ĐẠI HỌC KINH TẾ TP. HỒ CHÍ MINH

UEH UNIVERSITY

College of Technology and Design

SCHOOL OF BUSINESS INFORMATION TECHNOLOGY

BÁO CÁO

Nhóm sinh viên: Đặng Thị Thu Hiền, Đỗ Thanh Hoa, Bùi Tiến Hiếu, Nguyễn Trương Hoàng, Nguyễn Huy Hoàng

Môn học: Lập trình phân tích dữ liệu

Giảng viên hướng dẫn: Nguyễn An Tế

Đề tài: Nghiên cứu thuật toán A* cho bài toán hút bụi

Hồ Chí Minh, ngày ... tháng ... năm...

THÀNH VIÊN NHÓM

STT	Họ và tên	MSSV
1	Đặng Thị Thu Hiền	31221025556
2	Bùi Tiến Hiếu	31221026291
3	Đỗ Thanh Hoa	31211025193
4	Nguyễn Huy Hoàng	31221023992
5	Nguyễn Trương Hoàng	31221026058

LỜI NÓI ĐẦU

Nhóm chúng con xin được gửi lời cảm ơn tới thầy Nguyễn An Tế vì những gì thầy đã truyền tải lại cho chúng con ở trên lớp. Những bài giảng của thầy có thể nói rằng rất hay, rất đáng nhớ, rất đáng quý và cực kỳ đáng trân trọng. Những tri thức được truyền lại đó quả thật là những "chất vàng mười" của tri thức mà thầy đã rèn nên thông qua vốn kiến thức uyên bác của thầy. Nhờ có "chất vàng mười" đó mà vốn hiểu biết của chúng con được mở rộng và tư duy được sắc bén hơn theo cách tư duy logic hơn, có hệ thống hơn không chỉ riêng đối với ngành học mà còn ở những lĩnh vực khác. Đó là nền tảng kiên cố làm tiền đề để nhóm có thể tự tìm hiểu cho những kiến thức liên quan đến lĩnh vực của đề tài về sau này. Những giá trị mà thầy đã mang đến sẽ còn mãi với thời gian, bởi đó là những gì thật sự quý và đáng trân trọng, thứ mang trong mình sức mạnh vượt qua được "sự băng hoại của thời gian".

Một lần nữa, nhóm 03 xin chân thành cảm ơn thầy Nguyễn An Tế vì tất cả những gì thầy đã mang đến cho chúng con. Nhóm chúng con chúc thầy luôn được khỏe mạnh và dồi dào sức khỏe. Mong rằng không chỉ thầy và những người xung quanh thầy sẽ luôn được bình an và hạnh phúc.

TP. HCM, Ngày 21 Tháng 11 Năm 2024,

Nhóm 03

MỤC LỤC

DANH MỤC BẢNG BIỂU

DANH MỤC HÌNH ẢNH

CHƯƠNG I: TỔNG QUAN ĐỀ TÀI

1.1. Tổng quan về bài toán tìm đường đi ngắn nhất

Bài toán tìm đường đi ngắn nhất (Shortest Path Problem) là một trong những vấn đề cơ bản và quan trọng trong lý thuyết đồ thị. Bài toán tìm đường đi ngắn nhất yêu cầu xác định đường đi có tổng trọng số nhỏ nhất giữa hai đỉnh (nodes) trong một đồ thị có trọng số. Đồ thị có thể là có hướng hoặc không có hướng, và trọng số của các cạnh (edges) có thể là dương hoặc âm.

Bài toán này có thể chia làm 2 loại. Bài toán đường đi ngắn nhất nguồn đơn là bài toán tìm một đường đi giữa hai đỉnh trong đồ thị G(V,E) có trọng số cạnh và hai đỉnh u, v thuộc V sao cho tổng các trọng số của các cạnh tạo nên đường đi đó là nhỏ nhất. Bài toán tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh là một bài toán tương tự, trong đó ta phải tìm các đường đi ngắn nhất đi từ đỉnh u đến đỉnh v, với mọi cặp đỉnh u, v thuộc V [20].

Một số thuật toán phổ biến được sử dụng để xử lý bài toán này có thể kể đến như thuật toán Dijkstra, thuật toán Bellman-Ford, thuật toán Floyd-Warshall, thuật toán A*,... Bài toán tìm đường đi ngắn nhất có nhiều ứng dụng thực tiễn, từ việc tối ưu hóa mạng lưới giao thông, tìm kiếm đường đi trong bản đồ, đến các vấn đề trong viễn thông và mạng máy tính,...

1.2. Mục tiêu nghiên cứu

Đề tài sử dụng thuật toán A* để tìm đường đi có tổng chi phí bé nhất trong bài toán hút bụi của một con robot. Trong phạm vi này, nhóm sẽ nghiên cứu cách robot đi đến các ô dơ cần được dọn cũng như đường đi của robot với các hàm heuristic khác nhau. Để từ đó đưa ra một số nhận dịnh và tìm ra hướng phát triển cho đề tài của nhóm.

1.3. Phương pháp nghiên cứu

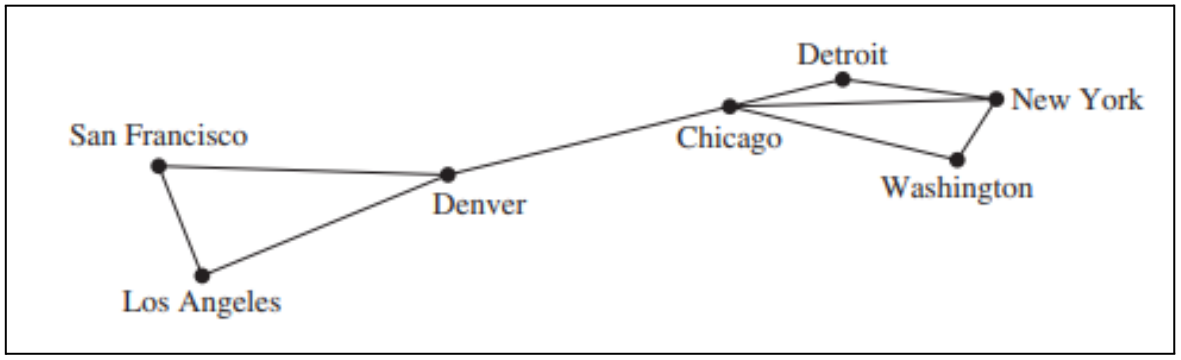
Thuật toán A* được áp dụng để lập trình cho robot hút bụi, nhằm tối ưu hóa quá trình làm sạch các ô bẩn trong ma trận. Người sử dụng có thể chỉ định kích thước ma trận và số lượng ô bẩn với vị trí ngẫu nhiên, từ đó robot sẽ xác định lộ trình di chuyển hiệu quả nhất. Kết quả thu được từ thuật toán không chỉ thể hiện lộ trình di chuyển của robot mà còn tính toán tổng chi phí cho quá trình làm sạch.

CHƯƠNG II: CƠ SỞ LÝ THUYẾT

2.1. Đồ thị

2.1.1. Sơ lược về đồ thị (Khái niệm đồ thị)

Đồ thị, hay diễn giải theo một cách khác, đó là một cách để biểu diễn mối quan hệ giữa các đối tượng. Các đối tượng đó được thể hiện bằng các *đỉnh* trong đồ thị, và mối quan hệ giữa chúng được biểu diễn bằng cấu trúc được biết với cái tên *cạnh*. Ta đã bắt gặp ứng dụng của đồ thị trong nhiều lĩnh vực, đó là vận chuyển, đó là mạng máy tính, đó là kỹ thuật điện, hoặc đó cũng có thể là địa lý,... [1]. Một ví dụ về đồ thị được thể hiện qua hình dưới, trong đó đỉnh là các thành phố, đó là {San Francisco, Los Angeles, Denver, Chicago, Detroit, New York, Washington}. Các cạnh giữa các đỉnh này thể hiện đường đi giữa các thành phố.

