hàm heuristic này trong kịch bản 1b mang lại thời gian tương đối thấp, nhưng vẫn thua so với tất cả các thuật toán dựa trên kinh nghiệm khác. Manhattan có trọng số, dù không phải là hàm heuristic chấp nhận được, trong các kịch bản này đều là hàm ước lượng hiệu quả nhất ứng dụng cho thuật toán A*, thậm chí trong kịch bản 1b còn có thời gian chạy tốt tương đương

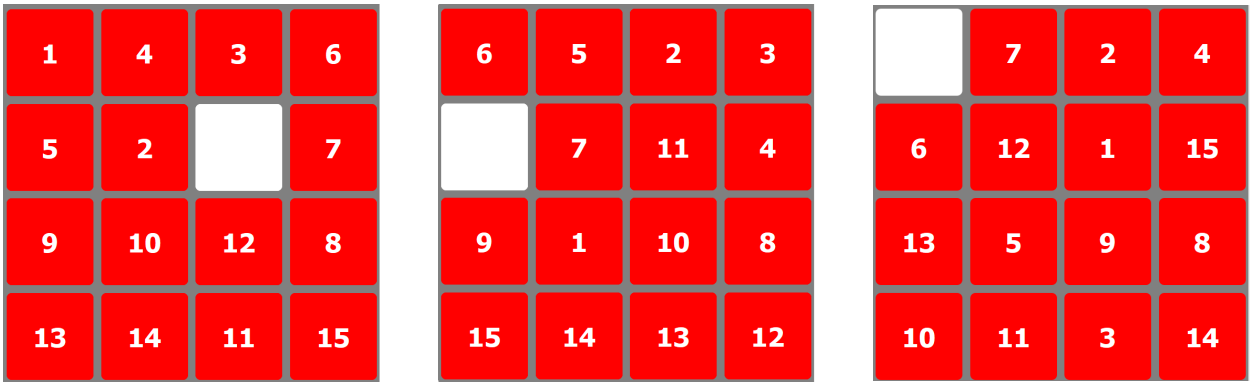
với giải thuật tham lam (khoảng 9 ms). Hàm *Mâu thuẫn toán tính* khi dùng trong trò chơi 9-puzzle đem lại hiệu quả chưa đáng kể. Điều này có thể được giải thích là do không gian lời giải còn ít, "độ khó" của bài toán còn chưa cao, các hàm heuristic đơn giản khác vì thế vẫn xử lý khá tốt. Thêm nữa, vì phức tạp nên thời gian đánh giá của hàm *mâu thuẫn tuyến tính* cũng cao hơn đáng kể. Tóm lại, với bài toán 8-puzzle, chiến lược tìm kiếm cơ bản, cụ thể là BFS vẫn tương đối hiệu quả; để tăng tốc độ đưa ra lời giải có thể sử dụng thuật toán A* với hàm Manhattan hoặc Manhattan trọng số.

Thuật toán	**Kịch bản 1a**		**Kịch bản 1b**		**Kịch bản 1c**	
	Thời gian (s)	Số bước	Thời gian (s)	Số bước	Thời gian (s)	Số bước
BFS	1.90	26	0.66	20	2.05	26
IDS	24.96	26	2.36	20	20.06	26
Tham lam (Manhattan)	0.007	34	0.012	40	0.05	48
A* (số ô sai vị trí)	4.9	26	0.28	20	3.81	26
A* (Manhattan)	0.65	26	0.014	20	0.42	26
A* (Manhattan trọng số)	0.46	26	0.009	20	0.23	26
A* (mâu thuẫn tuyến tính)	1.37	26	0.024	20	0.56	26
Tham lam (mâu thuẫn tuyến tính)	0.07	26	0.009	40	0.03	50

Bảng 2: Kết quả thực nghiệm trên trò chơi 8-puzzle

4.2 Kịch bản 15-puzzle

Thực nghiệm được tiến hành trên 3 cấu hình 15-puzzle (Hình 10) có số lần tráo tăng dần. Các thuật toán chạy quá 10 phút sẽ không có kết quả ghi trong bảng.



(a) 2a (b) 2b (c) 2c

Hình 10: Ba trạng thái khởi đầu trò chơi 15-puzzle.

