**ĐẠI HỌC BÁCH KHOA HÀ NỘI**

**TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG**

**\*\*\*\*\*\*\*\***

|  |  |
| --- | --- |
| **Thành viên** | **Công việc** |
| Vũ thường đạt | Xây dựng các hàm support thuật toán, xây dựng thuật toán a\*, dfs, |
| Dương Đức Huy | Xây dựng các hàm support thuật toán, xây dựng thuật toán dfs, viết cáo báo, làm slide,tạo testcase, thống kê |
| Nguyễn Vă n Nhâm | Xây thuật toán bfs, a\*, |
| Phạm Đình Hải | Xây dựng thuật toán a\*, bfs, design |
| Dương Hữu Hiếu | Xây dựng thuật toán a\*, tạo checkpoint. |

BÁO CÁO BÀI TẬP LỚN

**Học phần: Trí tuệ nhân tạo  
Sinh viên thực hiện**

**Giảng viên hướng dẫn:** TS. Trần Thế Hùng

**Nhóm số:** 19

**Hà Nội, tháng 5 năm 2024**

# I ) Mở đầu:

Trí tuệ nhân tạo đang phát triển theo hướng tốc độ không thể tin được trong thập kỷ này. Trong đó Game AI, dựa trên dựa nhiều vào các thuật toán AI cũng đang được cải thiện đáng kể theo xu hướng này  
Điển hình với sự hỗ trợ của các kỹ thuật học sâu, AlphaGo và người kế nhiệm của nó là AlphaZero đã chinh phục boardgame Cờ vây, trò chơi bị con người thống trị vì thế kỉ. Trong bài viết này, chúng em muốn tập trung vào một trò chơi truyền thống đơn giản hơn, Sokoban. Chúng ta sẽ thảo luận về cách để mô hình hóa vấn đề này và cố gắng khám phá khả năng giải quyết vấn đề này bằng cách sử dụng AI khác nhau thuật toán.

# II. Mô hình bài toán

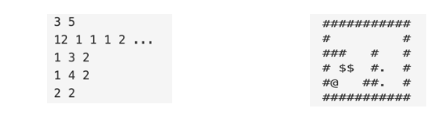
A. Các kí hiệu trong testcase

'#' là kí hiệu của các bức tường

'$' là viết tắt của hộp,

‘.’ là kí hiệu của mục tiêu

‘@’ là kí hiệu của người di chuyển



## **B. Xây dựng trạng thái giải quyết bài toán:**

**Định nghĩa trạng thái của vấn đề**: Trong mỗi trò chơi, chúng ta có thể nhận thấy rằng giải pháp của câu đố là một chuỗi các bước di chuyển từng bước một. Sự khác biệt giữa hai trạng thái là vị trí của người chơi và các hộp. Do đó, chúng ta định nghĩa một trạng thái của vấn đề là một tập hợp các vị trí của người chơi và các hộp.

**Các trạng thái kế tiếp**: Mỗi trạng thái phụ thuộc vào một bước di chuyển của người giữ kho. Một trạng thái có thể có tối đa bốn hàng xóm/trạng thái kế tiếp mà có thể khám phá trong bước di chuyển tiếp theo.

**Các bước di chuyển hợp lệ**: Một bước di chuyển hợp lệ bao gồm các tình huống sau:

* (1) Chỉ có người giữ kho di chuyển và anh ta di chuyển đến một vị trí trống trong phạm vi tường bao.
* (2) Người giữ kho di chuyển và đẩy một hộp đến một vị trí mới. Cả người giữ kho và hộp đều không vi phạm các tường và các ràng buộc của hộp.



