

**TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI**

**KHOA CÔNG NGHỆ THÔNG TIN**

**======\*\*\*======**

A yellow and red logo

Description automatically generated

**BÁO CÁO BÀI TẬP LỚN HỌC PHẦN**

**TRÍ TUỆ NHÂN TẠO**

**Đề tài**

**Tìm hiểu thuật toán tìm kiếm heuristic và ứng dụng**

**vào bài toán trò chơi 8 số**

Giảng viên hướng dẫn: **Lê Thị Thủy**

Nhóm – Lớp: 8 - 20231IT6043002

Thành viên: Hà Thị Huệ 2021607262

Bùi Huy Nam 2021603337

Lương Thu Hường 2021603215

**Hà Nội, 2023**

# MỤC LỤC

[MỤC LỤC 3](#_Toc152771986)

[DANH MỤC HÌNH ẢNH 4](#_Toc152771987)

[LỜI NÓI ĐẦU 5](#_Toc152771988)

[CHƯƠNG 1: KHÔNG GIAN TRẠNG THÁI VÀ CÁC THUẬT TOÁN TÌM KIẾM HEURISTIC 6](#_Toc152771989)

[I**. Không gian trạng thái.** 6](#_Toc152771990)

[1. Mô tả trạng thái. 6](#_Toc152771991)

[2. Toán tử chuyển trạng thái. 6](#_Toc152771992)

[3. Không gian trạng thái của bài toán. 8](#_Toc152771993)

[I**I. Các thuật toán tìm kiếm Heuristic** 8](#_Toc152771994)

[1. Tổng quan về giải thuật tìm kiếm Heuristic. 8](#_Toc152771995)

[2. Tìm kiếm tối ưu (Best-First-Search) 10](#_Toc152771996)

[3. Thuật toán 𝐴𝑇 13](#_Toc152771997)

[4. Thuật toán 𝐴𝐾𝑇 13](#_Toc152771998)

[5. Thuật toán 𝐴\* 14](#_Toc152771999)

[CHƯƠNG 2: ỨNG DỤNG THUẬT TOÁN 18](#_Toc152772000)

[**I. Không gian trạng thái của bài toán trò chơi 8 số. 18**](#_Toc152772001)

[1. Giới thiệu bài toán. 18](#_Toc152772002)

[2. Không gian trạng thái của bài toán. 18](#_Toc152772003)

[3. Điều kiện của trạng thái đầu 20](#_Toc152772004)

[**II. Phân tích các thành phần để cài đặt thuật toán giải quyết bài toán 21**](#_Toc152772005)

[1. Giải thuật sử dụng 21](#_Toc152772006)

[2. Cài đặt chi tiết 22](#_Toc152772007)

[KẾT LUẬN 28](#_Toc152772008)

[TÀI LIỆU THAM KHẢO 29](#_Toc152772009)

# DANH MỤC HÌNH ẢNH

Hình 1 Ví dụ minh họa trong tìm kiếm tối ưu 12

Hình 2 Ví dụ cho trạng thái đích của bài toán 19

Hình 3 Trạng thái ban đầu 20

Hình 4 Trạng thái kết thúc 20

Hình 5 Ví dụ minh họa cho không gian trạng thái của bài toán 8 số 20

Hình 6 Ví dụ trạng thái đầu 21

Hình 7 Khởi tạo class Node 23

Hình 8 Hàm tìm vị trí 1 phần tử trong ma trận 24

Hình 9 Hàm tính chi phí 24

Hình 10 Hàm lấy các Node kề 25

Hình 11 Hàm tìm Node có chi phí nhỏ nhất 25

Hình 12 Hàm xây dựng đường đi 26

Hình 13 Hàm thực hiện kiểm tra điều kiện trạng thái ban đầu 27

Hình 14 Hàm a\_star 28

# LỜI NÓI ĐẦU

George Polya định nghĩa heuristic là “sự nghiên cứu về các phương pháp và các quy tắc trong việc khám phá và phát minh” (Polya 1945). Nghĩa này có thể xuất phát từ gốc Hy Lạp của động từ eurisco nghĩa là “tôi phát hiện”. Khi Archimedes nhảy ra khỏi bồn tắm và chộp lấy chiếc vương miện bằng vàng, ông ta đã la lên “Eureka” có nghĩa là “Tôi đã tìm thấy nó!”. Trong tìm kiếm không gian trạng thái, heuristic là các luật dùng để chọn những nhánh nào có khả năng dẫn đến một giải pháp chấp nhận được.

Các chương trình giải quyết những vấn đề trí tuệ nhân tạo sử dụng heuristic cơ bản theo hai dạng:

* Vấn đề có thể không có giải pháp chính xác vì những điều không rõ ràng trong diễn đạt vấn đề hoặc trong các dữ liệu có sẵn. Chẩn đoán y khoa là một ví dụ. Tập hợp các triệu chứng cho trước có thể do nhiều nguyên nhân gây ra, bác sĩ có thể dùng heuristic để chọn kết quả chẩn đoán nào thích hợp nhất và đưa ra kế hoạch điều trị.
* Vấn đề có thể có giải pháp chính xác, nhưng chi phí tính toán để tìm ra nó không cho phép. Trong nhiều vấn đề (như cờ vua chẳng hạn), không gian trạng thái phát triển rất nhanh và rất rộng vì số lượng các trạng thái có thể xảy ra tăng theo hàm mũ hoặc giai thừa cùng với độ sâu tìm kiếm. Trong những trường hợp này, các kỹ thuật tìm kiếm thô sơ như tìm kiếm sâu hay tìm kiếm rộng sẽ không tìm được giải pháp trong một giới hạn thời gian. Heuristic sẽ giảm bớt độ phức tạp bằng cách hướng việc tìm kiếm theo con đường có nhiều hứa hẹn nhất. Nhờ đã loại bỏ bớt các trạng thái không hứa hẹn và con cháu của chúng ra khỏi việc xem xét nên thuật toán heuristic có thể khắc phục việc bùng nổ trạng thái và tìm ra một giải pháp có thể chấp nhận được.