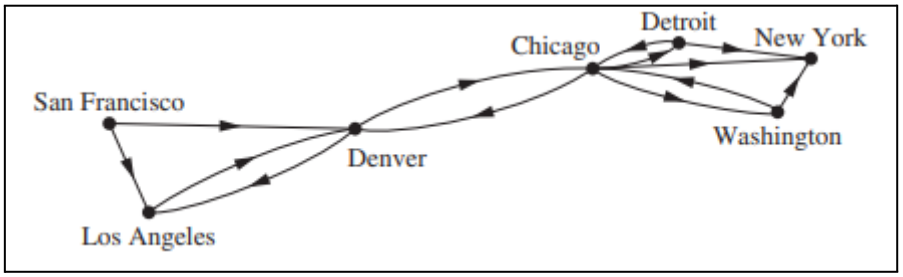


Hình 1: Ví dụ về đồ thị - Nguồn: [2]

Một cách tổng quát, gọi V = $\{v_1, v_2, ..., v_n\}$ là một tập hữu hạn có n phần tử khác rỗng và gọi $E \subset \{\{a, b\}: a, b \in V\}$. Một đồ thị G với các đỉnh là phần tử của V và cạnh là phần tử của E được ký hiệu là $G = (V, E)$. Nếu không có sự nhầm lẫn về cạnh của đồ thị, cạnh {a,b} có thể được viết thành ab.

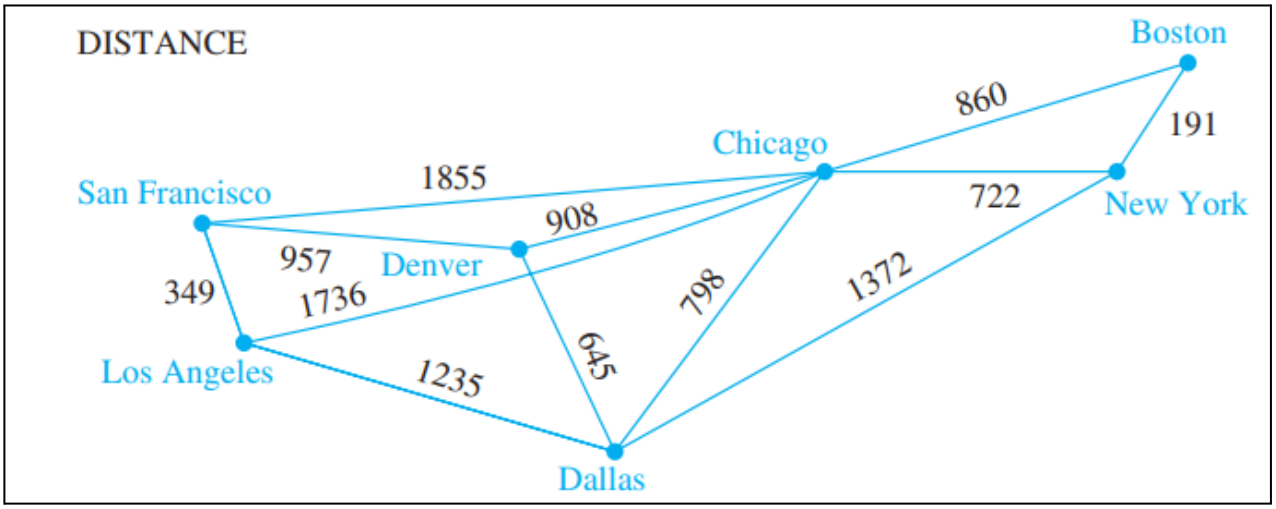
2.1.2. Các loại đồ thị

Một cạnh có thể có hướng hoặc không có hướng. Từ đó tạo nên các loại đồ thị như đồ thị vô hướng, đồ thị hữu hướng. Đồ thị vô hướng là đồ thị mà cạnh nối giữa 2 đỉnh được xem như là cặp đỉnh không sắp thứ tự, hay nói cách khác, một cạnh {a,b} được hiểu tương tự như {b,a}. Ta có thể thấy, hình 2.1 chính là ví dụ về đồ thị vô hướng. Đồ thị hữu hướng được biểu diễn như hình 2.2 là đồ thị mà trong đó cạnh nối giữa 2 đỉnh được xem như là cặp đỉnh có thứ tự, nghĩa là, cạnh {a,b} được quy định chặt chẽ đỉnh a sẽ ở trước đỉnh b, và cạnh {a,b} sẽ được phát biểu "bắt đầu từ a và kết thúc ở b".



Hình 2: Ví dụ về đồ thị hữu hướng - Nguồn: [2]

Mối quan tâm mà nhóm đặt trọng tâm sẽ là đồ thị vô hướng. Cụ thể hơn, đó là đồ thị vô hướng có trọng số. Một đồ thị được gọi là có trọng số khi có một số được gán cho mỗi cạnh của đồ thị.



Hình 3: Đồ thị vô hướng có trọng số - Nguồn: [2]

Mỗi cạnh ở hình trên được gán một trọng số, trọng số này thể hiện khoảng cách giữa hai đỉnh của đồ thị. Trọng số này có thể được sử dụng để tìm đường đi ngắn nhất giữa 2 đỉnh. Đó cũng chính là vấn đề mà nhóm sẽ giải quyết trong bài báo cáo này.

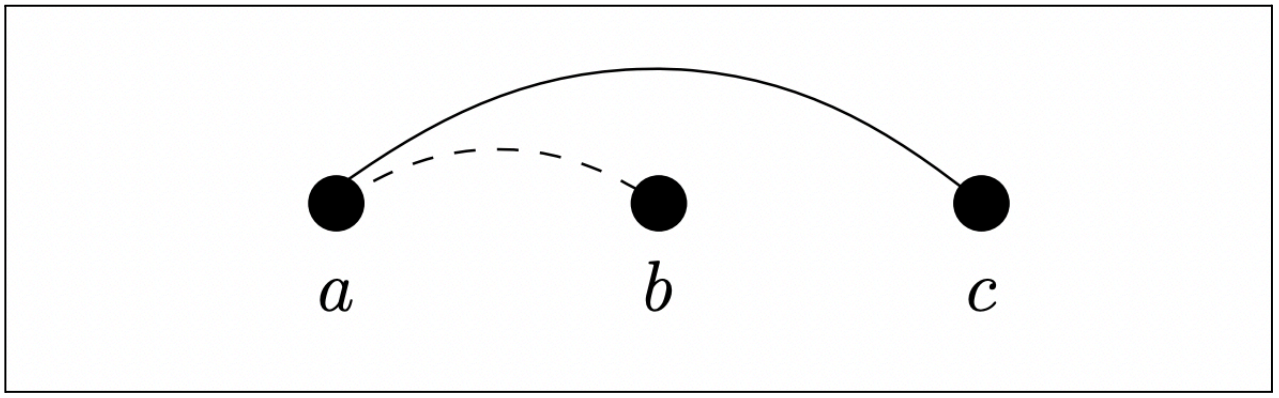
2.2. Tìm kiếm trên đồ thị (Graph search)

Trong số những thuật toán về đồ thị (Graph Algorithms) thì thuật toán tìm kiếm trên đồ thị (Graph Searching) là loại thuật toán đơn giản nhất. Tuy đây là dạng thuật toán đơn giản và được ứng dụng nhiều nhưng đến thời điểm hiện tại rất ít nghiên cứu chỉ ra nguyên lý rõ ràng về thứ tự duyệt đỉnh [4]. Theo [5], cách hoạt động chung của các thuật toán tìm kiếm trên đồ thị là bắt đầu từ một đỉnh gốc đã được chọn và tiến hành duyệt qua lần lượt các đỉnh và cạnh gần đó. Có rất nhiều thuật toán về tìm kiếm trên đồ thị, nhưng theo [6], các thuật toán này có thể được phân loại thành hai nhóm chính: Tìm kiếm cơ bản (uninformed search) và tìm kiếm với tri thức bổ sung (informed search).

2.2.1. Tìm kiếm cơ bản (Uninformed Search)

Theo [7], các thuật toán thuộc nhóm tìm kiếm cơ bản (Uninformed Search) còn được gọi là tìm kiếm mù, hoạt động theo nguyên tắc chỉ dựa trên định nghĩa vấn đề mà không có thông tin bổ sung về trạng thái hay không gian tìm kiếm. Các thuật toán này hoạt động bằng cách thử từng bước, tạo ra các trạng thái kế tiếp thông qua việc duyệt qua cây để phân biệt trạng thái mục tiêu và phi mục tiêu, mở rộng các nút tuần tự cho đến khi tìm ra giải pháp và dừng thuật toán lại. Điều này đồng nghĩa với việc chúng khám phá không gian tìm kiếm một cách mù quáng, kiểm tra mọi khả năng mà không ưu tiên hướng đi nào, bao gồm cả chi phí, thời gian hay một mục tiêu cụ thể nào đó.

