ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

ASSIGNMENT 2

HỌC PHẦN: TRÍ TUỆ NHÂN TẠO

ĐỀ TÀI: XÂY DỰNG THUẬT TOÁN UCS và A STAR CHO TRÒ CHƠI SOKOBAN

Giảng viên hướng dẫn: Lương Ngọc Hoàng

Lớp: Trí tuệ nhân tạo - CS106.O22

Sinh viên thực hiện: Nguyễn Vũ Khai Tâm – 22521293

TP. Hồ Chí Minh, tháng 3, năm 2023

I. Hàm Heuristic

- Hàm heuristic là hàm ước chi phí để đi từ trạng thái hiện tại đến trạng thái đích. Điều này giúp các thuật toán như A* hoặc các biến thể của Greedy xác định những bước di chuyển có khả năng dẫn đến giải pháp nhanh nhất, từ đó hướng dẫn quá trình tìm kiếm hiệu quả hơn.
- Một heuristic tốt cần đáp ứng các tiêu chuẩn sau:

  + Khả thi (Admissible): Heuristic không ước lượng chi phí vượt quá số lượng di chuyển thực tế từ trạng thái hiện tại đến trạng thái đích.

  + Tính nhất quán (Consistent): Giá trị heuristic từ một nút này sang nút khác không bao giờ lớn hơn tổng chi phí thực sự để di chuyển giữa hai nút đó cộng với heuristic từ nút thứ hai đến nút đích.
- Việc sử dụng heuristic trong Sokoban giúp giảm bớt số lượng trạng thái cần phải xem xét, cho phép giải pháp được tìm thấy nhanh hơn, đồng thời tối ưu hóa quá trình tìm kiếm, đặc biệt quan trọng trong trò chơi có nhiều khả năng di chuyển và cấu hình trạng thái.

1. Hàm heuristic_Manhattan (hàm sẵn có trong code mẫu)

```python
def heuristic_Manhattan(posPlayer, posBox):
    # print(posPlayer, posBox)
    """A heuristic function to calculate the overall distance between the else boxes and the else goals"""
    distance = 0
    completes = set(posGoals) & set(posBox)  # Tìm các hộp đã ở trên mục tiêu
    sortposBox = list(set(posBox).difference(completes)) # Loại bỏ các hộp đã hoàn thành khỏi danh sách hộp cần di chuyển
    sortposGoals = list(set(posGoals).difference(completes)) # Loại bỏ các mục tiêu đã được hộp chiếm từ danh sách mục tiêu
    for i in range(len(sortposBox)):
        distance += (abs(sortposBox[i][0] - sortposGoals[i][0])) + (abs(sortposBox[i][1] - sortposGoals[i][1]))
    return distance
```

- Hàm bắt đầu bằng việc loại bỏ những hộp đã đặt đúng chỗ ra khỏi danh sách tổng thể của hộp và mục tiêu. Điều này thực hiện qua việc tạo ra hai danh sách mới: một danh sách cho những hộp cần di chuyển (gọi là sortposBox) và một danh sách cho những mục tiêu chưa được đặt hộp đúng chỗ (gọi là sortposGoals).
- Sau đó, hàm này tính tổng khoảng cách Manhattan giữa mỗi hộp và mục tiêu. Khoảng cách Manhattan được tính là tổng giá trị tuyệt đối của hiệu số tọa độ x (hàng) và tọa độ y (cột) giữa hộp và mục tiêu.

$$\text{Heuristic Manhattan} = \sum_{\text{mỗi hộp}} \left(|x_{\text{hộp}} - x_{\text{mục tiêu}}| + |y_{\text{hộp}} - y_{\text{mục tiêu}}| \right)$$

2. Hàm heuristic_Euclid (hàm cải tiến)

```python
def heuristic_Euclid(posPlayer, posBox):
    # print(posPlayer, posBox)
    """A heuristic function to calculate the overall distance between the else boxes and the else goals"""
    distance = 0
    completes = set(posGoals) & set(posBox)  # Tìm các hộp đã ở trên mục tiêu
    sortposBox = list(set(posBox).difference(completes)) # Loại bỏ các hộp đã hoàn thành khỏi danh sách hộp cần di chuyển
    sortposGoals = list(set(posGoals).difference(completes)) # Loại bỏ các mục tiêu đã được hộp chiếm từ danh sách mục tiêu
    for i in range(len(sortposBox)):
        distance += math.sqrt((sortposBox[i][0] - sortposGoals[i][0])**2 + (sortposBox[i][1] - sortposGoals[i][1])**2)
    return distance
```

