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

KHOA: KHOA HỌC MÁY TÍNH

MÔN HỌC: CS106.O21.KHTN - TRÍ TUỆ NHÂN TẠO

Sokoban Solver ver. 2

Sinh viên:
Nguyễn Tuấn Anh – 21520142

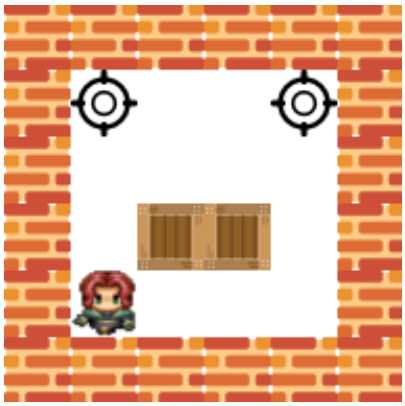
Giảng viên:
TS. Lương Ngọc Hoàng

Mục lục

Danh sách hình vẽ

1. Tổng quan về Sokoban

1.1. Quy tắc



Hình 1: Màn chơi Sokoban cơ bản.

Mục tiêu của trò chơi Sokoban là người chơi sẽ phải di chuyển nhân vật để đưa các hộp vào các vị trí đích trên bản đồ. Ở mỗi bước di chuyển, có thể di chuyển nhân vật qua trái, phải, lên, xuống nếu không vượt qua khỏi bản đồ và các chướng ngại vật. Minh họa như hình 1, người chơi sẽ tìm cách đưa hai hộp được đặt ở giữa đến hai vị trí ở mép bản đồ. Các bước hợp lệ để hoàn thành màn chơi trên là: `phải, lên, phải, xuống, phải, lên, lên, trái, trái, xuống, trái, lên`. Một chuỗi các bước di chuyển tối ưu là chuỗi các bước di chuyển hợp lệ và tìm ra được lời giải, đồng thời phải có độ dài nhỏ nhất.

Proposition

Một bản đồ nếu tồn tại lời giải thì sẽ tồn tại chuỗi các bước di chuyển tối ưu.

1.2. Mô hình hóa

Đầu tiên, tham số hóa tọa độ các đối tượng của trò chơi:

- Nhân vật: posPlayer(x, y).
- Tập các hộp: posBoxes = posBox(x_i, y_i).
- Tập các vị trí đích: posGoals = posGoal(x_i, y_i).
- Biên bản đồ: posWalls = posWall(x_j, y_j).

Sau đó, chúng ta mô hình hóa trò chơi thành bài toán tìm đường đi (cách di chuyển các hộp đến các vị trí đích) như sau:

- Trạng thái khởi đầu (start state): (posPlayer, posBoxes). Tọa độ nhân vật và các hộp ban đầu.
- Trạng thái kết thúc (goal state): posBoxes $\equiv$ posGoals. Tất cả các hộp được di chuyển đến đích.
- Không gian trạng thái (state space): $(\text{posPlayer}_T, \text{posBoxes}_T)$. Tập các vị trí nhân vật và các hộp.
- Hành động (action): Up (U, u), Down (D, d), Left (L, l), Right (R, r). Nhân vật được phép di chuyển lên, xuống, trái, phải.
- Hành động hợp lệ (legal action): posPlayer $\notin$ posWalls, posBoxes $\notin$ posWalls, newPosBox $\notin$ posBoxes. Nhân vật không được phép vượt qua biên, các hộp không được đẩy vượt qua biên, chỉ được đẩy một hộp ở mỗi thời điểm.
- Hàm tiến triển (successor function): $\forall$ legal action, update (posPlayer, PosBoxes). Cập nhật vị trí của nhân vật và các hộp nếu hành động là hợp lệ.

Lemma 1.1

Cho M, N lần lượt là số ô không có chướng ngại vật và số lượng hộp. Kích thước không gian trạng thái là

$$|\text{posPlayer}|.|\text{posBoxes}| = M.\binom{M-1}{N}$$

Remark 1.2. Bản đồ có số ô không có chướng ngại vật càng nhiều thì kích thước không gian trạng thái càng lớn. Hơn nữa, nếu bản đồ càng "thoáng" thì một thuật toán tìm đường đi sẽ gặp nhiều khó khăn khi không "cắt tỉa" được những trạng thái không có tiềm năng, dẫn đến gần như phải xét toàn bộ không gian trạng thái.

2. Hàm Heuristic

Thuật toán A* sử dụng hàm Heuristic để "ước lượng" chi phí từ trạng thái hiện tại đến đích. Vì vậy, nếu có được một hàm Heuristic đủ tốt thì thuật toán A* sẽ trở nên ưu việt. Khi thiết kế hàm Heuristic cho Sokoban, một ý tưởng đơn giản là không xét trạng thái trung gian, các chi phí chỉ tính từ vị trí các hộp hiện tái và đích đến. Nghĩa là, chúng ta đang đề cập đến việc tính khoảng cách Manhattan

$$\text{Heuristic} = \sum_{i=1}^{N} Manhattan(\text{posBox}_i, \text{posGoals}_i)$$

Rõ ràng, hàm Heuristic này thỏa mãn tính Admissibility bởi vì đường đi thật sự sẽ không bé hơn khoảng cách được ước tính do có thể có vật chắn trung gian. Tuy nhiên, tính Consistency không được đảm bảo do sự xuất hiện tùy biến của chướng ngại vật (chỉ thỏa mãn với các bản đồ có thể thuận lợi đẩy hộp một cách liên tục về phía đích). Do đó, hàm Heuristic này sẽ không đảm bảo có thể tìm ra được lời giải tối ưu.