Theo như ví dụ của Corneil và đồng nghiệp trong một bài nghiên cứu đã thể hiện rõ vấn đề của các thuật toán tìm kiếm cơ bản như sau : Trong một thuật toán tìm kiếm cơ bản (Uninformed Search), thứ tự chọn đỉnh có thể không dựa vào bất kỳ thông tin mang ý nghĩa nào [4]. Mặc dù có cạnh nối giữa a và c, đỉnh b vẫn được chọn trước c chỉ dựa trên cách thuật toán thực hiện việc mở rộng đỉnh mà không ưu tiên cụ thể về chi phí, khoảng cách, hay mục tiêu. Điều này thể hiện cách tiếp cận mù quáng của thuật toán khi lần lượt khám phá các đỉnh mà không có quy tắc hướng dẫn rõ ràng.



Hình 4: Ví dụ mô tả vấn đề "tìm kiếm mù" của các thuật toán Tìm kiếm cơ bản - Nguồn: [4]

Tất cả các thuật toán trong nhóm này đều được xây dựng dựa trên chiến lược không gian tìm kiếm và dưới đây là một số thuật toán phổ biến và được sử dụng nhiều trong nhóm tìm kiếm cơ bản (Uninformed Search) [8]:

- Tìm kiếm theo Chiều Rộng (Breadth-First Search - BFS): BFS duyệt các đỉnh theo từng mức, kiểm tra hết các đỉnh ở một cấp trước khi chuyển xuống cấp tiếp theo.

Điều này giúp tìm giải pháp nhanh nhất nếu tồn tại.

- Tìm kiếm theo Chiều Sâu (Depth-First Search - DFS): DFS ưu tiên khám phá chiều sâu của một nhánh trước, quay lại khi cần để thăm các nhánh khác. Phương pháp này tiết kiệm bộ nhớ nhưng có thể mắc kẹt nếu có vòng lặp vô hạn.

- Tìm kiếm Chiều Sâu Giới Hạn (Depth-Limited Search): Phiên bản DFS có giới hạn độ sâu, tránh vòng lặp vô hạn nhưng có thể bỏ lỡ giải pháp nếu nằm ngoài giới hạn.

- Tìm kiếm Chiều Sâu Lặp Lại (Iterative Deepening DFS): Kết hợp BFS và DFS, lặp DFS với độ sâu tăng dần để đảm bảo tìm được giải pháp ở mức thấp nhất nếu có.

2.2.2. Tìm kiếm với tri thức bổ sung (Informed Search)

Theo [6], Informed Search, hay còn gọi là Heuristic Search, là loại thuật toán tìm kiếm dùng thông tin bổ sung hoặc hàm heuristic để đánh giá các lựa chọn, giúp thuật toán ưu tiên những trạng thái gần với mục tiêu hơn. Đây là điểm khác biệt so với Uninformed Search, loại thuật toán thực hiện tìm kiếm mà không có chỉ dẫn cụ thể nào. Với Informed Search, các thuật toán như A* và Dijkstra có thể tìm giải pháp nhanh và hiệu quả hơn bằng cách tận dụng thông tin về vấn đề để rút ngắn thời gian tìm kiếm.

Một số ứng dụng của các loại thuật toán trong nhóm Informed Search [9]:

- Trí tuệ nhân tạo và robot: Informed Search giúp AI và robot tìm đường đi ngắn nhất, tối ưu hóa hoạt động di chuyển trong môi trường phức tạp. Ví dụ: Trong các nhà kho tích hợp công nghệ hiện đại, các robot được lập trình với thuật toán A* sẽ được sử dụng để lập kế hoạch lộ trình tối ưu cho việc lấy và vận chuyển hàng hóa, giúp giảm thiểu thời gian di chuyển và tránh va chạm với chướng ngại vật. Thuật toán này tính toán các lộ trình hiệu quả nhất, đảm bảo robot hoạt động một cách an toàn và hiệu quả trong môi trường làm việc phức tạp.

- Tối ưu hóa tuyến đường trong hệ thống GPS: Hệ thống định vị GPS chủ yếu dựa vào Informed Search để xác định tuyến đường nhanh nhất đến đích. Các thuật toán này tính đến khoảng cách, tình trạng giao thông và điều kiện đường xá để cung cấp lộ trình hiệu quả nhất. Ví dụ: Trong các ứng dụng như Google Maps, thuật toán A* và các thuật toán tương tự được sử dụng để tính toán lộ trình lái xe ngắn nhất, dựa trên dữ liệu giao thông theo thời gian thực để cập nhật lộ trình khi điều kiện thay

đổi.

- Giải quyết vấn đề trong trò chơi: Informed Search được sử dụng rộng rãi trong video game để điều khiển chuyển động của nhân vật và giải quyết các câu đố phức tạp. Những thuật toán này giúp các nhân vật không phải người chơi (NPC) đưa ra quyết định thông minh về việc di chuyển trong môi trường game. Ví dụ: Trong các trò chơi như Pacman, thuật toán A* được sử dụng để hướng dẫn chuyển động của các bóng ma, giúp chúng đuổi theo người chơi qua mê cung một cách hiệu quả nhất.

2.3. Giải thuật A*

2.3.1. Ý tưởng

A* là sự phát triển của thuật toán Dijkstra, một thuật toán nhằm xử lý việc lập kế hoạch đường đi hiệu quả giữa nhiều điểm (nút) bằng cách sử dụng hàm ước lượng heuristic. Peter Hart, Nils Nilsson và Bertram Raphael từ Viện Nghiên cứu Stanford đã giới thiệu thuật toán này vào năm 1968. A* sử dụng phương pháp tìm kiếm tốt nhất-đầu tiên và tìm ra đường đi có chi phí thấp nhất từ nút ban đầu đến một nút đích. A* đi qua đồ thị để xây dựng một cây đường đi từng phần. Các lá của cây này (gọi là tập hoặc cạnh mở) được lưu trữ trong một hàng đợi ưu tiên, trong đó các nút lá được sắp xếp theo hàm chi phí, kết hợp giữa ước tính heuristic của chi phí để đến đích và khoảng cách đã đi từ nút ban đầu [14].

A* giống với Thuật toán Dijkstra ở chỗ nó có thể được sử dụng để tìm đường đi ngắn nhất. A* cũng giống với Greedy Best-First-Search ở chỗ là A* có thể sử dụng một hàm ước lượng (heuristic) để dẫn đường. Trong trường hợp đơn giản, A* nhanh như Greedy Best-First-Search. Sự khác biệt dẫn đến hiệu suất của A* cao hơn các giải thuật kia là nó kết hợp các mảnh thông tin mà Thuật toán Dijkstra sử dụng (ưu tiên các đỉnh gần điểm xuất phát) và thông tin mà Greedy Best-First-Search sử dụng (ưu tiên các đỉnh gần mục tiêu) [15]. Trong thuật ngữ chuẩn thường dùng khi nói về A*, g(n) đại diện cho chi phí chính xác của đường đi từ điểm xuất phát đến bất kỳ đỉnh n nào, và h(n) đại diện cho chi phí ước lượng từ đỉnh n đến mục tiêu. A* cân bằng giữa hai yếu tố này khi di chuyển từ điểm xuất phát đến mục tiêu. Mỗi lần qua vòng lặp chính, A* xem xét đỉnh n có giá trị f(n) = g(n) + h(n) thấp nhất.