**CHƯƠNG 1: KHÔNG GIAN TRẠNG THÁI VÀ CÁC THUẬT TOÁN TÌM KIẾM HEURISTIC**

## Không gian trạng thái.

### Mô tả trạng thái.

Giải bài toán trong không gian trạng thái, trước hết phải xác định dạng mô tả trạng thái bài toán sao cho bài toán trở nên đơn giản hơn, phù hợp bản chất vật lý của bài toán (Có thể sử dụng các xâu ký hiệu, véctơ, mảng hai chiều, cây, danh sách,...).

Mỗi trạng thái chính là mỗi hình trạng của bài toán, các tình trạng ban đầu và tình trạng cuối của bài toán gọi là trạng thái đầu và trạng thái cuối.

Ví dụ: Bài toán đong nước: Cho 2 bình có dung tích lần lượt là m và n (lit). Với nguồn nước không hạn chế, dùng 2 bình trên để đong k lít nước. Không mất tính tổng quát có thể giả thiết k <= min(m,n).

* Tại mỗi thời điểm xác định, lượng nước hiện có trong mỗi bình phản ánh bản chất hình trạng của bài toán ở thời điểm đó.
* Gọi x là lượng nước hiện có trong bình dung tích m và y là lượng nước hiện có trong bình dung tích n.
* Như vậy bộ có thứ tự (x,y) có thể xem là trạng thái của bài toán. Với cách mô tả như vậy, các trạng thái đặc biệt của bài toán sẽ là:

+ Trạng thái đầu: (0,0)

+ Trạng thái cuối: (x,k) hoặc (k,y)

### **Toán tử chuyển trạng thái.**

Toán tử chuyển trạng thái thực chất là các phép biến đổi đưa từ trạng thái này sang trạng thái khác. Có hai cách dùng để biểu diễn các toán tử:

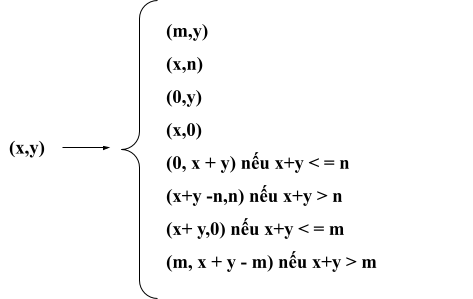
* + - Biểu diễn như một hàm xác định trên tập các trạng thái và nhận giá trị cũng trong tập này.
    - Biểu diễn dưới dạng các quy tắc sản xuất S? A có nghĩa là nếu có trạng thái S thì có thể đưa đến trạng thái A.

Ví dụ 1. Bài toán đong nước

Các thao tác sử dụng để chuyển trạng thái này sang trạng thái khác gồm:

* + - Đổ đầy một bình
    - Đổ hết nước trong một bình ra ngoài
    - Đổ nước từ bình này sang bình khác.

Như vậy, nếu trạng thái đang xét là (x,y) thì các trạng thái kế tiếp có thể chuyển đến sẽ là:



### **Không gian trạng thái của bài toán.**

* + - Không gian trạng thái là tập tất cả các trạng thái có thể có và tập các toán tử của bài toán.
    - Không gian trạng thái là một bộ bốn, Ký hiệu: K= (T, S, G, F).

Trong đó:

+ T: tập tất cả các trạng thái có thể có của bài toán.

+ S: trạng thái đầu.

+ G: tập các trạng thái đích.

+ F: tập các toán tử

Ví dụ 1. Không gian trạng thái của bài toán đong nước là bộ bốn T, S, G, F xác định như sau:

T = { (x,y) / 0 <= x <= m; 0 <= y <= n } S = (0,0)

G = { (x,k) hoặc (k,y) / 0 <= x <= m; 0 <= y <= n}

F = Tập các thao tác đong đầy, đổ ra hoặc đổ sang bình khác thực hiện trên một bình.

## Các thuật toán tìm kiếm Heuristic

### **Tổng quan về giải thuật tìm kiếm Heuristic.**

*a, Khái niệm*

Trong tìm kiếm không gian trạng thái, heuristic là các luật dùng để chọn những nhánh nào có nhiều khả năng nhất dẫn đến một giải pháp chấp nhận được

- Heuristic chỉ là một phỏng đoán chứa các thông tin về bước tiếp theo sẽ được chọn dùng trong việc giải quyết một vấn đề.

* + Heuristic là những tri thức được rút ra từ những kinh nghiệm, “trực giác” của con người
  + Heuristic có thể là những tri thức đúng hoặc sai

Vì các heuristic sử dụng những thông tin hạn chế nên chúng ít khi có khả năng đoán trước chính xác cách hành xử của không gian trạng thái ở những giai đoạn xa hơn.