## **C. Lập mô hình bài toán tìm kiếm đồ thị**

* **Trạng thái ban đầu**: Đầu tiên, ta có một trạng thái ban đầu cho vấn đề, trong đó chứa các vị trí ban đầu của người giữ kho và các hộp.
* **Các hành động**: Mỗi hành động được biểu diễn bằng sự thay đổi vị trí của người giữ kho và các hộp. Trong Sokoban, các hành động này thường là di chuyển của người giữ kho đến các vị trí khác hoặc đẩy hộp đến các vị trí mới.
* **Trạng thái kết thúc**: Mục tiêu của vấn đề là tìm ra một trạng thái cuối cùng mà tất cả các hộp đều được đặt đúng vào vị trí mục tiêu của chúng.
* **Kiểm tra mục tiêu**: Để kiểm tra xem một trạng thái có phải là trạng thái kết thúc hay không, ta thực hiện việc so khớp giữa các hộp và vị trí mục tiêu của chúng.
* **Chi phí của đường đi**: Chi phí của một đường đi từ trạng thái ban đầu đến trạng thái hiện tại được tính bằng số lần di chuyển, hoặc số bước đi.
* **Đỉnh**: các trạng thái có thể tiếp cận
* **Cạnh**: sự chuyển đổi hợp pháp giữa trạng thái này đến trạng thái khác
* **Đồ thị**: có hướng, có thể có chu trình

Bắt đầu từ trạng thái ban đầu của trò chơi Sokoban, ta có thể điều tra các trạng thái tiếp theo có thể đạt được bằng cách thực hiện các hành động di chuyển của người giữ kho và đẩy hộp. Để đạt được trạng thái mục tiêu, chúng ta sử dụng đệ quy, tức là tiếp tục áp dụng quá trình tìm kiếm từ trạng thái hiện tại cho đến khi đạt được trạng thái mục tiêu hoặc không thể đi tiếp được nữa.

* **Nhược điểm của mô hình**: So với cách giải quyết trực quan của con người khi giải câu đố Sokoban, mô hình trên có nhược điểm là tiêu tốn nhiều không gian. Điều này có nghĩa là trong quá trình giải quyết bài toán, việc lưu trữ và quản lý các trạng thái trung gian và các đường đi có thể làm tăng đáng kể khối lượng bộ nhớ yêu cầu.
* **Giải pháp sử dụng hàm heuristic**: Để giải quyết vấn đề tiêu thụ không gian, ta áp dụng các hàm heuristic khác nhau vào các thuật toán thử nghiệm. Hàm heuristic giúp đưa ra dự đoán thông minh về chi phí hoặc độ ưu tiên của các hành động tiếp theo dựa trên trạng thái hiện tại, giúp thuật toán quyết định các hướng đi tiếp theo một cách hiệu quả hơn và giảm thiểu việc tiêu tốn không gian lưu trữ.

# III. Thuật toán

**Đa dạng của lời giải**: Một câu đố Sokoban có thể có nhiều cách giải khác nhau. Các giải pháp này có thể khác nhau đáng kể về độ dài các bước di chuyển hoặc thời gian tìm kiếm.

**Yếu tố cần tính đến để tối ưu hóa thuật toán**:

**Tổng chiều dài các bước di chuyển**: Một thuật toán tối ưu cần giảm thiểu số lượng bước di chuyển cần thiết để hoàn thành câu đố Sokoban. Điều này có ý nghĩa quan trọng đặc biệt khi đối mặt với các bản đồ lớn, giúp hệ thống tiết kiệm không gian bộ nhớ và tăng hiệu quả của thuật toán.

**Thời gian tìm kiếm giải pháp**: Thời gian để tìm kiếm một giải pháp cũng là một yếu tố chính quyết định chất lượng của thuật toán. Trong các trò chơi tìm kiếm đồ thị như Sokoban, độ phức tạp thời gian có thể dẫn đến hiện tượng lặp vô hạn nếu không có giải pháp, hoặc dẫn đến việc không thể hoàn thành vì quá nhiều thời gian.