Thuật toán sử dụng hàm sau để tính toán khoảng cách ước tính từ nút hiện tại đến ô kết thúc, quyết định thứ tự ưu tiên của mỗi ô:

$$\mathbf{f(n) = g(n) + h(n)}$$

Giá trị f(n) ước tính khoảng cách từ ô hiện tại n để đến ô mục tiêu. Thuật toán sẽ chọn ô có giá trị f(n) thấp nhất để di chuyển trong bước tiếp theo. Trong đó, g(n) là khoảng cách từ nút hiện tại đến điểm bắt đầu tìm kiếm và h(n) là giá trị dự báo giữa ô hiện tại và ô mục tiêu (đây là yếu tố quyết định tính hiệu quả của thuật toán A*, hay còn gọi là hàm heuristic) [16]. Thuật toán A* chọn nút có giá trị f(n) thấp nhất từ danh sách mở (open list), đây sẽ là nút được duyệt tiếp theo trong quá trình thực hiện

Hàm Heuristic: Bằng cách so sánh giá trị hàm heuristic của từng nút, có thể kiểm soát tốc độ và độ chính xác của thuật toán. Trong một số trường hợp, không cần tìm đường ngắn nhất mà chỉ cần tìm đường nhanh nhất, điều này cho thấy sự linh hoạt của thuật toán A*. Dự đoán di chuyển càng gần giá trị thực tế thì đường đi cuối cùng sẽ càng giống với đường ngắn nhất thực sự [16]. Đối với hàm heuristic, chỉ số thường được sử dụng nhiều là khoảng cách Euclid, khoảng cách Manhattan và khoảng cách Chebyshev [17]. Trong đó:

Khoảng cách Euclid tính căn bậc hai của hiệu số bình phương giữa các tọa độ của một cặp đối tượng [18]. Về mặt toán học, nó có thể được biểu diễn là:

$$d_{(x,y)} = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

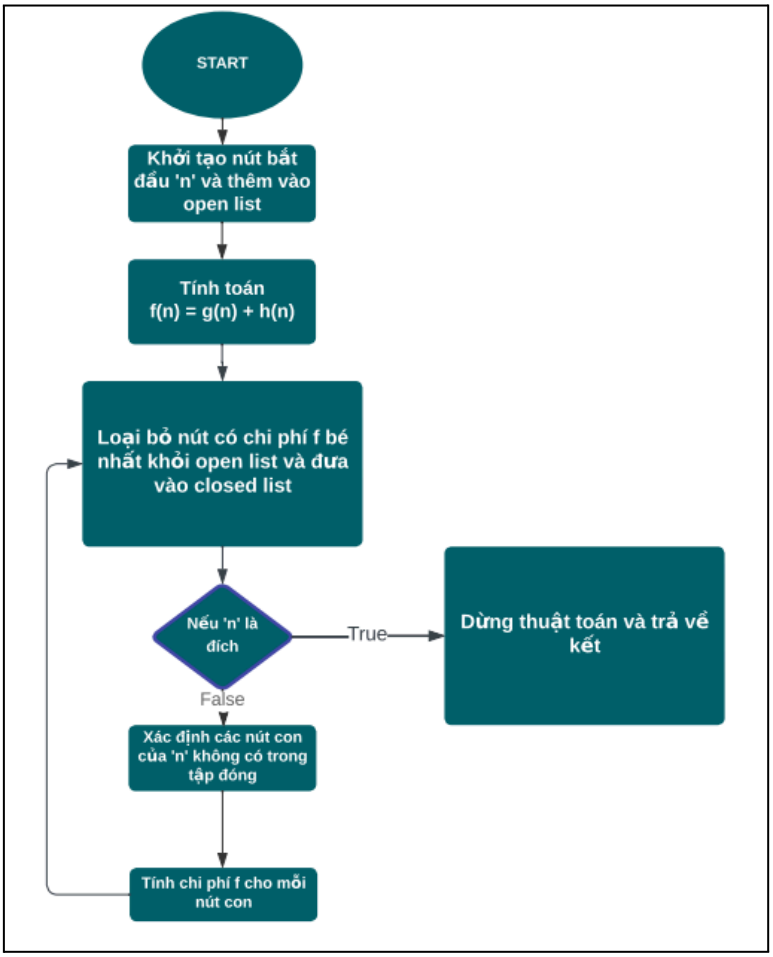
Khoảng cách Manhattan tính hiệu số tuyệt đối giữa các tọa độ của một cặp đối tượng [18]. Nó có thể biểu diễn dưới dạng sau:

$$d_{(x,y)} = \sum_{i=1}^{n} |(x_i - y_i)|$$

Ngoài ra, nhóm cũng sử dụng khoảng cách Chebyshev, được biểu diễn dưới dạng:

$$d_{(x,y)} = max(|x_i - y_i|)$$

2.3.2. Flowchart



Hình 5: Flowchart của thuật toán A - Nguồn: Doan Thanh Xuan và cộng sự (2022) [19]*

Dưới đây là phân tích của các phần của Sơ đồ khối ở hình 2.4.1 minh họa các bước của thuật toán A*.

Bắt đầu là việc **Khởi tạo và Thêm Nút Bắt Đầu vào Danh Sách Mở**. Thuật toán bắt đầu bằng cách khởi tạo điểm bắt đầu (nút) n và đặt nó vào danh sách mở. Danh sách mở lưu trữ các nút có sẵn để khám phá, được sắp xếp theo thứ tự ưu tiên. Tiếp theo là **Tính Hàm Chi Phí f(n) = g(n) + h(n)** với Hàm chi phí f(n) được tính cho nút hiện tại n, trong đó g(n) biểu thị chi phí để di chuyển từ nút bắt đầu đến nút hiện tại, và h(n) là hàm heuristic, ước tính chi phí để đạt đến mục tiêu từ nút hiện tại. Sau khi tính toán xong hàm chi phí thì **Di Chuyển Nút Có Giá Trị Thấp Nhất vào Danh Sách Đóng**. Đoạn này thì thuật toán loại bỏ nút có giá trị f nhỏ nhất khỏi danh sách mở và đưa nó vào danh sách đóng. Nút này là lựa chọn hứa hẹn nhất vì nó có chi phí ước tính thấp nhất để đạt đến mục tiêu. Cuối cùng là **Kiểm Tra Nếu Nút n Là Mục Tiêu**. Nếu nút hiện tại n là điểm đích, thuật toán kết thúc và đường đi tối ưu được tái tạo lại bằng cách lần theo các con trỏ từ mục tiêu đến điểm bắt

đầu. Nếu n không phải là điểm đích, thuật toán tiến hành xác định tất cả các nút kế thừa của n (các nút lân cận chưa được khám phá). Bất kỳ nút kế thừa nào chưa có trong danh sách đóng sẽ được thêm vào danh sách mở để khám phá tiếp theo. Các bước này bao gồm:

Mở Rộng Các Nút Kế Thừa: Nếu n không phải là điểm đích, thuật toán tiến hành xác định tất cả các nút kế thừa của n (các nút lân cận chưa được khám phá). Bất kỳ nút kế thừa nào chưa có trong danh sách đóng sẽ được thêm vào danh sách mở để khám phá tiếp theo.

Tính Hàm Chi Phí f cho Mỗi Nút Kế Thừa: Hàm chi phí f được tính cho mỗi nút kế thừa, kết hợp chi phí di chuyển từ điểm bắt đầu và ước tính heuristic đến mục tiêu. Các nút này sau đó được ưu tiên trong danh sách mở dựa trên giá trị f của chúng.

Chu trình này lặp đi lặp lại, khám phá các nút theo thứ tự giá trị f, cho đến khi đạt đến nút mục tiêu, đảm bảo rằng đường đi có chi phí thấp nhất được tìm thấy. Các danh sách mở và đóng quản lý các nút một cách hiệu quả, với danh sách mở lưu trữ các nút chưa được khám phá và danh sách đóng theo dõi các nút đã được xét, giúp tránh tái khám phá không cần thiết.

