ĐẠI HỌC QUỐC GIA TPHCM

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA KHOA HỌC MÁY TÍNH

TRÍ TUỆ NHÂN TẠO

Heuristics & A* search

BT2_23520127

Môn học: CS106.P21.KHTN

Sinh viên thực hiện:

Nguyễn Thiên Bảo - 23520127

Giáo viên hướng dẫn:

TS.Lương Ngọc Hoàng

Ngày 20 tháng 3 năm 2025

Mục lục

1 Tổng quan về bài tập Sokoban

Ở bài tập này ta sẽ cài đặt và so sánh hiệu suất thuật toán UCS với thuật toán A* với hai hàm Heuristics.

1.1 Mô hình hóa

Chi tiết cách mô hình hóa Sokoban(tham số hóa các đối tượng, không gian trạng thái, trạng thái khởi đầu, trạng thái kết thúc, tập các hành động hợp lệ, hàm tiến triển,..) đã được phân tích chi tiết ở BT1_23520127

2 Thuật toán A*

Thuật toán tìm đường đi A* sử dụng hàm Heuristic để đánh giá chi phí từ trạng thái hiện tại đến trạng thái mục tiêu, khả thi hơn nếu không gian trạng thái lớn bởi vì số đỉnh được mở rộng giảm đáng kể, thời gian thực thi nhờ đó cũng giảm. Vì vậy, nếu có được một hàm Heuristic đủ tốt thì thuật toán A* sẽ trở nên ưu việt.

2.1 hàm heuristic có sẵn

Ý tưởng:

- Hàm này tính khoảng cách giữa những thùng *chưa* được đưa vào goal với những ô goal còn trống.
- Đầu tiên, hàm xác định những thùng nào *đã* nằm đúng trên goal (hoàn thành) rồi bỏ qua chúng.
- Sau đó, với các thùng còn lại, hàm ghép trực tiếp (theo thứ tự) từng thùng với một goal tương ứng (cũng theo cùng thứ tự) rồi cộng tất cả khoảng cách Manhattan.

- Khoảng cách Manhattan giữa hai vị trí (x_1, y_1) và (x_2, y_2) được tính bởi:

$$\text{ManhattanDistance} = |x_1 - x_2| + |y_1 - y_2|.$$

Công thức tổng quát trong sokoban:

1. Bỏ qua các thùng đã nằm ở goal.
2. Xác định danh sách thùng chưa hoàn thành và goal còn trống, duyệt đồng bộ theo thứ tự.
3. Với mỗi cặp (thùng, goal), cộng khoảng cách Manhattan vào biến `distance`.
4. Kết quả trả về là tổng `distance`.

Tóm gọn:

$$\text{heuristic} = \sum_i \Big(\big|\, \text{box}_i.x - \text{goal}_i.x \,\big| \;+\; \big|\, \text{box}_i.y - \text{goal}_i.y \,\big| \Big).$$

2.2 Bảng thống kê độ dài đường đi

Map	steps-runtime-nodes UCS			steps-runtime-nodes A*		
	Steps	Time (s)	Nodes	Steps	Time (s)	Nodes
Map 1	12	0.0411	720	13	0.0060	122
Map 2	9	0.0030	64	9	0.0020	39
Map 3	15	0.0642	509	15	0.0050	54
Map 4	7	0.0020	55	7	0.0010	29
Map 5	20	58.0782	357203	22	0.0645	485
Map 6	19	0.0098	250	19	0.0087	208
Map 7	21	0.4255	6046	21	0.0512	715
Map 8	97	0.1795	2383	97	0.1763	2352
Map 9	8	0.0097	74	8	0.0026	42
Map 10	33	0.0135	218	33	0.0114	198

Map 11	34	0.0124	296	34	0.0196	284
Map 12	23	0.0755	1225	23	0.0346	563
Map 13	31	0.1465	2342	31	0.1116	1699
Map 14	23	2.3795	26352	23	0.7561	8108
Map 15	105	0.2352	2505	105	0.2163	2183
Map 16	34	13.7783	57275	42	0.2384	1286
Map 17	0	20.1855	29412	0	20.6422	71595
Map 18	treo	treo	treo	treo	treo	treo

2.3 Nhận xét

Kết quả đường đi của A* thường giống UCS, nhưng trên bản đồ 1, 5 và 16, A* cho ra kết quả chưa tối ưu (thua UCS). Điều này có thể do hàm Heuristic không hiệu quả; khoảng cách Manhattan, mặc dù chính xác khi không có vật cản, lại trở thành yếu điểm khi có vật cản, khiến khoảng cách thực tế lớn hơn nhiều.

Mặt khác, A* tối ưu hơn UCS về thời gian và không gian (số nút được mở ra). Tuy nhiên, với các màn chơi không có lời giải, A* cũng tốn nhiều thời gian mới đưa ra được là màn đó không có lời giải.