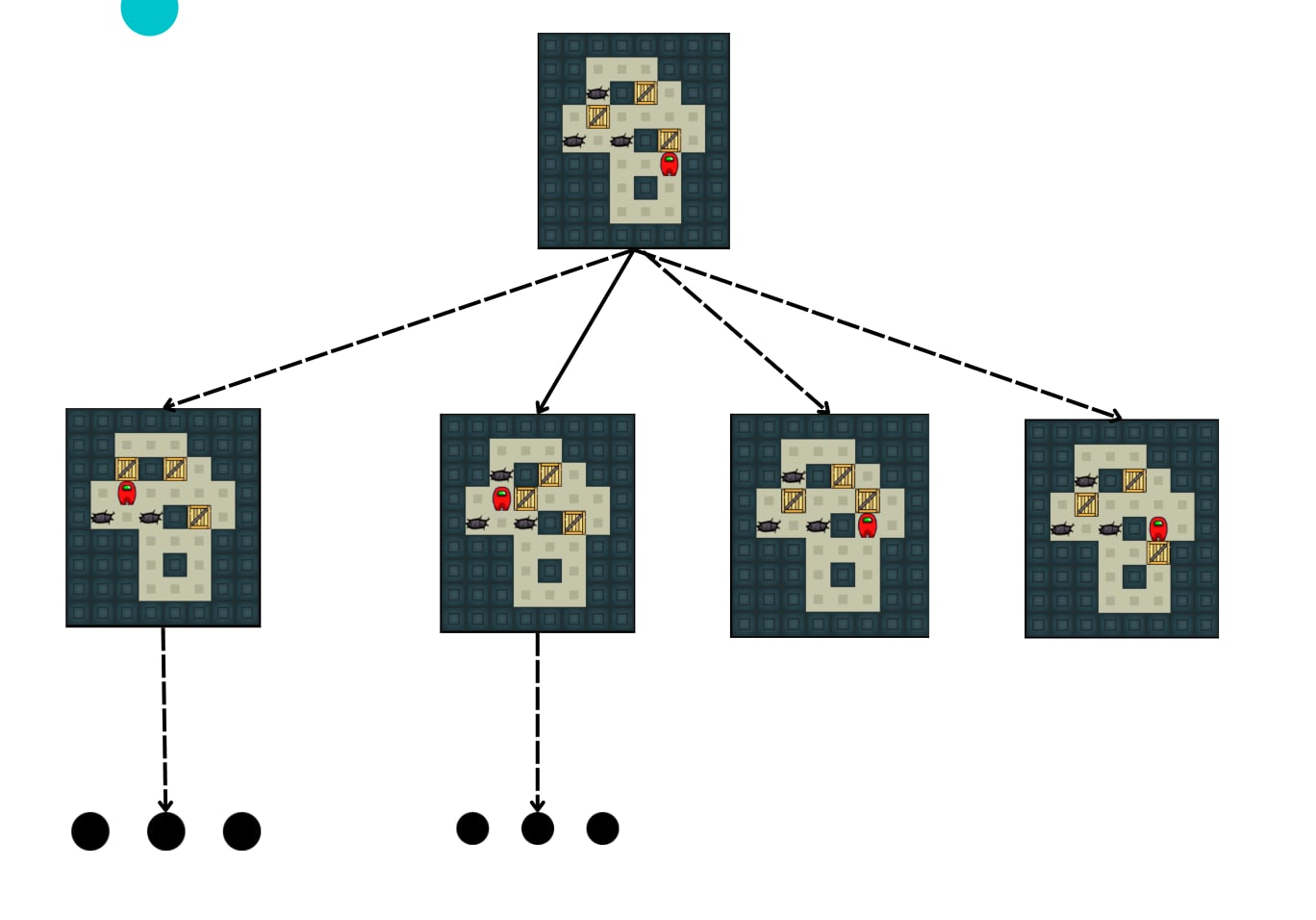
**Áp dụng các thuật toán khác nhau**:

BFS (Breath First Search) - Tìm kiếm theo chiều rộng

DFS (Depth First Search) - Tìm kiếm theo chiều sâu

T thtìm kiếm A\* (A-star)