2.4. Đánh giá

Đối với A*, nhóm cũng sẽ đánh giá dựa trên hai tiêu chí là độ phức tạp cũng như tính hoàn chỉnh và tính tối ưu. Ngoài ra, nhóm cũng sẽ đánh giá A* dựa trên hai tiêu chuẩn sau,

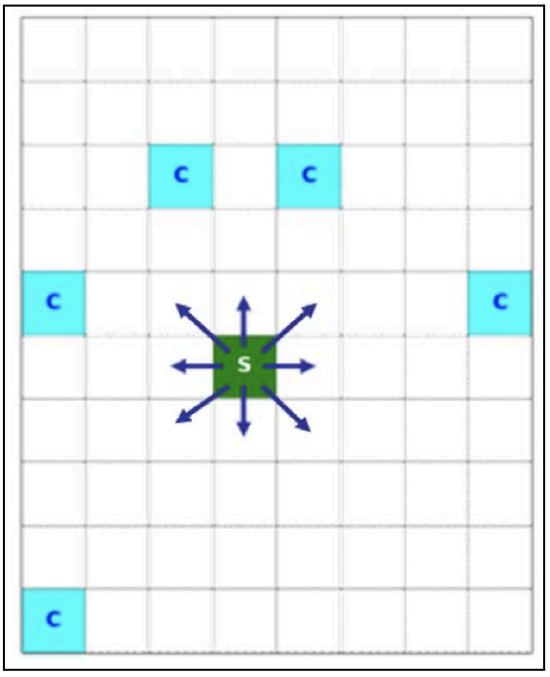
Một, đó là thời gian chạy trong thực tế của hai giải thuật A*. Quá trình đánh giá sẽ được tiến hành bằng cách chạy thuật toán trên một thiết bị và đo lường thời gian thực của hai thuật toán này.

Hai, tổng khoảng cách của đường đi ngắn nhất mà thuật toán tìm được. Với kết quả đường đi ngắn nhất mà hai giải thuật trả về, nhóm sẽ đo lường tổng khoảng cách của đường đi đó và thực hiện đánh giá.

CHƯƠNG III: KẾT QUẢ THỰC NGHIỆM

3.1. Phát biểu bài toán

Giả sử ta có một không gian là một hình chữ nhật, hình chữ nhật này được phân chia thành các ô nhỏ tạo thành một ma trận $A_{m,n}$ như hình dưới,



*Hình 6: Minh họa bài toán hút bụi với A**

Tại một thời điểm, các ô trong ma trận này có thể có các trạng thái sau: *free* (F) hoặc *dirty (D)* hoặc *clean (C)*. Trạng thái *free* nghĩa là ô đó không phải là ô dơ, không phải là vị trí bắt đầu của robot và không phải là ô dơ nhưng đã được dọn. Một ô mang trạng thái *dirty* thể hiện cho ta biết đó là ô dơ và một ô mang trạng thái *clean* nói cho ta biết đó là ô dơ nhưng đã được dọn. Giả sử ta có một con robot hút bụi có khả năng thực hiện hai hành động: (i) di chuyển *(Move)* đến một 1 trong 8 ô liền kề theo khung 3x3 và (ii) thực hiện thao tác hút bụi *(Suck)* để làm sạch ô đang đứng nếu ô đó là ô dơ sao cho thỏa mãn các điều kiện sau,

Một, gốc tọa độ (1,1) nằm ở vị trí góc dưới bên trái,

Hai, robot được khởi động và đặt tại một ô bất kỳ,

Ba, mỗi lần robot chỉ thực hiện 1 hành động và chi phí tiêu tốn cho một hành động di chuyển (*Move*) là 1,

Bốn, sau mỗi hành động của robot, mỗi ô mang trạng thái *dirty* còn lại chưa được hút bụi

sẽ bị tăng chi phí thêm 1 để làm sạch (*Suck*).

Bài toán đặt ra ở đây, đó là áp dụng thuật toán A* để tìm đường đi với tổng chi phí bé nhất nhằm hướng dẫn cho robot đi tới và dọn sạch các ô dơ mà vẫn thỏa mãn các điều kiện trên.

3.2. Mô hình hóa

Để có thể tìm ra lời giải cho bài toán hút bụi sử dụng giải thuật A* đề cập ở trên, nhóm sẽ tiến hành mô hình hóa cho bài toán của nhóm thông qua các lớp sau: Position, Cell và VacuumProblem.

Class	Mô tả
Position	Đại diện cho vị trí của một ô trong bài toán, là lớp biểu diễn tọa độ dưới dạng (x,y)
Cell	Thể hiện cho một ô trong bài toán, là lớp biểu diễn một ô với các thông tin về ô như trạng thái, vị trí, chi phí.
VacuumProblem	Biểu diễn cho ma trận bài toán, là không gian tìm kiếm của giải thuật.

*Bảng 1: Class sử dụng đối với giải thuật A**

Đối với lớp Position, ta sẽ có các thông tin sau,

Function	Loại	Mô tả
__init__	Constructor	Khởi tạo object của lớp Position với hai thuộc tính x và y đại diện cho vị trí của một ô trong không gian tìm kiếm.

Bảng 2: Lớp Position

Đối với lớp Cell, ta sẽ có các thông tin sau,

Function	Loại	Mô tả
__init__	Constructor	Khởi tạo object của lớp Cell. Lưu trữ thông tin của một ô thông qua các thuộc tính position cho vị trí, status cho trạng thái của ô, g cho chi phí thực, h cho chi phí ước lượng và f cho tổng chi phí của g và h.

Bảng 3: Lớp Cell

Đối với lớp VacuumProblem, ta sẽ có,

Function	Loại	Mô tả
__init__	Constructor	Khởi tạo object của lớp VacuumProblem. Lưu trữ thông tin của không gian tìm kiếm thông qua các thuộc tính matrix cho ma trận tìm kiếm, robot_initial cho ô bắt đầu của robot, initial_state cho ô của các ô dơ và heufunc cho hàm heuristic muốn sử dụng.
actions	Method	Nhận tham số là một Cell và trả về những hành động có thể thực hiện được của Cell đó trong khung 3x3.
result	Method	Yêu cầu tham số là một Cell và hành động thực hiện đối với Cell đó, trả về kết quả là một Cell mới

		sau khi thực hiện hành động.
heuristic	Method	Tính toán chi phí heuristic giữa một Cell và một Cell khác

Bảng 4: Lớp VacuumProblem

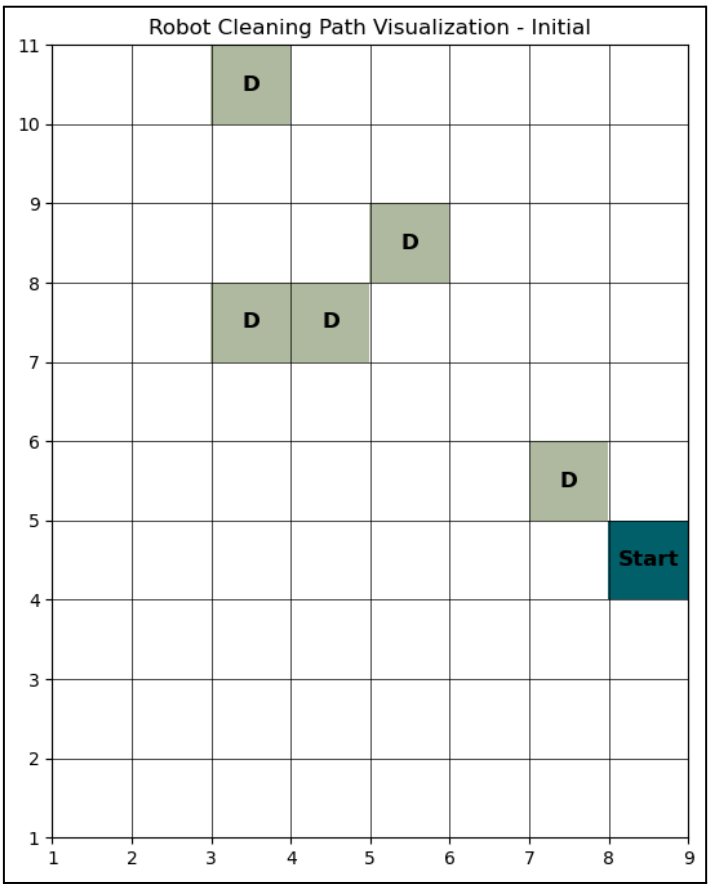
Như vậy, với các bảng trên, bài toán sẽ được mô hình hóa thông qua các lớp Position cho vị trí, Cell cho một ô và VacuumProblem cho không gian bài toán. Nhóm sẽ xây dựng hàm astar() dựa trên ba lớp này.

astar() sẽ nhận vào một đối tượng của lớp VacuumProblem. Khi đó hàm này sẽ lưu thông tin về vị trí của robot, về vị trí của ô dơ. Với các thông tin đó, nhóm sẽ kiểm soát vòng lặp bằng cách sử dụng một biến đếm số lượng ô dơ đã được dọn, mỗi lần dọn sẽ trừ biến đó đi 1 cho đến khi biến đó bằng 0 nghĩa là các ô dơ đã được dọn sạch. Hàm astar() của nhóm sẽ được tinh chỉnh một chút so với giải thuật A* được giới thiệu ở chương II. Đó là nhóm sẽ không sử dụng closed list là tập các đỉnh đã duyệt qua. Lý do cho việc này là bởi nhóm xét đến khía cạnh tối ưu chi phí, nếu cho phép duyệt qua các ô mà robot đã đi qua sẽ phần nào giảm bớt tổng chi phí so với việc đi một đường khác do điều kiện không được đi qua đỉnh đã thăm. Hơn nữa, với chiến lược giải quyết bài toán được giới thiệu ở mục 3.3, nếu không cho phép đi qua đỉnh đã thăm thì có trường hợp sẽ rơi vào vòng lặp vô hạn.