Bảng 3 cho thấy hai thuật toán tìm kiếm dựa trên kinh nghiệm không còn phù hợp để giải bài toán 15-puzzle vì thời gian chạy hai thuật toán nói chung là quá lớn, và cao hơn hẳn các thuật toán còn lại. Thêm nữa, hàm heuristic *số ô sai vị trí* do đưa ra giá trị ước lượng nói chung thấp hơn nhiều so với thực tế nên trong hai kịch bản 2b và 2c đều mang lại thời gian chạy lớn hơn 10 phút. Trong kịch bản đơn giản 2a, hàm Manhattan thuần túy đang mang lại hiệu quả cao nhất (chỉ 42 ms),tốt hơn so với cả hai thuật toán tham lam. Rất có thể trong trường hợp này, hai thuật toán tham lam do có hàm ước lượng tổng thể không "hợp lý" bằng thuật toán A* nên khi tìm kiếm dễ bị "đi sai đường" hơn. Trong hai kịch bản sau thì cách tiếp cận tham lam đã lại tiếp tục phát huy được ưu thế về mặt thời gian, trong đó hàm ước lượng *mâu thuẫn tuyến tính* luôn trả về kết quả sớm hơn, mặc dù không phải lúc nào số bước trả về cũng thấp hơn Manhattan. Cũng trong hai kịch bản sau thì ưu điểm của hàm ước lượng *mâu thuẫn tuyến tính* đã được thể hiện, nhất là trong 2c khi thuật toán A* sử dụng hàm này mang lại kết quả tốt nhất ở mọi tiêu chí. Trong 2b, hàm ước lượng Manhattan mang lại thời gian chạy không phải cao nhất nhưng số bước đạt được đang thấp nhất. Trong 3c, mặc dù không phải ước lượng chấp nhận được, Manhattan trọng số đem lại số bước chạy không bị thấp hơn quá nhiều so với Manhattan. Kết luận, chiến lược tìm kiếm cơ bản đã không còn phù hợp với 15-puzzle, hàm mâu thuẫn tuyến tính sẽ mang lại hiệu quả tốt hơn so với các cách tiếp cận khác khi độ khó của trò chơi tăng lên.

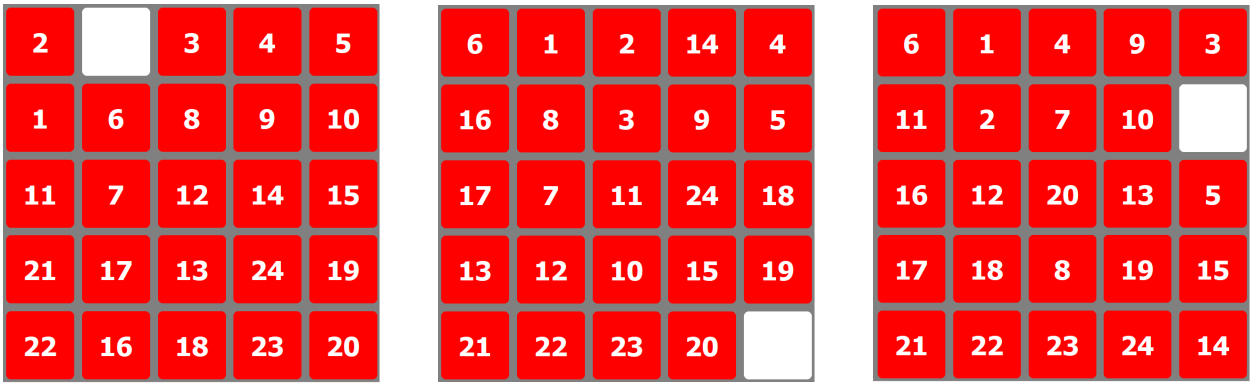
Thuật toán	Kịch bản 2a		Kịch bản 2b		Kịch bản 2c	
	Thời gian (s)	Số bước	Thời gian (s)	Số bước	Thời gian (s)	Số bước
BFS	13	17	> 600	x	> 600	x
IDS	20	17	> 600	x	> 600	x
Tham lam (Manhattan)	0.13	39	2.2	95	9.6	128
A* (số ô sai vị trí)	0.18	17	> 600	x	> 600	x
A* (Manhattan)	0.042	17	5.5	31	82	40
A* (Manhattan trọng số)	0.067	17	4.2	31	72.2	44
A* (mâu thuẫn tuyến tính)	0.061	17	4.4	31	57.2	40
Tham lam (mâu thuẫn tuyến tính)	0.077	39	0.25	57	7.45	132

Bảng 3: Kết quả thực nghiệm trên trò chơi 15-puzzle

4.3 Kịch bản 24-puzzle

Thực nghiệm được tiến hành trên 3 cấu hình 24-puzzle (Hình 11), trong đó cấu hình đầu tiên đơn giản hơn so với hai cấu hình còn lại. Các thuật toán chạy quá 10 phút sẽ không có kết quả ghi trong bảng.

Có thể thấy trong Bảng 4, hai thuật toán tìm kiếm cơ bản và thuật toán A* sử dụng hàm heuristic đơn giản đếm số ô sai vị trí không thể đưa ra được kết quả trong vòng 10 phút. Trong kịch bản 3a, hai thuật toán tham lam đưa ra lời giải có chỉ có số bước nhiều hơn 2 so với lời giải tối ưu, nhưng thời gian chạy ngắn nhất lại là thuật toán A* sử dụng hàm ước lượng heuristic Manhattan trọng số. Trong cả 3 kịch bản, hàm ước lượng Manhattan trọng số luôn mang lại lời giải có số bước đi bằng với hàm Manhattan truyền thống khi áp dụng với thuật toán A*. Trong khi đó, hàm *mâu thuẫn tuyến tính* bộc lộ những vấn đề khi số