1. *Chức năng của Heuristic.*

Các chương trình giải quyết những vấn đề trí tuệ nhân tạo sử dụng Heuristic cơ bản theo hai dạng:

* + Vấn đề có thể không có giải pháp chính xác vì những điều không rõ ràng trong diễn đạt vấn đề hoặc trong các dữ liệu có sẵn.
  + Vấn đề có thể có giải pháp chính xác, nhưng chi phí tính toán để tìm ra nó không cho phép.

1. *Ưu điểm của Heuristic.*

Thuật giải Heuristic thể hiện cách giải bài toán với các đặc tính sau:

* + Thường tìm được lời giải tốt (Nhưng không chắc là lời giải tốt nhất).
  + Giải bài toán theo thuật giải Heuristic thường dễ dàng và nhanh chóng đưa ra kết quả hơn so với giải thuật tối ưu, vì vậy chi phí thấp hơn.
  + Thuật giải Heuristic thường thể hiện khá tự nhiên, gần gũi với cách suy nghĩ và hành động con người.

1. *Phương pháp xây dựng thuật giải Heuristic.*

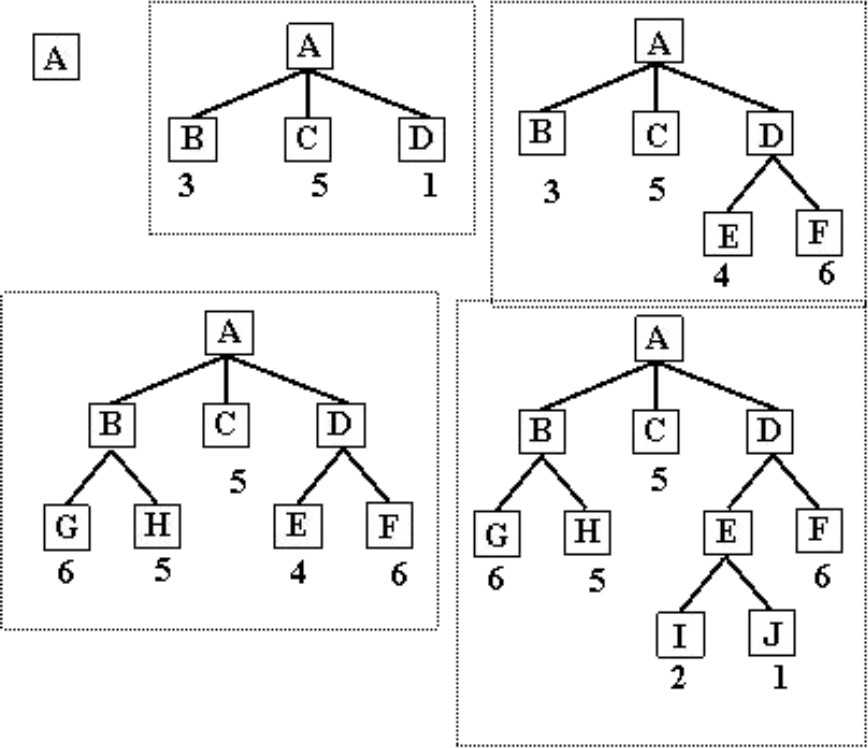
Thuật giải Heuristic gồm hai phần: Hàm đánh giá Heuristic và thuật toán để sử dụng nó trong tìm kiếm không gian trạng thái.

Có nhiều các để xây dựng một thuật giải Heuristic, trong đó người ta thường dựa và một số nguyên lý cơ bản như sau:

* + *Nguyên lý vét cạn thông minh*: Trong một bài toán tìm kiếm nào đó, khi không gian tìm kiếm lớn, ta thường tìm cách giới hạn lại không gian tìm kiếm hoặc thực hiện một kiểu dò tìm đặc biệt dựa vào đặc thù của bài toán để nhanh chóng tìm ra mục tiêu
  + *Nguyên lý tham lam (Greedy)*: lấy tiêu chuẩn tối ưu (Trên phạm vi toàn cục) của bài toán để làm tiêu chuẩn chọn lựa hành động cho phạm vi cục bộ của từng bước (Hay từng giai đoạn) trong quá trình tìm kiếm lời giải
  + *Nguyên lý thứ tự*: thực hiện hành động dựa trên một cấu trúc thứ tự hợp lý của không gian khảo sát nhằm nhanh chóng đạt được một lời giải tốt.
  1. **Tìm kiếm tối ưu (Best-First-Search)**

Ưu điểm của tìm kiếm theo chiều sâu là không phải quan tâm đến sự mở rộng của tất cả các nhánh. Ưu điểm của tìm kiếm chiều rộng là không bị sa vào các đường dẫn bế tắc (các nhánh cụt). Tìm kiếm tối ưu (Best-First Search-BeFS) sẽ kết hợp hai phương pháp trên cho phép ta đi theo một con đường duy nhất tại một thời điểm, nhưng đồng thời vẫn xét được những hướng khác. Nếu con đường đang đi không triển vọng bằng những con đường đang quan sát, ta sẽ chuyển sang đi theo một trong số các con đường này.