3.3. Kết quả đạt được

Xét trường hợp một không gian tìm kiếm có cấu hình sau, robot hút bụi sẽ làm sạch các ô dơ trên ma trận $A_{10,8}$ với số lượng ô dơ là 5 ở các vị trí sau: (5,7), (7,4), (7,3), (8,5), (10,3).

Những ô dơ sẽ được tô màu xanh xám với nhãn là “D”, robot hút bụi sẽ ở vị trí có màu xanh rêu đậm với nhãn là “Start” như hình dưới đây,



Hình 7: Cấu hình ban đầu của bài toán hút bụi

Để có thể hướng dẫn robot đến và làm sạch những ô dơ này mà vẫn thỏa mãn các điều kiện của bài toán và giải thuật A* thì nhóm sẽ có chiến lược như sau,

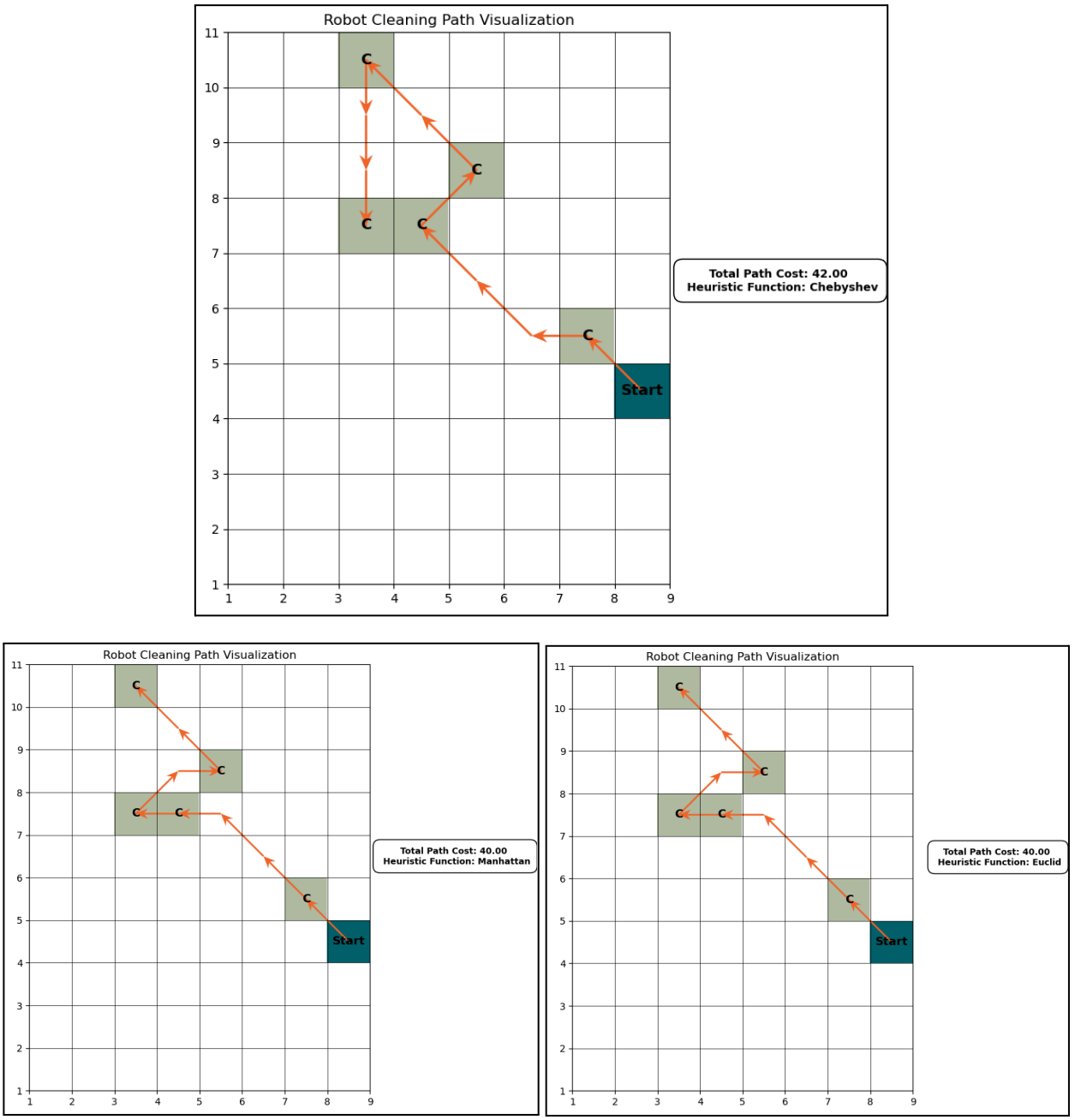
Thứ nhất, nhóm sẽ tìm ô dơ gần nhất với robot hút bụi bằng cách so sánh tổng của chi phí hiện tại và chi phí ước lượng từ robot đến ô dơ, ô dơ nào có chi phí bé nhất sẽ được coi là ô dơ gần nhất.

Thứ hai, sau khi tìm được ô dơ gần nhất, nhóm sẽ xác định những hành động nào có thể được thực hiện để có thể đi tới ô dơ gần nhất.

Thứ ba, với tập những hành động có thể được thực hiện, nhóm sẽ chọn ra hành động nào có tổng chi phí di chuyển và chi phí ước lượng từ hành động đó đến ô dơ là bé nhất. Cần lưu ý rằng, hành động ở đây là hành động di chuyển theo khung 3x3, không bao gồm hành động “Suck”.

Thứ tư, robot sẽ tiếp tục tìm kiếm hành động có chi phí nhỏ nhất để đi tới ô dơ gần nhất và khi đã tới ô dơ cần tìm, robot sẽ làm sạch ô dơ đó, gán trạng thái ô dơ này là “clean” và loại ô dơ này ra khỏi danh sách các ô dơ cần làm sạch, sau đó tiếp tục thực hiện như quy

trình trên. Và với chiến lược đó, hàm ước lượng khoảng cách nhóm sẽ sử dụng đó là "Euclid", "Manhattan" và "Chebyshev".



Hình 8: Lời giải cho bài toán hút bụi với các hàm heuristic khác nhau

Khi giải thuật chạy xong, đường đi của robot sẽ được biểu diễn bằng những mũi tên màu cam và những ô dơ được dọn sẽ được thể hiện bởi chữ cái "C" như hình trên.

CHƯƠNG IV: ĐÁNH GIÁ

4.1. Độ phức tạp

Nhóm sẽ đánh giá độ phức tạp của A* trên hai khía cạnh, thứ nhất đó là độ phức tạp không gian và thứ hai, độ phức tạp thời gian. Vì giải thuật A* mà nhóm thực hiện tìm kiếm trên cây, do đó, trước khi đánh giá, nhóm sẽ gọi *b* là hệ số phân nhánh tối đa tức số nhánh lớn nhất mà một nút trong cây có thể có, gọi *d* là độ sâu của lời giải có chi phí thấp nhất.

Đối với độ phức tạp không gian, do A* lưu trữ tất cả các nút trong bộ nhớ, vậy nếu gọi tất cả các nút trong bộ nhớ là n thì độ phức tạp không gian sẽ là O(n).