(a) 3a (b) 3b (c) 3c

Hình 11: Ba trạng thái khởi đầu trò chơi 24-puzzle.

hàng tăng lên 5. Cụ thể, trong trường hợp 3b, thời gian chạy thuật toán A* bị tăng lên gấp gần 9 lần so với khi sử dụng hàm Manhattan, còn thuật toán tham lam trong trường hợp 3c còn không ra được kết quả trong vòng 10 phút. Điều này có lẽ do, đôi khi số lượng cặp mâu thuẫn tuyến tính của một trạng thái (tạm gọi là m) cao hơn trạng thái khác (tạm gọi là p), nhưng có thể trạng thái m lại nhanh chóng đạt được đến vị trí đích hơn. Thực tế khi sự xáo trộn nhiều, rất khó để tồn tại các ô chỉ cần đi thẳng là đến đích. Nếu chỉ ước lượng một cách đơn giản là thêm vào giá trị ước lượng số bước phải di chuyển của một ô trong cặp mâu thuẫn tuyến tính, trạng thái m có xu hướng bị ước lượng số bước đi tới đích càng kém lạc quan hơn so với p, khiến việc tìm lời giải bị "vòng" qua nhiều trạng thái tồi hơn. Vậy nên, trong trò chơi 24-puzzle, thuật toán Manhattan có trọng số hoặc Manhattan thông thường nên được sử dụng.

Thuật toán	Kịch bản 3a		Kịch bản 3b		Kịch bản 3c	
	Thời gian (s)	Số bước	Thời gian (s)	Số bước	Thời gian (s)	Số bước
BFS	53	17	> 600	x	> 600	x
IDS	73	17	> 600	x	> 600	x
Tham lam (Manhattan)	0.62	19	41	168	19	63
A* (số ô sai vị trí)	0.02	17	> 600	x	> 600	x
A* (Manhattan)	0.013	17	129	44	95	37
A* (Manhattan trọng số)	0.012	17	35	44	53	37
A* (mâu thuẫn tuyến tính)	0.037	17	300	46	40	37
Tham lam (mâu thuẫn tuyến tính)	0.016	19	21	196	> 600	x

Bảng 4: Kết quả thực nghiệm trên trò chơi 24-puzzle

5 Kết luận

Trong quá trình làm bài tập lớn, nhóm đã thực hiện được các công việc sau:

- Tìm hiểu bài toán N-puzzle và mô hình hóa dưới dạng bài toán tìm kiếm trên đồ thị.
- Tìm hiểu các cách tiếp cận để giải bài toán.
- Cài đặt và tìm cách tối ưu thời gian chạy cho các giải thuật đề xuất.
- Xây dựng phần mềm để mô phỏng lời giải các thuật toán.
- Tiến hành các thực nghiệm để đo độ hiệu quả của các cách tiếp cận.
- Phân tích, bàn luận về các kết quả đạt được.

Các khó khăn gặp phải gồm:

- Mới tiếp cận thư viện PyQt5 để làm giao diện nên trong quá trình lập trình gặp nhiều lỗi, chẳng hạn như chương trình không thể "nghe" được các phím nhập vào.
- Quá trình tối ưu mất nhiều thời gian, phải tìm hiểu, tham khảo cách dùng các cấu trúc dữ liệu trong Python để chương trình chạy nhanh hơn.

Trong tương lai, nhóm muốn tối ưu thuật toán IDA* và thử nghiệm các thuật toán khác như là *Recursive best-first search*. Vấn đề về bộ nhớ cũng sẽ được quan tâm.

Để hoàn thành được báo cáo này, nhóm chúng em xin gửi lời cảm ơn sâu sắc đến TS. Nguyễn Nhật Quang vì những tiết dạy tuyệt vời và đáng quý của thầy ở môn "Nhập môn Trí tuệ nhân tạo".

Toàn bộ mã nguồn trong báo cáo được cho trong link:

https://github.com/Tahuubinh/AI_HUST_Project

6 Tài liệu tham khảo

[1] Ratner, Daniel, and Manfred K. Warmuth. "Finding a Shortest Solution for the N × N Extension of the 15-PUZZLE Is Intractable." AAAI. 1986.

[2] Mathew, Kuruvilla, Mujahid Tabassum, and Mohana Ramakrishnan. "Experimental comparison of uninformed and heuristic AI Algorithms for N puzzle solution." Proceedings of the International Journal of Digital Information and Wireless Communications, Hongkong, China (2013): 12-14.

[3] https://algorithmsinsight.wordpress.com/graph-theory-2/a-star-in-general/implementing-a-star-to-solve-n-puzzle

[4] https://stackoverflow.com/questions/49003059/n-puzzle-a-star-python-solver-efficiency

[5] https://www.geeksforgeeks.org/iterative-deepening-searchids-iterative-deepening-depth-first-searchiddfs

[6] https://www.code-learner.com/how-to-develop-a-digital-n-puzzle-game-with-python