Một cách cụ thể, tại mỗi bước của tìm kiếm BeFS, ta chọn đi theo trạng thái có khả năng cao nhất trong số các trạng thái đã được xét cho đến thời điểm đó. BeFS khác với tìm kiếm leo đồi là chỉ chọn trạng thái có khả năng cao nhất trong số các trạng thái kế tiếp có thể đến được t trạng thái hiện tại. Như vậy, với tiếp cận này, ta sẽ ưu tiên đi vào những nhánh tìm kiếm có khả năng nhất (giống tìm kiếm leo đồi), nhưng ta sẽ không bị lẩn quẩn trong các nhánh này vì nếu càng đi sâu vào một hướng mà ta phát hiện ra rằng hướng này càng đi thì càng xấu, đến mức nó xấu hơn cả những hướng mà ta chưa đi, thì ta sẽ không đi tiếp hướng hiện tại nữa mà chọn đi theo một hướng tốt nhất trong số những hướng chưa đi. Đó là tư tưởng chủ đạo của tìm kiếm tối ưu.

Ví dụ minh họa:

*Hình 1 Ví dụ minh họa trong tìm kiếm tối ưu*

Khởi đầu, chỉ có một nút (trạng thái) A nên nó sẽ được mở rộng tạo ra 3 nút mới B,C và D. Các con số dưới nút là giá trị cho biết độ tốt của nút. Con số càng nhỏ, nút càng tốt. Do D là nút có khả năng nhất nên nó sẽ được mở rộng tiếp sau nút A và sinh ra 2 nút kế tiếp là E và F. Đến đây, ta lại thấy nút B có vẻ có khả năng nhất (trong các nút B,C,E,F) nên ta sẽ chọn mở rộng nút B và tạo ra 2 nút G và H. Nhưng lại một lần nữa, hai nút G, H này được đánh giá ít khả năng hơn E, vì thế sự chú ý lại trở về E. E được mở rộng và các nút được sinh ra từ E là I và J. Ở bước kế tiếp, J sẽ được mở rộng vì nó có khả năng nhất. Quá trình này tiếp tục cho đến khi tìm thấy một lời giải.

Để cài đặt các thuật giải theo kiểu tìm kiếm BFS, thường cần dùng 2 tập hợp:

* OPEN : tập chứa các trạng thái đã được sinh ra nhưng chưa được xét đến (vì ta đã chọn một trạng thái khác). Thực ra, OPEN là một loại hàng đợi ưu tiên (priority queue) mà trong đó, phần tử có độ ưu tiên cao nhất là phần tử tốt nhất. Người ta thường cài đặt hàng đợi ưu tiên bằng Heap.
* CLOSE : tập chứa các trạng thái đã được xét đến. Chúng ta cần lưu trữ những trạng thái này trong bộ nhớ để đề phòng trường hợp khi một trạng thái mới được tạo ra lại trùng với một trạng thái mà ta đã xét đến trước đó. Trong trường hợp không gian tìm kiếm có dạng cây thì không cần dùng tập này.

**Thuật giải**

* Đặt OPEN chứa trạng thái khởi đầu.
* Cho đến khi tìm được trạng thái đích hoặc không còn nút nào trong OPEN, thực hiện :

+ Chọn trạng thái tốt nhất (Tmax) trong OPEN (và xóa Tmax khỏi OPEN)

+ Nếu Tmax là trạng thái kết thúc thì thoát.

+ Ngược lại, tạo ra các trạng thái kế tiếp Tk có thể có từ trạng thái Tmax. Đối với mỗi trạng thái kế tiếp Tk thực hiện:

Tính f(Tk);

Thêm Tk vào OPEN

BFS khá đơn giản. Tuy vậy, trên thực tế, cũng như tìm kiếm chiều sâu và chiều rộng, hiếm khi ta dùng BFS một cách trực tiếp. Thông thường, người ta thường dùng các phiên bản của BFS là 𝐴𝑇, 𝐴𝐾𝑇 và 𝐴\*.

**Thông tin về quá khứ và tương lai:**

Thông thường, trong các phương án tìm kiếm theo kiểu BeFS, chi phí f của một trạng thái được tính dựa theo hai giá trị mà ta gọi là là g và h. Trong đó h, như đã biết, đó là một ước lượng về chi phí từ trạng thái hiện hành cho đến trạng thái đích (thông tin tương lai), còn g là chiều dài quãng đường đã đi từ trạng thái ban đầu cho đến trạng thái hiện tại (thông tin quá khứ). Khi đó hàm ước lượng tổng chi phí f(n) được tính theo công thức:

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

* 1. **Thuật toán** 𝐴𝑇

Thuật giải 𝐴𝑇 là một phương pháp tìm kiếm theo kiểu BeFS với chi phí của đỉnh là giá trị hàm g (tổng chiều dài thực sự của đường đi từ đỉnh bắt đầu đến đỉnh hiện tại).

**Giải thuật**

1. Đặt OPEN chứa trạng thái khởi đầu.
2. Cho đến khi tìm được trạng thái đích hoặc không còn nút nào trong OPEN, thực hiện:
   1. Chọn trạng thái (Tmax) có giá trị g nhỏ nhất trong OPEN (và xóa Tmax khỏi OPEN)
   2. Nếu Tmax là trạng thái kết thúc thì thoát.
   3. Ngược lại, tạo ra các trạng thái kế tiếp Tk có thể có từ trạng thái Tmax. Đối với mỗi trạng thái kế tiếp Tk thực hiện:

g(Tk) = g(Tmax) + cost(Tmax, Tk) Thêm Tk vào OPEN.

**\*Note:** Vì chỉ sử dụng hàm g (mà không dùng hàm ước lượng h’ để đánh giá độ tốt của một trạng thái nên ta cũng có thể xem AT chỉ là một thuật toán.