- Hàm này cũng bắt đầu bằng việc loại bỏ những hộp đã đặt đúng chỗ ra khỏi danh sách tổng thể của hộp và mục tiêu. Điều này thực hiện qua việc tạo ra hai danh sách mới: một danh sách cho những hộp cần di chuyển (gọi là sortposBox) và một danh sách cho những mục tiêu chưa được đặt hộp đúng chỗ (gọi là sortposGoals).
- Sau đó, hàm này sử dụng khoảng cách Euclid (còn gọi là khoảng cách theo đường chim bay) để ước lượng chi phí di chuyển từ hộp đến mục tiêu. Khoảng cách này được tính bằng cách lấy căn bậc hai của tổng bình phương hiệu số tọa độ x và y giữa hộp và mục tiêu.

$$\text{Heuristic Euclid} = \sum_{\text{mỗi hộp}} \sqrt{(x_{\text{hộp}} - x_{\text{mục tiêu}})^2 + (y_{\text{hộp}} - y_{\text{mục tiêu}})^2}$$

II. Cài đặt thuật toán A Star Search

- A Star (A*) Search là một thuật toán tìm kiếm đường đi thông minh, khác với UCS, thuật toán A* không chỉ tập trung vào tổng chi phí từ trạng thái bắt đầu đến trạng thái hiện tại, mà còn kết hợp thêm ước lượng chi phí (heuristic) để đến được trạng thái mục tiêu. Điều này giúp A* ưu tiên mở rộng những node có tổng chi phí (chi phí đã đi cộng với ước lượng) thấp nhất, thông qua việc sử dụng hàng đợi ưu tiên (PriorityQueue) trong quản lý các trạng thái trên frontier và actions.
- Tương tự như UCS, A* cũng được triển khai theo Graph Search, sử dụng frontier để lưu trữ các node cần xét, actions để theo dõi chuỗi hành động dẫn đến từng node tương ứng, và exploredSet để ghi nhận các trạng thái đã được khám phá. Sự khác biệt chính là việc A* đánh giá và ưu tiên các node trong frontier như đã trình bày ở trên. Trong mỗi lần lặp, một node được lấy ra từ frontier để xét, nếu trạng thái của node là trạng thái kết thúc, quá trình tìm kiếm kết thúc và lời giải được trả về. Nếu không, tìm các hành động hợp lệ từ trạng thái hiện tại để tạo ra các node mới và nhờ sự trợ giúp của hàm isFailed() để loại trừ

các trạng thái không mong muốn. Quá trình này lặp lại cho đến khi tìm thấy lời giải hoặc khi frontier trống rỗng, tức là không có lời giải nào khả dĩ từ trạng thái bắt đầu.

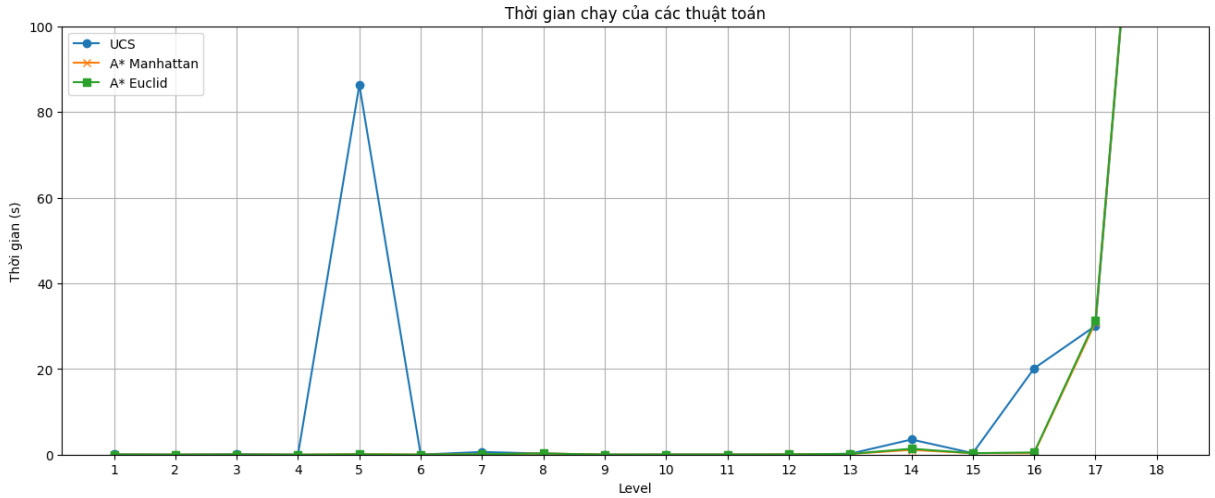
III. Thống kê

Bảng thống kê về thời gian,số bước đi và số nút đã được mở ra của các thuật toán được cài đặt trong game Sokoban qua từng level