Đối với độ phức tạp thời gian, do đối với bài toán hút bụi, chiến lược mà nhóm chọn sẽ phải tìm ô dơ gần nhất trước sau đó mới tìm kiếm đường đi tới ô dơ này. Vì vậy, với hành động thứ nhất - chọn ô dơ, nếu gọi *m* là số lượng ô dơ thì số lần xét các ô dơ sẽ là tổng của dãy số (1 + 2 + … + (m - 1) + m) hay $\frac{m(m+1)}{2}$ và do đó, O($\frac{m(m+1)}{2}$). Sau khi tìm được ô dơ gần nhất, giải thuật sẽ tìm các hành động có thể thực hiện được trong khung 3x3, vậy có 8 hành động tối đa, hệ số phân nhánh tối đa *b* sẽ là 8, vì vậy số lượng các nút được sinh ra là 8^d và do đó, O(8^d). Và từ đó, ta sẽ có độ phức tạp thời gian là O($\frac{m(m+1)}{2} + 8^d$).

4.2. Tính hoàn chỉnh và tính tối ưu

Nếu không gian trạng thái là hữu hạn và có giải pháp để tránh rơi vào lặp các trạng thái tới vô hạn thì giải thuật A* sẽ đạt được tính hoàn chỉnh [12]. Thiếu một trong hai điều kể trên thì A* được coi là không hoàn chỉnh. Song, đối với bài toán hút bụi mà nhóm phát biểu ở trên cũng như chiến lược tìm kiếm mà nhóm xây dựng, dễ thấy rằng không gian bài toán là hữu hạn (ma trận A với kích thước (m,n)) và nhóm sẽ kiểm soát trạng thái để tránh rơi vào trường hợp lặp vô hạn thông qua biến đếm số lượng các ô dơ, mỗi lần một ô được dọn biến này sẽ giảm đi một và giảm dần tới 0 thì quá trình tìm kiếm sẽ kết thúc. Như vậy, A* là hoàn chỉnh trong phạm vi bài toán đã được phát biểu.

Nghiệm của bài toán phần nào phụ thuộc vào vị trí bắt đầu của robot, giả sử robot bắt đầu ở vị trí xa so với các ô dơ, thì nghiễm nhiên rằng, chi phí sẽ lớn hơn so với một robot bắt đầu ở vị trí gần các ô dơ. Một trường hợp khác, đó là giả sử hai ô có cùng chi phí, khi tính toán chọn ô dơ gần nhất, giải thuật A* mà nhóm xây dựng sẽ chọn ô dơ đầu tiên. Do vậy mà lời giải sẽ bị ảnh hưởng bởi việc ô dơ nào sẽ được chọn trước và dẫn đến tổng chi phí

thay đổi. Với luận điểm đó, A* là không tối ưu đối với phạm vi của bài toán.

4.3. Thời gian chạy và chi phí

Xét bài toán hút bụi với cấu hình như ở hình 7, nhóm tiến hành đo lường thời gian trung bình chạy của giải thuật với ba hàm heuristic Euclidean, Manhattan và Chebyshev thu được kết quả,

Hàm heuristic	Thời gian (ms)
Euclidean	4.28
Manhattan	3.87
Chebyshev	2.82

Bảng 5: Thời gian chạy trung bình của A với các hàm heuristic khác nhau khi số lượng ô dơ tăng*

Với bảng trên, Chebyshev là hàm cho ta thời gian trung bình chạy nhanh nhất. Vậy nếu ta tăng kích thước ma trận lên và giữ nguyên số lượng ô dơ thì kết quả sẽ như thế nào, nhóm phỏng đoán Chebyshev vẫn là hàm cho kết quả nhanh nhất, hãy cùng với nhóm kiểm chứng điều này, nhóm sẽ tiến hành tăng kích thước (m,n) của ma trận mỗi lần chạy giải thuật lên một đơn vị gọi là Δ với cấu hình ma trận ban đầu là $A_{m,n}$. Mỗi lần chạy giải thuật cặp (m,n) - tăng lên Δ đơn vị, khi đó ta có một không gian mới $A_{m+\Delta,\ n+\Delta}$. Ở đây, nhóm sẽ đặt Δ = 1 và như vậy, ta có $A_{m+1,\ n+1}$.

Chọn (m,n) = (10,8) như cấu hình ở hình 7, giữ nguyên số lượng ô dơ là 5, chỉ thay đổi kích thước ma trận, điểm dừng sẽ là khi (m,n) = (50,48), ta có được kết quả,

Hàm heuristic	Thời gian (ms)
Euclidean	5.35
Manhattan	3.27
Chebyshev	2.16

Bảng 6: Thời gian chạy trung bình của A với các hàm heuristic khác nhau khi thay đổi kích thước ma trận*

Với kết quả có được ở hai bảng trên, xét về mặt thời gian chạy thì dường như Chebyshev là hàm cho thời gian chạy trung bình nhanh nhất. Tuy vậy, như ở hình 8 thì Chebyshev lại

có chi phí cao nhất trong ba hàm nhưng đây chỉ là một trường hợp cụ thể, hãy làm rõ điều này hơn bằng cách xem xét khi tăng số lượng ô dơ và thay đổi kích thước ma trận thì chi phí sẽ thay đổi ra sao.

Vẫn sử dụng cấu hình như đã đề cập, liệu khi tăng số lượng ô dơ, chi phí bình giữa ba hàm heuristic có gì thay đổi hay không,

	Chi phí trung bình	
Hàm heuristic	**Tăng số lượng ô dơ**	**Tăng kích thước ma trận**
Euclidean	1037. 56521	191.25
Manhattan	1091.13768	202.475
Chebyshev	1043.3043	186.775

Bảng 7: Chi phí trung bình của A với các hàm heuristic khác nhau khi tăng số lượng ô dơ và khi tăng kích thước ma trận*

Dường như khi tăng số lượng ô dơ lên, hàm Euclidean cho ta kết quả khả quan hơn so với hai hàm còn lại, Manhattan lại là hàm cho ta chi phí cao nhất. Tuy vậy, trong trường hợp kích thước ma trận và không thay đổi số lượng ô dơ thì điều gì sẽ xảy ra. Thú vị thay, khi tăng số lượng ô dơ Euclidean là hàm có chi phí nhỏ nhất trong khi tăng kích thước ma trận Chebyshev lại cho ta chi phí nhỏ nhất song chênh lệch giữa hai hàm này không phải là một con số quá lớn, và trong cả hai trường hợp Manhattan vẫn giữ nguyên vị trí của mình là hàm cho chi phí cao nhất.

CHƯƠNG V: KẾT LUẬN

5.1. Về A*

Như vậy, thông qua tìm hiểu giải thuật A* cho bài toán hút bụi. Chiến lược của nhóm để giải quyết bài toán đó là chọn ô dơ gần nhất, sau đó tìm đường đi phù hợp nhất đến ô này và lặp lại quá trình này đến khi các ô dơ đã được dọn sạch.

Với chiến lược như vậy, nhóm đánh giá giải thuật A* dựa trên độ phức tạp về thời gian và không gian, tính hoàn chỉnh và tối ưu cũng như thời gian chạy và chi phí trung bình của giải thuật khi thay đổi số lượng ô dơ và kích thước ma trận.

Độ phức tạp về thời gian và không gian của giải thuật A* trong phạm vi bài toán lần lượt là O(n) và O($\frac{m(m+1)}{2} + 8^d$). Nhóm đã chứng minh được tính hoàn chỉnh của giải thuật với luận điểm là không gian bài toán hút bụi hữu hạn và tránh được việc lặp ô hạn thông qua biến đếm số lượng ô dơ đã được dọn nhưng thất bại trong việc chứng minh A* đạt được tính tối ưu. Trong quá trình xem xét giải thuật về khía cạnh thời gian và chi phí với ba hàm heuristic khác nhau khi tăng số lượng ô dơ và kích thước ma trận, nhóm nhận thấy Chebyshev là hàm cho thời gian chạy trung bình nhanh nhất ở cả hai trường hợp. Euclidean là hàm cho chi phí trung bình nhỏ nhất khi tăng số lượng ô dơ nhưng chi phí trung bình nhỏ nhất khi xét đến tăng kích thước ma trận lại thuộc về Chebyshev.