* 1. **Thuật toán** 𝐴𝐾𝑇

Thuật giải 𝐴𝐾𝑇 trong quá trình tìm đường đi chỉ xét đến các đỉnh và giá của chúng. Nghĩa là việc tìm đỉnh triển vọng chỉ phụ thuộc hàm g(n) (thông tin

quá khứ). Tuy nhiên thuật giải này không còn phù hợp khi gặp phải những bài toán phức tạp (độ phức tạp cấp hàm mũ) do ta phải tháo một lượng nút lớn. Để khắc phục nhược điểm này, người ta sử dụng thêm các thông tin bổ sung xuất phát từ bản thân bài toán để tìm ra các đỉnh có triển vọng, tức là đường đi tối ưu sẽ tập trung xung quanh đường đi tốt nhất nếu s dụng các thông tin đặc tả về bài toán (thông tin quá tương lai).

Theo thuật giải này, chi phí của đỉnh được xác định: f(n) = g(n) + h(n) Đỉnh n được chọn nếu f(n) min. Việc xác định hàm ước lượng h(n) được thực hiện dựa theo:

- Chọn toán tử xây dựng cung sao cho có thể loại bớt các đỉnh không liên quan và tìm ra các đỉnh có triển vọng.

- Sử dụng thêm các thông tin bổ sung nhằm xây dựng tập OPEN và cách lấy các đỉnh trong tập OPEN.

Để làm được việc này người ta phải đưa ra độ đo, tiêu chun để tìm ra các đỉnh có triển vọng. ác hàm s dụng các kỹ thuật này gọi là hàm đánh giá. Sau đây là một số phương pháp xây dựng hàm đánh giá:

* + - Dựa vào xác suất của đỉnh trên đường đi tối ưu.
    - Dựa vào khoảng cách, sự sai khác của trạng thái đang xét với trạng thái đích hoặc các thông tin liên quan đến trạng thái đích.

**Giải thuật**

1. Đặt OPEN chứa trạng thái khởi đầu.
2. Cho đến khi tìm được trạng thái đích hoặc không còn nút nào trong OPEN, thực hiện :
   1. Chọn trạng thái (Tmax) có giá trị f nhỏ nhất trong OPEN (và xóa Tmax khỏi OPEN)
   2. Nếu Tmax là trạng thái kết thúc thì thoát.
   3. Ngược lại, tạo ra các trạng thái kế tiếp Tk có thể có từ trạng thái Tmax. Đối với mỗi trạng thái kế tiếp Tk thực hiện :

g(Tk) = g(Tmax) + cost(Tmax, Tk) Tính h’(Tk)

f(Tk) = g(Tk) + h’(Tk) Thêm Tk vào OPEN

* 1. **Thuật toán 𝐴\***

A\* là giải thuật tìm kiếm trong đồ thị, tìm đường đi từ một đỉnh hiện tại đến đỉnh đích có sử dụng hàm để ước lượng khoảng cách hay còn gọi là hàm Heuristic.

Từ trạng thái hiện tại A\* xây dựng tất cả các đường đi có thể đi dùng

hàm ước lượng khoảng cách (hàm Heuristic) để đánh giá đường đi tốt nhất có thể đi. Tùy theo mỗi dạng bài khác nhau mà hàm Heuristic sẽ được đánh giá khác nhau. A\* luôn tìm được đường đi ngắn nhất nếu tồn tại đường đi như thế.

A\* lưu giữ một tập các đường đi qua đồ thị, từ đỉnh bắt đầu đến đỉnh kết thúc, tập các đỉnh có thể đi tiếp được lưu trong tập OPEN.

Thứ tự ưu tiên cho một đường đi được quyết định bởi hàm Heuristic

được đánh giá:

f(x) = g(x) + h(x)

Trong đó:

- g(x) là chi chi phí của đường đi từ điểm xuất phát cho đến thời điểm hiện tại.

- h(x) là hàm ước lượng chi phí từ đỉnh hiện tại đến đỉnh đích f(x) thường có giá trị càng thấp thì độ ưu tiên càng cao.

- Open: tập các trạng thái đã được sinh ra nhưng chưa được xét đến

**Giải thuật:**

- Close: tập các trạng thái đã được xét đến.

- Cost(p, q): là khoảng cách giữa p, q.

- g(p): khoảng cách từ trạng thái đầu đến trạng thái hiện tại p.

- h(p): giá trị được lượng giá từ trạng thái hiện tại đến trạng thái đích.

- f(p) = g(p) + h(p)

Các bước thực hiện

- Bước 1:

Open: = {s}

Close: = {}

- Bước 2: while (Open !={})

+ Chọn trạng thái (đỉnh) tốt nhất p trong Open (xóa p khỏi Open).

+ Nếu p là trạng thái kết thúc thì thoát.

+ Chuyển p qua Close và tạo ra các trạng thái kế tiếp q sau p.

-Nếu q đã có trong Open

Nếu g(q) > g(p) + Cost(p, q) thì

g(q) = g(p) + Cost(p, q)

f(q) = g(q) + h(q)

prev(q) = p (đỉnh cha của q là p)

-Nếu q chưa có trong Open

g(q) = g(p) + cost(p, q)

f(q) = g(q) + h(q)

prev(q) = p

Thêm q vào Open

-Nếu q có trong Close

Nếu g(q) > g(p) + Cost(p, q)

Bỏ q khỏi Close

Thêm q vào Open

* Bước 3: Không tìm được

Cũng như tìm kiếm theo chiều rộng (breadth-first search), A\* là thuật toán đầy đủ (complete) theo nghĩa rằng nó sẽ luôn luôn tìm thấy một lời giải nếu bài toán có lời giải.