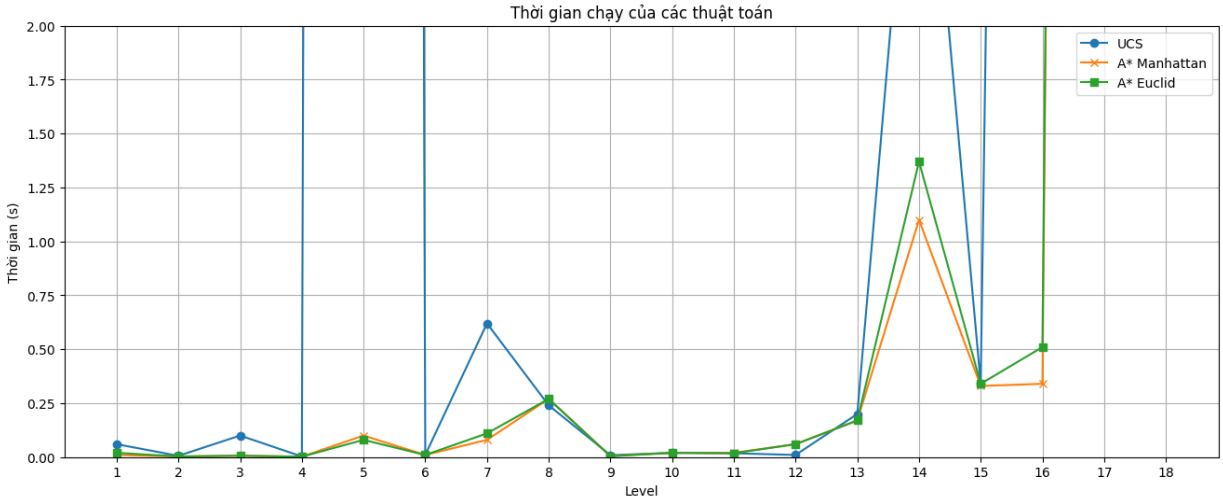
LEVELS	Thời gian (s)			Số bước đi			Số nút đã được mở ra		
	UCS	**A* heuristic Manhattan**	**A* heuristic Euclid**	**UCS**	**A* heuristic Manhattan**	**A* heuristic Euclid**	**UCS**	**A* heuristic Manhattan**	**A* heuristic Euclid**
1	0.06	0.01	0.02	12	13 (*)	12	720	122	223
2	0.006	0.003	0.003	9	9	9	64	39	39
3	0.10	0.007	0.006	15	15	15	509	54	49
4	0.003	0.002	0.002	7	7	7	55	29	29
5	86.32	0.10	0.08	20	22 (*)	20	357203	485	349
6	0.01	0.01	0.01	19	19	19	250	208	219
7	0.62	0.08	0.11	21	21	21	6046	715	1097
8	0.24	0.27	0.27	97	97	97	2383	2352	2365
9	0.009	0.005	0.004	8	8	8	74	42	42
10	0.02	0.02	0.02	33	33	33	218	198	198
11	0.018	0.02	0.018	34	34	34	296	284	284
12	0.01	0.06	0.06	23	23	23	1225	563	628
13	0.20	0.17	0.17	31	31	31	2342	1699	1820
14	3.52	1.10	1.37	23	23	23	26352	8108	10057
15	0.34	0.33	0.34	105	105	105	2505	2183	2261
16	20.14	0.34	0.51	34	42 (*)	36 (*)	57275	1286	1927
17	30.07	30.9	31.4	**0**	**0**	**0**	71595	71595	71595
18	**infinity**	**infinity**	**infinity**	**-**	**-**	**-**	**-**	**-**	**-**

