ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

BÀI TẬP MÔN TRÍ TUỆ NHÂN TẠO

BÀI TẬP 2 – Heuristics & A* search

GV hướng dẫn:
TS. Lương Ngọc Hoàng

Sinh viên thực hiện:
Trương Phúc Trường – 22521587

TP. HCM, ngày 20 tháng 3 năm 2024

1. Heuristics đã sử dụng trong file solver.py có ý tưởng là gì?

```python
def heuristic(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)
    sortposBox = list(set(posBox).difference(completes))
    sortposGoals = list(set(posGoals).difference(completes))
    for i in range(len(sortposBox)):
        distance += (abs(sortposBox[i][0] - sortposGoals[i][0])) + (abs(sortposBox[i][1] - sortposGoals[i][1]))
    return distance
```

– Hàm heuristic đã được định nghĩa trong đoạn mã trên sử dụng ý tưởng khoảng cách Manhattan để ước tính chi phí từ trạng thái hiện tại đến trạng thái đích trong bài toán Sokoban.

– Ý tưởng chính của hàm heuristic này là tính tổng khoảng cách Manhattan giữa các hộp chưa ở đúng vị trí đích với vị trí đích tương ứng. Khoảng cách Manhattan giữa hai điểm (x1, y1) và (x2, y2) là |x1 - x2| + |y1 - y2|, tức là tổng của giá trị tuyệt đối của hiệu giữa các tọa độ x và y.

Cụ thể, hàm heuristic thực hiện các bước sau:

– Tìm các hộp đã ở đúng vị trí đích bằng cách giao nhau giữa tập “posGoals” (vị trí các mục tiêu) và “posBox” (vị trí các hộp).

– Loại bỏ các hộp đã ở đúng vị trí khỏi tập “posBox” và “posGoals” để thu được hai tập mới “sortposBox” và “sortposGoals” chứa các hộp chưa ở đúng vị trí và vị trí đích tương ứng.

– Duyệt qua các cặp hộp và vị trí đích tương ứng, tính khoảng cách Manhattan giữa chúng và cộng dồn vào biến “distance”.

Hàm heuristic này đáp ứng tính chất của heuristic tốt cho thuật toán A*, đó là:

– Luôn nhỏ hơn hoặc bằng chi phí thực tế từ trạng thái hiện tại đến trạng thái đích (tính chất admissible).

– Có thể tính toán nhanh và dễ dàng để đánh giá các trạng thái.

2. Thiết kế và thực nghiệm so sánh thêm với heuristics mới

Không có

3. Bảng thống kê so sánh thời gian chạy (số giây) và số nút đã được mở ra của hai thuật toán UCS và A* cho mỗi level

Level	Thời gian chạy (giây)		Số nút đã được mở ra		Lời giải trả về của A* có tối ưu hay không?
	UCS	A* search	UCS	A* search	
1	0.091	0.013	720	86	Không
2	0.007	0.022	64	95	Không
3	0.134	0.010	509	60	Không
4	0.015	0.010	55	51	Không
5	99.440	0.188	357203	800	Có
6	0.024	0.040	250	294	Không
7	0.843	0.212	6046	1579	Không
8	0.373	0.426	2383	3272	Không
9	0.032	0.005	74	29	Không
10	0.024	0.048	218	252	Không
11	0.049	0.048	296	357	Không
12	0.144	0.122	1225	1016	Không
13	0.313	0.449	2342	3819	Không
14	4.666	2.614	26352	16920	Không
15	0.492	0.659	2505	3925	Không
16	23.356	1.017	57275	3221	Có
17	KXĐ	56.396*	KXĐ	123934*	KXĐ
18	KXĐ	KXĐ	KXĐ	KXĐ	KXĐ

*: Kết quả này được hiển thị ra màn hình nhưng không tìm được lời giải.

KXĐ: Không thể xác định được do thuật toán mất quá nhiều thời gian để giải cũng như còn phụ thuộc còn cấu hình máy khởi chạy thuật toán.

Về câu hỏi lời giải trả về của A có phải lời giải tối ưu hay không cho level tương ứng*, theo như quan sát và tính toán em nhận thấy số bước đi mà thuật toán A* cho ra gần như giống với số bước đi mà thuật toán UCS cho ra được (đã tìm được ở BT1) nên hầu hết các màn chơi thuật toán A* không nhỉnh hơn được UCS. Nhưng ở màn chơi 5 và 16, ta thấy được sự tối ưu của A* so với UCS, cụ thể ở cả 2 màn chơi, thuật toán A* tìm được lời giải trong khoảng thời gian rất ngắn so với UCS.