Nếu hàm heuristic h có tính chất thu nạp được (admissible), nghĩa là nó không bao giờ đánh giá cao hơn chi phí nhỏ nhất thực sự của việc đi tới đích, thì bản thân A\* có tính chất thu nạp được (hay tối ưu) nếu sử dụng một tập đóng. Nếu không sử dụng tập đóng thì hàm h phải có tính chất đơn điệu (hay nhất quán) thì A\* mới có tính chất tối ưu. Nghĩa là nó không bao giờ đánh giá chi phí đi từ một nút tới một nút kề nó cao hơn chi phí thực. Phát biểu một cách hình thức, với mọi nút x,y trong đó y là nút tiếp theo của x:

A\* còn có tính chất hiệu quả một cách tối ưu (optimally efficient) với mọi hàm heuristic h, có nghĩa là không có thuật toán nào cũng sử dụng hàm heuristic đó mà chỉ phải mở rộng ít nút hơn A\*, trừ khi có một số lời giải chưa đầy đủ mà tại đó h dự đoán chính xác chi phí của đường đi tối ưu.

Quan hệ với tìm kiếm chi phí đều (uniform-cost search)

Thuật toán Dijkstra là một trường hợp đặc biệt của A\* trong đó đánh giá heuristic là một hàm hằng h(x) = 0 với mọi x.

**Mức độ phức tạp**

Vấn đề sử dụng bộ nhớ của A\* còn rắc rối hơn độ phức tạp thời gian. Trong trường hợp xấu nhất, A\* phải ghi nhớ số lượng nút tăng theo hàm mũ. Một số biến thể của A\* đã được phát triển để đối phó với hiện tượng này, một trong số đó là A\* lặp sâu dần (iterative deepening A\*), A\* bộ nhớ giới hạn (memory-bounded A\* - MA\*) và A\* bộ nhớ giới hạn đơn giản (simplified memory bounded A\*). Một thuật toán tìm kiếm có thông tin khác cũng có tính chất tối ưu và đầy đủ nếu đánh giá heuristic của nó là thu nạp được (admissible). Đó là tìm kiếm đệ quy theo lựa chọn tốt nhất (recursive best-first search - RBFS).

**Ưu điểm**

Một thuật giải linh động, tổng quát, trong đó hàm chứa cả tìm kiếm chiều sâu, tìm kiếm chiều rộng và những nguyên lý Heuristic khác. Nhanh chóng tìm đến lời giải với sự định hướng của hàm Heuristic. Chính vì thế mà người ta thường nói A\* chính là thuật giải tiêu biểu cho Heuristic.

**Nhược điểm**

A\* rất linh động nhưng vẫn gặp một khuyết điểm cơ bản - giống như chiến lược tìm kiếm chiều rộng - đó là tốn khá nhiều bộ nhớ để lưu lại những trạng thái đã đi qua.

# CHƯƠNG 2: ỨNG DỤNG THUẬT TOÁN

**HEURISTIC VÀO BÀI TOÁN TRÒ CHƠI 8 SỐ**

## Không gian trạng thái của bài toán trò chơi 8 số.

### **Giới thiệu bài toán.**

Bài toán 8-puzzle (hay còn gọi là 8 số) là một bài toán quen thuộc với những người bắt đầu tiếp cận với môn Trí tuệ nhân tạo. Bài toán có nhiều phiên bản khác nhau dựa theo số ô, như 8-puzzle, 15-puzzle,... ở mức độ đơn giản nhất, chúng em xem xét dạng bài toán 8-puzzle.

Bài toàn gồm một bảng ô vuông kích thước 3x3, có tám ô được đánh số từ 1 tới 8 và một ô trống. Trạng thái ban đầu, các ô được sắp xếp một cách ngẫu nhiên, nhiệm vụ của người chơi là tìm cách đưa chúng về đúng thứ tự theo như yêu cầu. Ví dụ đưa về giống như hình dưới:

A screenshot of a number game

Description automatically generated

*Hình 2: Ví dụ cho trạng thái đích của bài toán*

- Trong quá trình giải bài toàn, tại mỗi bước, ta giả định chỉ có ô trống là di chuyển, như vậy, tối đa ô trống có thể có 4 khả năng di chuyển (lên trên, xuống dưới, sang trái, sang phải).

### **2. Không gian trạng thái của bài toán.**

Trong bảng ô vuông 3 hàng, 3 cột , mỗi ô chứa một số nằm trong phạm vi từ 1 đến 8 sao cho không có 2 ô có cùng giá trị, có một ô trong bảng bị trống (không chứa giá trị nào cả. Xuất phát từ một sắp xếp nào đó các số trong bảng, hãy dịch chuyển ô trống sang phải, sang trái, lên trên hoặc xuống dưới (nếu có thể được) để đưa về bảng ban đầu về bảng quy ước trước. Chẳng hạn hình 3 dưới đây là bảng xuất phát và hình 4 là bảng mà ta phải thực hiện các bước di chuyển ô trống để đạt được.