- ***Chú thích: (*) là lời giải (đường đi) không tối ưu***

- **Thời gian**



Hình 1: Thời gian tìm ra lời giải của các thuật toán qua các màn



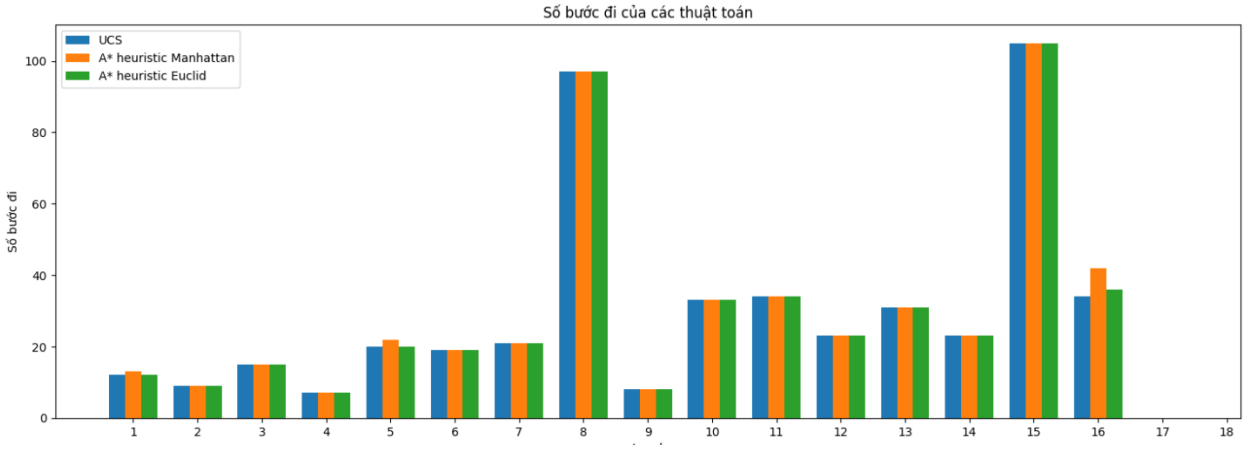
Hình 2: Thời gian tìm ra lời giải của các thuật toán qua các màn với trục thời gian nhỏ hơn

Dựa vào bảng kết quả và đồ thị mô hình hóa thời gian tìm ra lời giải của các thuật toán, ta có những nhận xét cơ bản sau :

- Ở các level đơn giản, cả ba thuật toán có thời gian tìm ra lời giải khá tương đồng, A* có xu hướng giải nhanh hơn so với UCS.

- Có một số level mà thuật toán UCS mất thời gian tương đối lâu để tìm ra lời giải, đặc biệt là ở level 5 và level 16. Trong khi đó A* Euclid và A* Manhattan tìm ra lời giải ở hai level nhanh hơn đáng kể, điều này cho thấy rằng không có ước lượng heuristic có thể làm UCS trở nên kém hiệu quả hơn so với A* tại các level có độ phức tạp cao hoặc không gian trạng thái lớn.
- A* Manhattan và A* Euclid có thời gian chạy khá tương đồng và ổn định qua các level, A* Manhattan có xu hướng nhỉnh hơn một chút.
- Ở level 18, cả ba thuật toán đều không tìm ra lời giải trong khoảng thời gian khá lớn (5 tiếng), điều này cho thấy đây là một level có độ phức tạp cao