2.3.1 Vì sao `heuristic` không đảm bảo tối ưu?

- **Không tính đến cấu trúc bản đồ phức tạp:** Sokoban có thể cần di chuyển thùng vòng ra nhiều hướng, tránh tường, tránh góc chết. Trong khi đó, `heuristic` này chỉ cộng khoảng cách Manhattan thùng–goal theo "trực tiếp", không xét chướng ngại thực tế.
- **Chưa đảm bảo tính *admissible*:** Muốn A* tối ưu, heuristic phải không được "dư" so với chi phí thực. Trong Sokoban, việc chỉ cộng khoảng cách Manhattan đơn thuần có thể nhiều khi đánh giá quá thấp *(hoặc quá cao)* so với thực tế.

3 Hàm Heuristic mới

3.1 Hàm `heuristicpro(posPlayer, posBox)`

Ý tưởng:

- Hàm này là một phiên bản nâng cấp và "thông minh" hơn hàm trên, thường cho giá trị heuristic lớn hơn (nhưng sát với thực tế hơn).
- Vẫn bỏ qua các thùng đã nằm đúng vị trí goal.
- Ghép tham lam (greedy): Đối với mỗi thùng chưa nằm đúng, tìm ô goal trống gần nhất (theo khoảng cách Manhattan) rồi ghép và loại goal đó khỏi danh sách.
- Cộng thêm khoảng cách từ người chơi đến thùng gần nhất (các thùng chưa vào goal). Đây là cải tiến so với hàm cũ.

Công thức tổng quát:

1. Gọi unplaced_boxes là tập thùng *chưa* lên goal, empty_goals là tập goal còn trống.
2. Tính
$$\text{player\_dist} = \min_{b \in \text{unplaced\_boxes}} \big(|\, \text{posPlayer}.x - b.x \,| + |\, \text{posPlayer}.y - b.y \,| \big).$$
3. Với mỗi thùng trong unplaced_boxes, tìm *goal trống gần nhất* trong empty_goals, cộng dồn khoảng cách Manhattan vào box_goal_dist.
$$\text{box\_goal\_dist} = \sum_{b \in \text{unplaced\_boxes}} \min_{\substack{g \in \text{temp\_goals} \\ \text{(mỗi goal chỉ dùng một lần)}}} \Big(\big|b.x - g.x\big| + \big|b.y - g.y\big| \Big).$$
4. Tổng heuristic:
$$\text{distance} = \text{player\_dist} + \text{box\_goal\_dist}.$$

Lưu ý: ta có thể tinh chỉnh tham số tùy vào trường hợp (ví dụ 1,5* player_dist và 0.5 * box_goal_dist) để cho ra hàm Heuristic tối ưu cho mỗi màn chơi.

3.2 Bảng thống kê độ dài đường đi giữa hai hàm Heuristics

Map	steps-runtime-nodes A* mới			steps-runtime-nodes A* cũ		
	Steps	Time (s)	Nodes	Steps	Time (s)	Nodes
Map 1	12	0.0050	52	13	0.0060	122
Map 2	9	0.0021	33	9	0.0020	39
Map 3	15	0.0085	61	15	0.0050	54
Map 4	7	0.0010	17	7	0.0010	29
Map 5	20	0.0595	392	22	0.0645	485
Map 6	19	0.0077	164	19	0.0087	208
Map 7	21	0.0531	611	21	0.0512	715
Map 8	97	0.1878	2154	97	0.1763	2352
Map 9	8	0.0020	23	8	0.0026	42
Map 10	33	0.0131	193	33	0.0114	198
Map 11	34	0.0150	280	34	0.0196	284
Map 12	23	0.0398	431	23	0.0346	563
Map 13	31	0.1157	1468	31	0.1116	1699
Map 14	23	1.1302	10000	23	0.7561	8108
Map 15	105	0.2382	2090	105	0.2163	2183
Map 16	40	2.6814	10798	42	0.2384	1286
Map 17	0	22.7363	71595	0	20.6422	71595
Map 18	treo	treo	treo	treo	treo	treo

3.3 Nhận xét

Nhìn chung, hàm heuristic mới tốt hơn hàm cũ về đường đi ở mọi bản đồ đang xét, nó đã cho ra kết quả tối ưu ở map 1, 5 (map 16 tuy nhỏ hơn hàm cũ nhưng vẫn chưa cho kết quả tối ưu).

3.3.1 Vì sao `heuristicpro` vẫn không đảm bảo tối ưu?

- **Vẫn bỏ qua môi trường cản trở:** Tường, góc hẹp, hoặc có nhiều thùng cản đường nhau. Heuristic "tham lam" box–goal này không thể lường hết việc phải "giải phóng" lối đi.
- **Không đảm bảo *admissible*:** Để A* cho kết quả tối ưu, cần điều kiện này. Việc "cộng" quãng đường người chơi + ghép tham lam vẫn có thể thừa hoặc thiếu so với thực tế tuỳ từng tình huống, nên không chắc chắn duy trì tính admissible.