A screenshot of a game

Description automatically generated

*Hình 3: Trạng thái ban đầu*

A screenshot of a game

Description automatically generated

*Hình 4: Trạng thái kết thúc*

Ta có không gian trạng thái của bài toán được miêu tả như sau:

A diagram of a diagram

Description automatically generated

*Hình 5:Ví dụ minh họa cho không gian trạng thái của bài toán 8 số*

Như vậy, chúng ta nhận thấy rằng, nếu không sử dụng giải thuật một cách hợp lý, tập không gian tìm kiếm sẽ vô cùng lớn. Điều này khiến cho việc đưa khối tiến đến trạng thái đích là vô cùng khó khăn

### **3. Điều kiện của trạng thái đầu**

Có những trạng thái của bảng số không thể chuyển về trạng thái đích. Người ta chứng minh được rằng, để có thể chuyển từ trạng thái đầu tới trạng thái đích, thì trạng thái đầu này phải thỏa mãn điều kiện được xác định như sau:

Ta xét lần lượt từ trên xuống dười, từ trái sang phải, với mỗi ô số đang xét (giả sử là ô thứ i), ta kiểm tra xem phía sau có bao nhiêu ô số có giá trị nhỏ hơn ô đó. Sau đó ta tính tổng N = n1 + n2 + … + n8.

Ta có quy tắc chung sau cho bài toán n-puzzle:

** Nếu số ô vuông lẻ:**

N mod 2 = 0 (1)

** Nếu số ô vuông chẵn:**

N mod 2 = 0 và ô trống phải nằm ở hàng chẵn xét từ trên xuống. (2)

N mod 2 = 1 và ô trống phải nằm ở hàng lẻ xét từ trên xuống. (3)

Cụ thể, ta đang xét bài toán 8-puzzle – có 9 ô vuông nên trạng thái đầu phải thỏamãn điều kiện (1).

**Ví dụ**: Cho trạng thái đầu sau 2-0-6-8-7-5-4-3-1

A screenshot of a cell phone

Description automatically generated

*Hình 6: Ví dụ trạng thái đầu*

Xét ô thứ nhất có giá trị 2: Phía sau có 1 ô nhỏ hơn (1) => n1 = 1

Xét ô thứ hai có giá trị 6: Phía sau có 4 ô nhỏ hơn (5,4,3,1) => n2 = 4

Xét ô thứ ba có giá trị 8: Phía sau có 5 ô nhỏ hơn (7,5,4,3,1) => n3 = 5

Xét ô thứ tư có giá trị 7: Phía sau có 4 ô nhỏ hơn (5,4,3,1) => n4 = 4

Xét ô thứ năm có giá trị 5: Phía sau có 3 ô nhỏ hơn (4,3,1) => n5 = 3

Xét ô thứ sáu có giá trị 4: Phía sau có 2 ô nhỏ hơn (3,1) => n6 = 2

Xét ô thứ bảy có giá trị 3: Phía sau có 1 ô nhỏ hơn (1) => n7 = 1

Xét ô thứ tám có giá trị 1: Phía sau không còn ô nào nhỏ hơn => n8 = 0

N = 1 + 4 + 5 + 4 + 3 + 2 + 1 + 0 = 20

Ta có 20 mod 2 = 0 => Thỏa mãn.

Những trạng thái của bảng số mà có thể chuyển về trạng thái đích gọi là cấu hình hợp lệ, ngược lại gọi là cấu hình không hợp lệ.

Với bảng số kích thước mxm (m là cạnh) thì ta có không gian trạng thái là (mxm)!

Với bài toán 8-puzzle, các trạng thái có thể có của bảng số là (3x3)! = 362880. Nếu m tăng lên 1, sẽ làm không gian trạng thái tăng lên rất lớn, do vậy các phiên bản bài toán với m>3 ít khi được áp dụng.

## II. Phân tích các thành phần để cài đặt thuật toán giải quyết bài toán

### **1. Giải thuật sử dụng**

A\* là giải thuật tìm kiếm trong đồ thị, tìm đường đi từ một đỉnh hiện tại đến đỉnh đích có sử dụng hàm để ước lượng khoảng cách hay còn gọi là hàm Heuristic.

Từ trạng thái hiện tại A\* xây dựng tất cả các đường đi có thể đi dùng hàm ước lượng khoảng cách (hàm Heuristic) để đánh giá đường đi tốt nhất có thể đi. Tùy theo mỗi dạng bài khác nhau mà hàm Heuristic sẽ được đánh giá khác nhau. A\* luôn tìm được đường đi ngắn nhất nếu tồn tại đường đi như thế. A\* lưu giữ một tập các đường đi qua đồ thị, từ đỉnh bắt đầu đến đỉnh kết thúc, tập các đỉnh có thể đi tiếp được lưu trong tập Open.

Thứ tự ưu tiên cho một đường đi được quyết định bởi hàm Heuristic

được đánh giá f(x) = g(x) + h(x)

- g(x) là chi chi phí của đường đi từ điểm xuất phát cho đến thời điểm hiệntại.

- h(x) là hàm ước lượng chi phí từ đỉnh hiện tại đến đỉnh đích f(x) thường có giá trị càng thấp thì độ ưu tiên càng cao

### **2. Cài đặt chi tiết**

*a. Khởi tạo lớp Node*

A computer screen shot of a code

Description automatically generated

*Hình 7: Khởi tạo class Node*

Trong đó:

- current\_node: Trạng thái hiện tại của trò chơi

- before\_node: Trạng thái trước đó của trò chơi

- g: Chi phí đã di chuyển từ trạng thái ban đầu đến trạng thái hiện tại.

- h: Giá trị dự đoán chi phí từ trạng thái hiện tại đến trạng thái kết thúc.