Ưu điểm chính của A* đến từ chi phí ước lượng h(n) trong công thức tính chi phí để chọn đường đi, f(n) = g(n) + h(n). Điều này giúp cho quá trình tìm kiếm lời giải của giải thuật trở nên chính xác hơn khi sử dụng thêm thông tin ước lượng từ h(n) so với các giải thuật tìm kiếm mù khác. Nhưng đổi lại, A* lại phụ thuộc vào hàm heuristic, nếu hàm heuristic không tốt, tức cho một ước lượng có chênh lệch quá lớn so với thực tế thì giải thuật có thể trở nên kém hiệu quả hơn mặc dù tính hoàn chỉnh vẫn được đảm bảo.

5.2. Hướng phát triển

Đối với bài toán hút bụi sử dụng thuật toán A* mà nói, hướng phát triển chính của đề tài chính là việc cải tiến hàm heuristic để tính toán tối ưu hơn chi phí ước lượng. Trong bài nghiên cứu của nhóm tác giả đã sử dụng khoảng cách Chebyshev, Manhattan và Euclidean để tính toán cho hàm heuristic. Hướng phát triển của đề tài có thể là nghiên cứu sử dụng thêm khoảng cách Octile hoặc các hàm heuristic khác để cải thiện chi phí ước lượng, qua

đó giúp thuật toán A* tiến tới gần lời giải tối ưu hơn và tốt hơn nữa, đó là đạt được lời giải toàn cục.

TÀI LIỆU THAM KHẢO

[1] Michael T. Goodrich, Roberto Tamassia, Michael H. Goldwasser (2013). Chapter 14: Graph Algorithms. *Data Structures & Algorithms in Python*, Wiley, page 620 - 686.

[2] Kenneth H. Rosen (2011). Chapter 10: Graphs. *Discrete Mathematics and Its Applications*, seventh edition, McGraw Hill, page 641 - 735.

[3] Lê Minh Hoàng (2002). Chương 4: Bài toán đường đi ngắn nhất. Giải thuật & lập trình, Trang 238 - 241.

[4] Corneil, D. G., & Krueger, R. M. (2008). A unified view of graph searching. *SIAM Journal on Discrete Mathematics*, *22*(4), 1259-1276.

[5] Tarjan, R. (1972). Depth-first search and linear graph algorithms. *SIAM Journal on Computing, 1*(2), 146–160

[6] Algasi, K. A. (2017). *Uninformed and Informed Search Techniques in Artificial Intelligence* (Master's thesis, Eastern Mediterranean University (EMU)-Doğu Akdeniz Üniversitesi (DAÜ)).

[7] Kapi, A. Y., Sunar, M. S., & Zamri, M. N. (2020). A review on informed search algorithms for video games pathfinding. *International Journal*, *9*(3).

[8] Dục Đoàn Trình. (2022, March 25). *Thuật toán tìm kiếm Uninformed Search Algorithms*. w3seo. Retrieved October 29, 2024, from https://websitehcm.com/thuat-toan-tim-kiem-uninformed-search-algorithms/

[9] Applied AI. (2024, September 23). *Informed Search Algorithms in Artificial Intelligence*. Applied AI Course. Retrieved October 29, 2024, from https://www.appliedaicourse.com/blog/informed-search-algorithms-in-artificial-intelligence/

[10] Trần Quốc Chiến, Nguyễn Thanh Tuấn, ALGORITHMS OF THE PROBLEM OF FINDING THE SHORTEST PATHS FROM A SET OF NODES TO ANOTHER SET OF NODES **https://lrel.ued.udn.vn/vi/tu-sach-giang-vien/tin-hoc/nguyen-thanh-tuan-31.html?download=1&id=1**

[11] Robert Sedgewick, Kevin Wayne (2010). Algorithms fourth edition. Page 652

[12] Estefania Cassingena Navone. (28/09/2020). "Dijkstra's Shortest Path Algorithm - A Detailed and Visual Introduction"

[13] Russell, S. J., Norvig, P., & Davis, E. (2010). *Artificial intelligence: A modern approach*. Prentice Hall.

[14] Candra, A., Budiman, M. A., & Pohan, R. I. (2021, June). Application of a-star algorithm on pathfinding game. In Journal of Physics: Conference Series (Vol. 1898, No. 1, p. 012047). IOP Publishing, from: https://iopscience.iop.org/article/10.1088/1742-6596/1898/1/012047/pdf

[15] Introduction to A*. (2024, September 8). Stanford CS Theory. Retrieved November 22, 2024, from https://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html

[16] Yan, Y. (2023). Research on the A Star Algorithm for Finding Shortest Path. Highlights in Science, Engineering and Technology, 46, 154-161.

[17] Liu, L., Wang, B., & Xu, H. (2022). Research on path-planning algorithm integrating optimization A-star algorithm and artificial potential field method. Electronics, 11(22), 3660. Truy cập tại: https://www.mdpi.com/2079-9292/11/22/3660

[18] Ranjitkar, H. S., & Karki, S. (2016). Comparison of A*, Euclidean and Manhattan distance using Influence map in MS. Pac-Man.Truy cập tại: https://www.diva-portal.org/smash/get/diva2:918778/FULLTEXT02

[19] Xuan, D. T., Nam, L. G., Viet, D. T., & Thang, V. T. (2021, December). A-star algorithm for robot path planning based on digital twin. In *Regional Conference in Mechanical Manufacturing Engineering* (pp. 83-90). Singapore: Springer Nature Singapore. Truy cập tại: https://www.researchgate.net/publication/360981613_A-star_Algorithm_for_Robot_Path_Planning_Based_on_Digital_Twin

[20] GIẢI THUẬT TÌM ĐƯỜNG ĐI NGẮN NHẤT GIỮA HAI TẬP ĐỈNH. (n.d.). TT Học liệu & Công nghệ thông tin. Retrieved November 21, 2024, from https://lrel.ued.udn.vn/vi/tu-sach-giang-vien/tin-hoc/nguyen-thanh-tuan-31.html?download=1&id=1

PHỤ LỤC

1. Cách chạy code

Trước khi chạy thuật toán, người dùng nên tải các thư viện cần thiết, để xem các thư viện cần dùng cho thuật toán, hãy xem ở cell import các thư viện. Nếu người dùng chưa tải thư viện nào, hãy mở terminal lên và pip install <tên thư viện>.

Khi chạy code, vì là file .ipynb nên người dùng hãy bấm Run All để chạy hết các cell. Trong quá trình chạy, chương trình sẽ yêu cầu người dùng nhập số dòng, số cột và số lượng ô dơ. Khi nhập xong, chương trình sẽ thể hiện kết quả ra cho người dùng.

2. Link dự phòng

Trong trường hợp file .ipynb trong file .RAR không chạy được, người dùng hãy nhấp vào đây để đi tới thư mục chứa code. Tiến hành tải về và chạy thuật toán.

ĐÁNH GIÁ CÔNG VIỆC

1. Đặng Thị Thu Hiền

Công việc	Mức độ hoàn thành
Tìm hiểu về bài toán hút bụi và A*	100%
Viết báo cáo: Chương I	100%
Slides dùng để trình chiếu: tìm template	100%
Slides dùng để trình chiếu: Chương I, Chương II (2.1, 2.2, 2.4), Chương IV, Chương V	100%
Test Code A*	100%

2. Bùi Tiến Hiếu

Công việc	Mức độ hoàn thành
Tìm hiểu về bài toán hút bụi và A*	100%
Test code A*	100%

3. Đỗ Thanh Hoa

Công việc	Mức độ hoàn thành
Tìm hiểu về bài toán hút bụi và A*	100%
Viết báo cáo: Chương II - 2.2	100%
Slides dùng để trình chiếu: Chương II (2.3), Chương III	100%
Format lại báo cáo	100%
Test Code A*	100%

4. Nguyễn Huy Hoàng

Công việc	Mức độ hoàn thành
Tìm hiểu về bài toán hút bụi và A*	100%
Viết báo cáo: Chương II - 2.1, 2.4; Chương III, Chương IV, Chương V (5.1)	100%
Xây dựng các class cho thuật toán A* và code giải thuật A*	100%
Test Code A*	100%

5. Nguyễn Trương Hoàng

Công việc	Mức độ hoàn thành
Tìm hiểu về bài toán hút bụi và A*	100%
Viết báo cáo: Chương I - 2.3, Chương V (5.2)	100%
Visualisation cho giải thuật A*: biểu diễn không gian bài toán và lời giải	100%
Test Code A*	100%