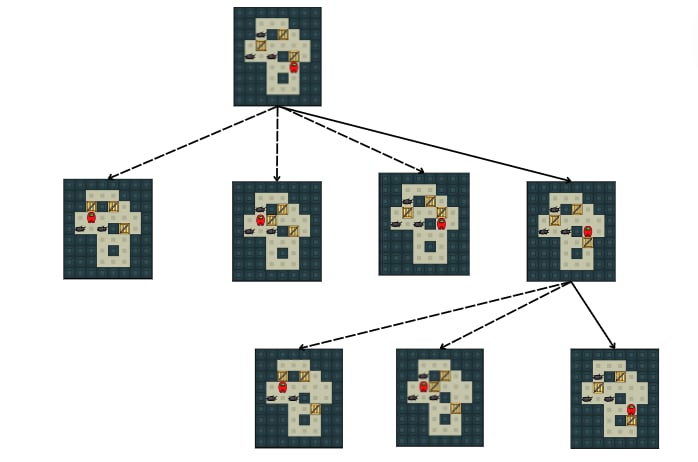
### **A. BFS (Breath First Search)**



1. **Định nghĩa của BFS**: BFS là một thuật toán để duyệt hoặc tìm kiếm một cây hoặc đồ thị. Quá trình tìm kiếm bắt đầu từ một nút gọi là nút gốc (root), và nó khám phá tất cả các đỉnh kề hoặc các trạng thái kế tiếp cùng một cấp độ trước khi đi tiếp đến các trạng thái ở cấp độ tiếp theo.
2. **Tính chất của BFS**: BFS luôn hoàn chỉnh vì nó áp dụng kiểm tra từng cấp độ một, nghĩa là nó không bao giờ bỏ sót một giải pháp. Khi áp dụng kiểm tra từng cấp độ này, BFS sẽ liên tục duyệt qua các trạng thái cho đến khi tìm thấy giải pháp hoặc đã duyệt qua tất cả các nút mà không thấy giải pháp.
3. **Vấn đề của BFS trong bài toán cụ thể**: Tuy nhiên, thời gian và độ phức tạp của BFS có thể là vấn đề. Thuật toán có thể tốn nhiều thời gian để xử lý và dễ dàng gặp vấn đề về không gian nhớ (heap memory). Trong vấn đề này, hệ số nhánh (branch factor) của mỗi trạng thái là 4 (lên, xuống, trái, phải). Mỗi cấp độ của BFS sẽ kiểm tra tất cả các trạng thái kế tiếp của một trạng thái và duyệt qua cho đến khi tìm được giải pháp hoặc đã kiểm tra hết tất cả các nút.

Đối mặt với các vấn đề về thời gian xử lý và không gian lưu trữ, đặc biệt là khi áp dụng cho các vấn đề có không gian tìm kiếm lớn.

### **B. DFS (Depth First Search)**



1. **Định nghĩa của DFS**: DFS bắt đầu tìm kiếm từ một nút gốc và khám phá càng xa càng tốt theo một trong các nút kế tiếp trước khi quay lại (backtrack). Thuật toán này tiêu tốn thời gian nhưng chỉ chiếm không gian tuyến tính. Nó có xu hướng tìm kiếm nhanh hơn so với BFS trong thực tế.
2. **Đặc điểm của DFS**:

DFS thường tạo ra các giải pháp nhanh hơn so với BFS vì nó khám phá một nhánh một cách sâu và đến khi không còn nhánh nào để khám phá nữa thì mới quay lại.

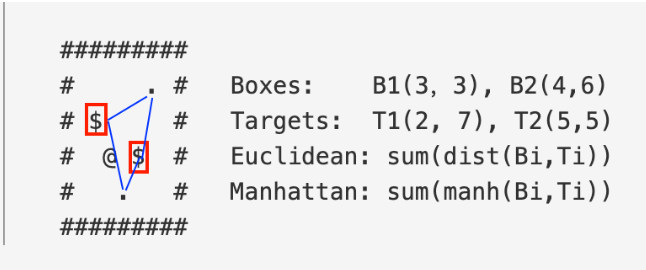
Tuy nhiên, DFS thường tạo ra các giải pháp dài hơn, bao gồm nhiều bước di chuyển. Điều này có thể là không hiệu quả trong một số trường hợp như trong câu đố Sokoban, khi muốn giải quyết với số bước di chuyển ít nhất có thể.

Một vấn đề của DFS là có thể rơi vào vòng lặp vô hạn trong một nhánh của đồ thị, dù giải pháp có thể nằm ở gốc của nhánh tiếp theo. Điều này xảy ra khi m (độ sâu tối đa tìm kiếm) thấp hơn d (độ sâu của đồ thị).  
DFS là một thuật toán tìm kiếm mạnh mẽ và có thể hiệu quả hơn BFS trong nhiều trường hợp, nhưng cần cân nhắc đến khả năng tạo ra các giải pháp dài và khả năng rơi vào vòng lặp vô hạn

### **C. Heuristic**

Một hàm heuristic là một hàm tính toán chi phí xấp xỉ từ trạng thái ban đầu đến trạng thái mục tiêu. Trong bối cảnh của các trò chơi, các hàm heuristic thường khác nhau tùy thuộc vào từng trường hợp cụ thể. Thường thì một trò chơi có thể có nhiều hàm heuristic khác nhau. Một hàm heuristic tốt sẽ có xu hướng ước tính chính xác khoảng cách từ một trạng thái đến trạng thái mục tiêu, từ đó chỉ ra hướng đi tối ưu và giảm độ phức tạp của thuật toán tìm kiếm.