- dir: Hướng di chuyển để đạt được trạng thái hiện tại từ trạng thái trước

đó.

- Phương thức f(): Tính tổng chi phí tạm thời (g + h).

*b. Khởi tạo hàm tìm vị trí 1 phần tử trong ma trận hiện tại*

A computer screen shot of a program code

Description automatically generated

*Hình 8: Hàm tìm vị trí 1 phần tử trong ma trận*

Hàm này có nhiệm vụ:

- Tìm vị trí của một phần tử trong trạng thái hiện tại (ma trận).

- Sử dụng vòng lặp để duyệt qua từng hàng và cột, trả về tọa độ nếu phần tử được tìm thấy.

*c. Khởi tạo hàm tính chi phí từ các ô tới vị trí đích*

*A computer screen shot of a code

Description automatically generated*

*Hình 9: Hàm tính chi phí*

Hàm này sẽ:

- Tính giá trị h dựa trên khoảng cách Manhattan giữa các ô số và vị trí đích.

- Duyệt qua từng hàng và cột trong trạng thái hiện tại, tính khoảng cách hàng và cột tới vị trí đích của số hiện tại, sau đó cộng giá trị tới cost.

*d, Hàm lấy các Nodes kề*

A computer code on a black background

Description automatically generated

*Hình 10: Hàm lấy các Nodes kề*

Hàm này sẽ:

- Tạo ra danh sách các nút kề từ một nút hiện tại.

- Xác định vị trí của ô trống (0) trong trạng thái hiện tại.

- Duyệt qua các hướng (U, D, L, R) và tính vị trí mới của ô trống sau mỗi bước di chuyển.

- Nếu vị trí mới nằm trong giới hạn của ma trận, tạo một trạng thái mới bằng cách hoán đổi ô trống và ô kế bên, và thêm nút mới này vào danh sách listNode.

*e. Xây dựng hàm tìm Node có chi phí nhỏ nhất*

A computer screen shot of a code

Description automatically generated

*Hình 11: Hàm tìm Node có chi phí nhỏ nhất*

Tìm nút có giá trị f nhỏ nhất trong tập openSet.

Lặp qua các nút trong openSet, so sánh giá trị f và lưu lại nút có giá trị f thấp nhất.

*f. Hàm xây dựng đường đi tốt nhất*

A computer screen shot of a code

Description automatically generated

*Hình 12: Hàm xây dựng đường đi*

- Xây dựng đường đi từ trạng thái kết thúc trở về trạng thái ban đầu, sử dụng thông tin về các nút đã thăm trong quá trình tìm kiếm.

- Bắt đầu từ trạng thái kết thúc, lặp qua các nút trước đó thông qua trường before\_node để tạo ra một chuỗi các trạng thái và hướng di chuyển.

*g. Hàm thực hiện kiểm tra điều kiện trạng thái ban đầu*

A computer screen shot of a program code

Description automatically generated

*Hình 13: Hàm thực hiện kiểm tra điều kiện trạng thái ban đầu*

-Hàm get\_reverse\_count thực hiện đếm số lượng cặp đảo chiều.

-Hàm is\_backtracking\_goal thực hiện kiểm tra điều kiện trạng thái ban đầu nếu số lượng cặp đảo chiều chia hết cho 2 thì thỏa mãn thực hiện tìm đường đi.

*h. Xây dựng hàm a\_star thực hiện thuật toán*

*A computer screen shot of a program code

Description automatically generated*

*Hình 14: Hàm a\_star*

Đây là hàm chính thực hiện thuật toán tìm kiếm A\* để giải bài toán trò chơi trượt ô số.

- Khởi tạo tập open\_set với nút ban đầu, và tập close\_set rỗng.

- Trong vòng lặp, lấy nút tốt nhất từ open\_set, thêm nút này vào close\_set.

- Nếu trạng thái hiện tại của nút tốt nhất là trạng thái kết thúc, xây dựng đường đi và trả về.

- Nếu không, tạo các nút kề từ nút tốt nhất và thêm chúng vào open\_set.

# KẾT LUẬN

Như vậy, mỗi một phương pháp đều có ưu và nhược điểm riêng, tuy nhiên giải thuật A\* áp dụng cho bài toán trò chơi 8 số là hiệu quả hơn cả. Giải thuật A\* là thuật toán tìm kiếm trong đồ thị có thông tin phản hồi, sử dụng đánh giá heuristic để xếp loại từng nút và duyệt nút theo hàm đánh giá này. Với các dạng bài toán tìm kiếm trên không gian trạng thái, có hai trường hợp cần tới heuristic:

* Những vấn đề không thể có nghiệm chính xác do các mệnh đề không phát biểu chặt chẽ hay do thiếu dữ liệu để khẳng định kết quả.
* Những vấn đề có nghiệm chính xác nhưng chi phí tính toán để tìm ra nghiệm là quá lớn (dẫn đến bùng nổ tổ hợp).

=> Heuristic giúp ta tìm kiếm đạt kết quả với chi phí thấp hơn.

# TÀI LIỆU THAM KHẢO

[https://websitehcm.com/cac-thuat-toan-informed-search-algorithms](https://websitehcm.com/cac-thuat-toan-informed-search-algorithms/)/ <https://www.iostream.vn/article/thuat-giai-a-DVnHj> <https://voer.edu.vn/m/thuat-toan-tim-kiem-heuristic/90488a73> <https://cuuduongthancong.com/atc/1153/cac-phuong-phap-tim-kiem-heuristic>