- **Số bước đi**

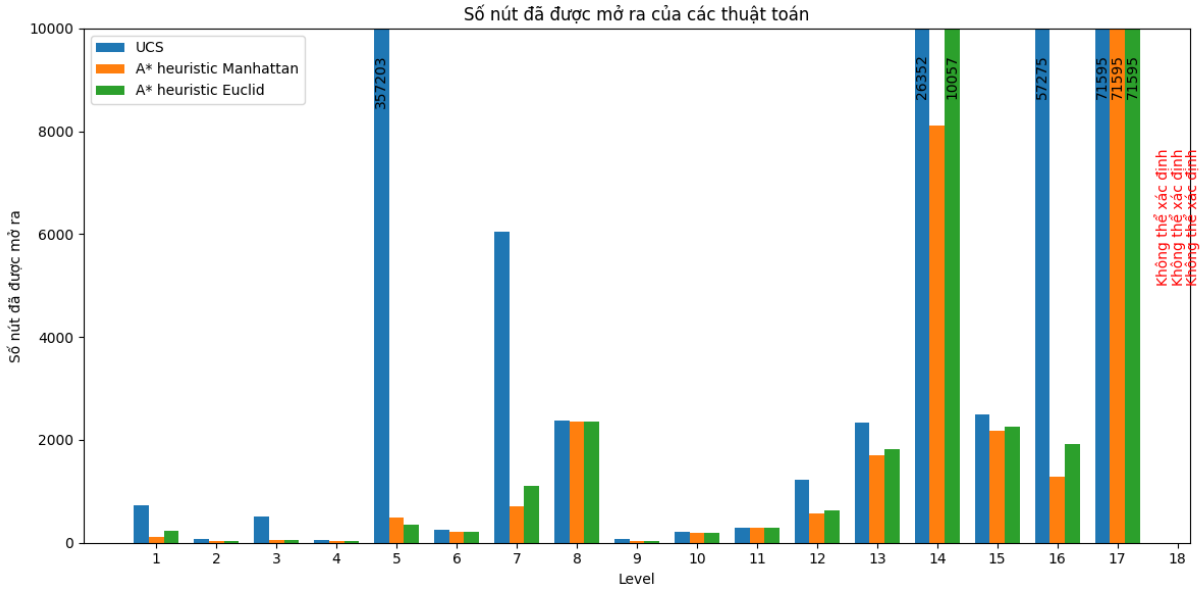


Hình 3: Số bước đi của lời giải các thuật toán qua các màn

Dựa vào bảng kết quả trực nghiệm và đồ thị mô hình hóa kết quả, ta có những nhận xét cơ bản sau :

- Đa số các thuật toán đều tìm ra lời giải với số bước tối ưu và tương đương nhau ở những level không quá phức tạp hoặc có lối giải rõ ràng mà mọi thuật toán đều có thể đạt tới một cách dễ dàng.
- UCS luôn tìm ra đường đi tối ưu.
- Lời giải của A* Manhattan ở level 1, 5, 16 không tối ưu, trong khi đó lời giải của A* Euclid chỉ không tối ưu ở level 17 (nhiều hơn 2 bước so với lời giải tối ưu).
- Level 17 thì cả 3 giải thuật đều trả về mảng rỗng do đây là bài toán không có lời giải.

- **Số nút đã được mở ra của các thuật toán**



Hình 4: Số nút mở ra của các thuật toán qua các màn

- UCS mở nhiều nút hơn so với hai phiên bản của A* trong hầu hết các level. Điều này do UCS không sử dụng heuristic để ước lượng khoảng cách đến trạng thái đích, nên nó không phân biệt được nút nào có khả năng dẫn đến mục tiêu hơn.
- A* với heuristic Manhattan và A* với heuristic Euclid mở ít nút hơn UCS rất nhiều từ level 1 đến level 16. Điều này cho thấy ảnh hưởng tích cực của việc sử dụng heuristic trong việc giảm số lượng nút cần phải xét, từ đó tối ưu quá trình tìm kiếm.
- Cả hai phiên bản của A* tương đối ổn định qua các level, với một số biến động nhưng không quá lớn, phản ánh việc cả hai heuristic cung cấp ước lượng có hiệu quả cho việc hạn chế không gian tìm kiếm.
- Màn 17 không có lời giải nên tất cả các thuật toán phải mở hết tất cả các node có thể có là 71595.

KẾT LUẬN

- Trong các màn chơi từ 1 đến 17, mặc dù thuật toán UCS luôn đảm bảo tìm ra lời giải tối ưu, nhưng quá trình tìm kiếm của nó lại tiêu tốn nhiều thời gian do phải xem xét một lượng lớn các node. Điều này khiến UCS hoạt động chậm so với A*.
- Nhờ vào việc sử dụng hàm heuristic, thuật toán A* có thể tìm ra lời giải mà không cần phải mở nhiều node như UCS. Điều này giúp giảm đáng kể thời gian cần thiết để đạt được lời giải, và trong hầu hết các trường hợp, lời giải vẫn đảm bảo tối ưu.

- Tuy nhiên, hiệu suất của thuật toán A* rất phụ thuộc vào việc thiết kế của hàm heuristic. Việc lựa chọn hoặc thiết kế một hàm heuristic phù hợp là rất quan trọng để có thể tạo ra sự cân bằng giữa việc đạt được lời giải tối ưu và rút ngắn thời gian tìm kiếm.