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

Sinh viên:
Nguyễn Tuấn Anh – 21520142

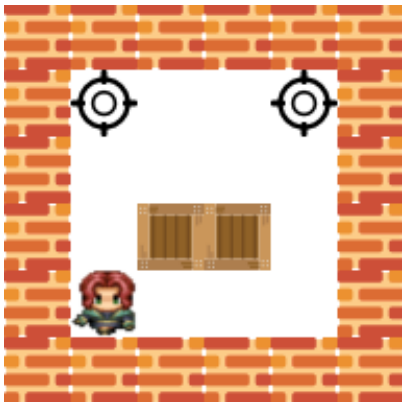
Giảng viên:
TS. Lương Ngọc Hoàng

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. Thống kê thực nghiệm

Bảng 1: Kết quả thực nghiệm 3 thuật toán tìm kiếm Depth-First Searc, Breadth-First Search và Uniform Cost 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. Hiệu suất thuật toán được đo bằng độ dài đường đi (số lượng bước di chuyển nhân vật để các hộp được đặt vào tất cả vị trí đích). https://github.com/nAuTahn

Map	Depth-First Search	Breadth-First Search	Uniform Cost Search
1	79	12	12
2	24	9	9
3	403	15	15
4	27	7	7
5	-	20	20
6	55	19	19
7	707	21	21
8	323	97	97
9	74	8	8
10	37	33	33
11	36	34	34
12	109	23	23
13	185	31	31
14	865	23	23
15	291	105	105
16	-	34	34
17	×	×	×
18	×	×	×

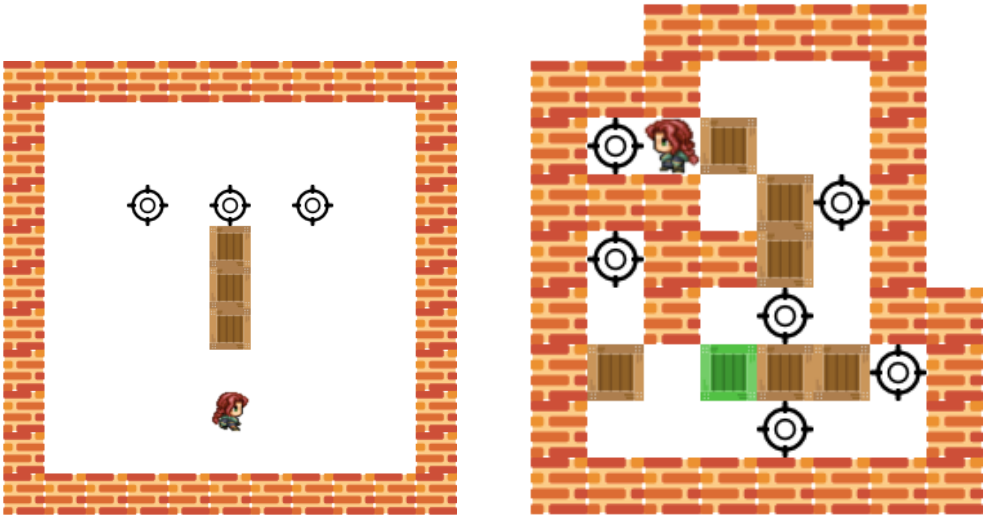
Remark 2.1. Cả 3 thuật toán đều cho ra lời giải, đặc biệt lời giải được tìm ra bởi Breadth-First Search và Uniform Cost Search giống nhau ở 16 bản đồ. Bản đồ 17 và 18 không có lời giải.

Remark 2.2. Depth-First Search không thành công ở bản đồ 5 và 16. Khả năng tìm được lời giải của DFS là có, nhưng đa phần chưa tối ưu và thời gian đánh đổi quá lớn do thuật toán tập trung khai thác một nhánh trạng thái.

Remark 2.3. Breadth-First Search và Uniform Cost Search đều cho ra lời giải tối ưu. Tuy nhiên có sự khác biệt về thời gian do UCS sử dụng Priority Queue được triển khai bằng Heap, chậm hơn Deque. Ngoài ra, hàm cost của UCS cũng ảnh hưởng phần nào. Cơ chế của UCS, với các bản đồ khó cần nhiều bước di chuyển để đi đến lời giải tối ưu, thì thường sẽ có nhiều khả năng tìm được lời giải tối ưu trước BFS. Điều này do BFS phải duyệt tất cả trạng thái trong một độ sâu. Thực nghiệm trên 18 bản đồ trên, vẫn có một số bản đồ mà UCS chạy chậm hơn BFS. Với sự tùy biến của các bản đồ trong tự nhiên, thì việc một bản đồ có lời giải tối ưu thỏa

mãn tính chất được nói đến trước đó là rất cao. Trong những trường hợp như vậy, UCS sẽ tốt hơn cả so với DFS và BFS.

Remark 2.4. Cả ba thuật toán đều gặp khó khăn ở bản đồ 5. Đây cũng là bản đồ khó giải nhất. Bởi vì thỏa mãn nội dung của Remark 1.2 và Lemma 1.1, nghĩa là các thuật toán khi gặp bản đồ 5 đều gần như phải xét toàn bộ không gian trạng thái rất lớn. Ngoài bản đồ 5, thì bản đồ 16 cũng khá khó giải. Tuy nhiên, mặc dù không gian trạng thái lớn, nhưng bản đồ 16 không thỏa mãn nội dung Remark 1.2 nên thời gian chạy không quá lớn so với bản đồ 5.



(a) Bản đồ 5 (b) Bản đồ 16

Hình 2: Hai bản đồ khó giải.