Trong câu đố Sokoban, chúng ta sử dụng hai hàm heuristic khả dĩ:



1. **Euclidean distances (Khoảng cách Euclid)**: Hàm heuristic này tính tổng khoảng cách Euclid giữa các hộp và các vị trí mục tiêu của chúng. Khoảng cách Euclidean là khoảng cách trực tiếp giữa hai điểm trong không gian Euclid. Hàm này giả định rằng người chơi có thể di chuyển hộp một cách trực tiếp đến vị trí mục tiêu một cách nhanh chóng nhất.
2. **Manhattan distances (Khoảng cách Manhattan)**: Hàm heuristic này tính tổng các giá trị tuyệt đối của sự khác biệt giữa các vị trí của hộp và vị trí mục tiêu của chúng, dựa trên phép tính khoảng cách Manhattan. Khoảng cách Manhattan là tổng của độ dài các đoạn ngang và dọc giữa hai điểm trên một lưới.

## **D) A\***

Thuật toán A\* là một thuật toán duyệt đồ thị và tìm kiếm đường đi, thường được sử dụng trong các vấn đề tìm kiếm nhờ vào những ưu điểm như tính toàn vẹn và tối ưu hóa. Đây là một thuật toán tìm kiếm thông minh (informed search), có nghĩa là nó không chỉ xem xét chi phí đường đi từ một nút đến mục tiêu mà còn kết hợp nó với chi phí thực tế để đến được nút hiện tại. Thuật toán A\* sử dụng PriorityQueue để quản lý các trạng thái dựa trên chi phí tổng hợp (f(n) = g(n) + h(n)), trong đó g(n) là chi phí đã di chuyển và h(n) là chi phí ước tính còn lại.

g(n): độ sâu

h(n): khoảng cách manhattan

A video game with a cartoon character

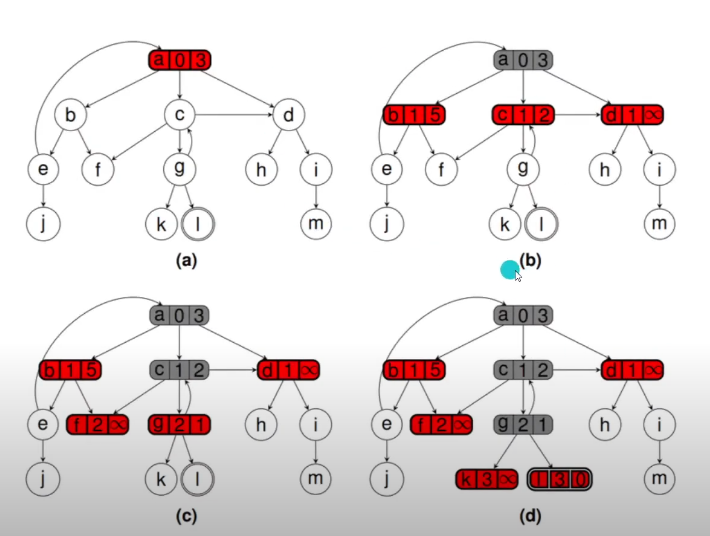
Description automatically generated

A video game with a blue and black bug

Description automatically generated

A video game with a blue and white object

Description automatically generated



Level1:

A screenshot of a computer

Description automatically generated

A screenshot of a video game

Description automatically generated

Kết quả:

A black background with white numbers

Description automatically generated

A screenshot of a video game

Description automatically generated

**Rút ra:**

* **Tính tối ưu:** A\* đảm bảo tìm kiếm một lời giải tối ưu, có nghĩa là đường đi tìm được sẽ có chi phí thấp nhất.
* **Tính toàn vẹn:** A\* sẽ luôn tìm thấy một giải pháp nếu có giải pháp tồn tại.
* **Hàm heuristic:** Để A\* hoạt động hiệu quả, hàm heuristic phải đáp ứng được yêu cầu về tính tối ưu và chặt chẽ, tức là nó phải đánh giá gần đúng và không đánh giá quá cao chi phí đến mục tiêu.