3. Thống kê thực nghiệm

Bảng 1: Kết quả thực nghiệm 2 thuật toán tìm kiếm Uniform Cost Search và A* Search trên 18 bản đồ trò chơi sau khi đã mô hình hóa. Ký hiệu "-" khi thuật toán không thành công (Memory Error, Overflow), "×" khi bản đồ không có lời giải. Với mỗi thuật toán, các giá trị như Length (độ dài chuỗi bước đi), Time (thời gian giải bài toán) và #Nodes (số đỉnh được mở rộng) được thể hiện như bên dưới. https://github.com/nAuTahn

	Uniform Cost Search			A* Search		
Map	Length	Time (ms)	#Nodes	Length	Time (ms)	#Nodes
1	12	49.56	2115	13	5.99	265
2	9	3.99	183	9	3.53	112
3	15	77.31	1465	15	8.01	166
4	7	3.01	174	7	2.51	97
5	20	66394.26	1355329	22	71.71	1898
6	19	12.68	612	19	12.51	530
7	21	500.01	17283	21	71.13	2308
8	97	195.67	5357	97	191.39	5305
9	8	6.56	181	8	2.01	95
10	33	13.54	526	33	13.57	479
11	34	14.54	668	34	14.02	640
12	23	80.32	3220	23	39.54	1553
13	31	167.16	6202	31	125.04	4574
14	23	2889.30	72422	23	801.79	22806
15	105	265.19	6328	105	239.87	5526
16	34	15374.23	150971	42	253.19	3300
17	×	×	×	×	×	×
18	-	-	-	-	-	-

Remark 3.1. Uniform Cost Search (UCS) và A* Search (A*) đều cho ra kết quả trên 17 bản đồ. Tuy nhiên có sự khác biệt lớn về thời gian thực thi và số lượng node được chọn để mở rộng. Mặc dù hàm Heuristic khá tốt nhưng vẫn không giúp A* có thể giải được bản đồ 18. Thuật toán UCS, như đã biết, sẽ dựa hoàn toàn vào chi phí đã biết từ đỉnh gốc đến đỉnh hiện tại mà không quan tâm đến mục tiêu. Điều này sẽ kém hiệu quả khi không gian trạng thái lớn, có thể lấy ví dụ ở bản đồ 5 và bản đồ 16. Thuật toán A* sử dụng thêm hàm Heuristic để "ước lượng" chi phí từ đỉnh hiện tại đến đích, 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.

Remark 3.2. Lời giải trả về của A* đa phần giống UCS, nhưng ở bản đồ 1 và bản đồ 16 thì A* lại cho ra kết quả chưa tối ưu. Điều này có thể do hàm Heuristic chưa đủ tốt. Nghĩa là, mặc dù khoảng cách Manhattan cung cấp một chặn dưới khá chính xác (nếu không có vật cản), nhưng đó cũng lại là điểm yếu nếu như có vật cản (khoảng cách thực tế sẽ lớn hơn nhiều).

4. Đề xuất hàm Heuristic mới

Như đã đề cập ở Remark 3.2, hàm Heuristic được sử dụng trước đó còn quá "lỏng lẻo", vậy nên ý tưởng tự nhiên đó là tìm cách tăng chặn dưới của hàm một lượng vừa phải. Ở đây, khoảng cách giữa nhân vật và các hộp được sử dụng làm phần thêm vào. Cụ thể

$$\text{Heuristic} = \sum_{i=1}^{N} Manhattan(\text{posPlayer}, \text{posBox}_i) + Manhattan(\text{posBox}_i, \text{posGoals}_i)$$

Bảng 2: Kết quả thực nghiệm thuật toán A* Search trên hàm Heuristic mới.

	A* Search (new Heuristic)		
Map	Length	Time (ms)	#Nodes
1	13	3.65	75
2	9	3.68	52
3	15	4.06	66
4	7	2.06	56
5	22	85.87	1289
6	19	19.26	415
7	21	12.19	217
8	97	295.02	3966
9	8	6.54	108
10	33	37.66	441
11	34	29.93	608
12	23	32.86	681
13	33	140.77	2789
14	23	260.39	4139
15	107	484.57	4838
16	34	264.99	1214
17	×	×	×
18	-	-	-

Remark 4.1. Hàm Heuristic mới được sử dụng đã làm giảm đáng kể số lượng đỉnh mở rộng, đồng thời lời giải trả về cũng có thể được xem là tối ưu (trừ bản đồ 13 và 15). Tuy nhiên thời gian thực thi cũng tăng nhưng không nhiều.