# **V. Phân tích hiệu suất**

**Đánh giá dựa trên:**

**Số lượng trạng thái đã khám phá**: Đây là một phép đo quan trọng để biết mỗi thuật toán đã khám phá bao nhiêu trạng thái khác nhau trong quá trình tìm kiếm giải pháp. Mỗi trạng thái thường tương ứng với một bước trong quá trình giải quyết bài toán. Thuật toán nào khám phá ít trạng thái hơn thường có hiệu suất tốt hơn về mặt không gian và thời gian.

**Thời gian tìm kiếm**: Phép đo này đo lường thời gian mà mỗi thuật toán sử dụng để tìm kiếm giải pháp. Đây là yếu tố quan trọng để đánh giá hiệu suất thời gian của các thuật toán, đặc biệt là khi đối mặt với các bài toán có kích thước lớn và phức tạp.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Testcase | BFS | | DFS | | A\* | |
| State | Time | State | Time | State | Time |
| 1 | 51 | 0.0427 | 43 | 0.018 | 40 | 0.034 |
| 2 | 265 | 0.779 | 161 | 0.165 | 176 | 0.367 |
| 3 | 257 | 0.508 | 200 | 0.25 | 218 | 0.3769 |
| 4 | 365 | 1 | 154 | 0.135 | 233 | 0.341 |
| 5 | 588 | 1,444 | 530 | 1.171 | 508 | 1.177 |
| 6 | timeout | timeout | 2268 | 36.5 | 4552 | 176 |
| 7 | timeout | timeout | 23.36 | 2001 | 3204 | 36.56 |
| 8 | Not found | | | | | |
| 9 | 87 | 0.115 | 41 | 0.018 | 44 | 0.038 |
| 10 | Not found | | | | | |
| 11 | 198 | 0.513 | 113 | 0.118 | 130 | 0.16 |
| 12 | Not found | | | | | |
| 13 | 1144 | 15,464 | 1121 | 14 | 696 | 4.39 |
| 14 | 106 | 0.087 | 47 | 0.05 | 95 | 0.12 |
| 15 | 295 | 0.419 | 88 | 0.0517 | 267 | 0.37 |
| 16 | 122 | 0.1870 | 109 | 0.11 | 70 | 0.069 |
| 17 | 407 | 0.8893 | 316 | 0.5426 | 305 | 0.583 |
| 18 | 135 | 0.123 | 113 | 0.105 | 103 | 0.114 |
| 19 | 312 | 0.415 | 251 | 0.326 | 274 | 0.59 |
| 20 | 123 | 0.130 | 77 | 0.055 | 108 | 0.132 |
| 21 | Not found | | | | | |
| 22 | 28 | 0.0135 | 26 | 0.015 | 29 | 0.014 |
| 23 | 89 | 0.048 | 78 | 0.054 | 53 | 0.035 |
| 24 | 39 | 0.017 | 33 | 0.009 | 40 | 0.021 |
| 25 | Not found | | | | | |
| 26 | 364 | 0.608 | 103 | 0.06 | 86 | 0.058 |
| 27 | Not found | | | | | |
| 28 | 559 | 1,417 | 126 | 0.107 | 85 | 0.068 |
| 29 | 62 | 0.036 | 40 | 0.012 | 49 | 0.036 |
| 30 | 552 | 1,446 | 462 | 0.996 | 327 | 0.59 |

**A**\*: Cả hai thuật toán này được cho là hiệu quả hơn so với các thuật toán còn lại. A\* có hiệu suất tốt nhất. Tuy nhiên, trong thực tế, thuật toán A\* không luôn cung cấp giải pháp tối ưu. Điều này xuất phát từ việc hàm heuristic (Euclidean và Manhattan distances) không luôn đảm bảo tính chất nhất quán (consistent), đặc biệt là khi số lượng hộp tăng lên.

**BFS** Các thuật toán này được kỳ vọng sẽ luôn tìm ra giải pháp tối ưu (nếu có) vì chi phí thực sự của mỗi bước là hằng định và đã được chứng minh bằng các trường hợp thử nghiệm.

**DFS**: Được cho là có hiệu suất tồi nhất vì chi phí mỗi bước cao hơn đáng kể so với các thuật toán khác. Tuy nhiên, thời gian chạy của DFS thường thấp hơn BFS.