Đại học Quốc gia TP.HCM

Trường Đại học Công nghệ Thông tin

Báo cáo về việc sử dụng thuật toán tìm đường đi A* cho trò chơi Sokoban

Nguyễn Duy Hoàng
MSSV: 22520467
Lớp CS106.O21
Giảng viên môn học: Lương Ngọc Hoàng

Mục lục

1 Ý tưởng đằng sau heuristic

Trong lĩnh vực trò chơi và trí tuệ nhân tạo, hàm heuristic được định nghĩa là một phần của thuật toán tìm kiếm thông tin, đánh giá chi phí xấp xỉ từ trạng thái ban đầu đến trạng thái mục tiêu. Các hàm heuristic có thể thay đổi tùy thuộc vào cách thiết lập của trò chơi, và thường có nhiều hàm heuristic khác nhau được sử dụng. Trong trò chơi giải đố Sokoban, em đã sử dụng hai hàm heuristic khác nhau: (1.1) khoảng cách Manhattan và (1.2) khoảng cách Euclidean.

1.1 Khoảng cách Manhattan (*Được cài đặt mẫu*)

Ý Tưởng

Heuristics Manhattan được đặt theo tên của đường phố Manhattan, nơi mà các con đường di chuyển vuông góc với nhau, tạo thành một lưới.

Công Thức Tính Khoảng Cách

Hàm khoảng cách Manhattan tính toán khoảng cách sẽ được đi để đi từ một điểm dữ liệu này đến điểm dữ liệu khác nếu theo một con đường theo dạng lưới (*grid*). Khoảng cách Manhattan giữa hai điểm là tổng của sự chênh lệch giữa các thành phần tương ứng của chúng, có thể được biểu diễn bằng công thức sau:

$$d = |x_1 - x_2| + |y_1 - y_2| \text{ , trong đó } (x_1, y_1) \text{ và } (x_2, y_2) \text{ là tọa độ của hai điểm X và Y.}$$

1.2 Khoảng cách Euclidean (*Hàm heuristic mới*)

Ý Tưởng

Heuristics Euclidean đo lường khoảng cách "đường chim bay".

Công Thức Tính Khoảng Cách

Hàm khoảng cách Euclidean tính toán khoảng cách dựa theo định lý Pythagoras trong hình học, trong đó khoảng cách giữa hai điểm được tính bằng độ dài của đường thẳng nối chúng. Khoảng cách Euclidean giữa hai điểm là căn bậc hai của tổng bình phương của sự chênh lệch giữa các thành phần tương ứng của chúng, có thể được biểu diễn bằng công thức sau:

$$d = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2} \text{ , trong đó } (x_1, y_1) \text{ và } (x_2, y_2) \text{ là tọa độ của hai điểm X và Y.}$$

Dưới đây là cách cài đặt hàm *Euclidean*:

```
import math

def euclidean(posPlayer, posBox):
    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 += math.sqrt((sortposBox[i][0] - sortposGoals[i][0])**2
            + (sortposBox[i][1] - sortposGoals[i][1])**2)
    return distance
```

1.3 Khoảng cách Chebyshev (*Hàm heuristic mới*)

Ý Tưởng

Heuristics Chebyshev đo lường khoảng cách giữa hai điểm dựa trên "metric tối đa", đo lường sự khác biệt lớn nhất giữa các tọa độ trên mỗi trục của chúng.

Công Thức Tính Khoảng Cách

Hàm khoảng cách Chebyshev tính toán khoảng cách giữa hai điểm bằng cách sử dụng khoảng cách tuyệt đối giữa các thành phần tương ứng của chúng. Khoảng cách Chebyshev giữa hai điểm có thể được biểu diễn bằng công thức:

$$d = \max(|x_1 - y_1|, |x_2 - y_2|)$$, trong đó (x_1, y_1) và (x_2, y_2) là tọa độ của hai điểm X và Y.

Dưới đây là cách cài đặt hàm *Chebysev*:

```
def chebysev(posPlayer, posBox):
    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 += max(abs(sortposBox[i][0] - sortposGoals[i][0]), abs(
            sortposBox[i][1] - sortposGoals[i][1]))
    return distance
```

1.4 So sánh 3 hàm heuristic khi được áp dụng trong thuật toán A*

Về mặt lý thuyết, em nghĩ thuật toán Manhattan trong bài toán Sokoban thường cho hiệu suất tốt hơn so với Euclidean và Chebysev. Điều này là do bài toán Sokoban thường có các vật cản và hướng di chuyển bị hạn chế, khiến cho các đường đi giữa các trạng thái không gian thường di chuyển theo các hướng ngang và dọc, phù hợp với cách ước lượng khoảng cách của Manhattan.

Để đánh giá hiệu suất của các hàm heuristic Euclidean, Manhattan và Chebysev khi được áp dụng trong thuật toán A*, em đã thu thập số liệu về số bước di chuyển và thời gian tìm kiếm trên từng cấp độ của trò chơi.

Level	Manhattan	Euclidean	Chebysev
1	13	13	12
2	9	9	9
3	15	15	15
4	7	7	7
5	22	22	20
6	19	19	19
7	21	21	21
8	97	97	97
9	8	8	8
10	33	33	33
11	34	34	34
12	23	23	23
13	31	31	31
14	23	23	23
15	105	105	105
16	42	36	34
17	NS	NS	NS
18	DNF	DNF	DNF

Bảng 1: Thống kê số bước di chuyển

Level	Manhattan	Euclidean	Chebysev
1	0.006	0.019	0.021
2	0.003	0.003	0.003
3	0.007	0.006	0.006
4	0.002	0.001	0.002
5	0.087	0.085	0.065
6	0.011	0.011	0.013
7	0.074	0.104	0.184
8	0.264	0.256	0.273
9	0.003	0.003	0.003
10	0.014	0.014	0.015
11	0.017	0.018	0.018
12	0.046	0.054	0.071
13	0.166	0.162	0.186
14	0.996	1.186	1.672
15	0.308	0.309	0.340
16	0.301	0.427	0.363
17	NS	NS	NS
18	DNF	DNF	DNF

Bảng 2: Thống kê thời gian tìm ra đường đi

Chú thích: NS - Không có lời giải, DNF - Chưa hoàn thành vì thời gian chạy quá lâu

Về mặt thực nghiệm trên các màn chơi từ màn chơi 1 đến màn chơi 16 trong trò chơi *Sokoban*, em đã quan sát và thu thập dữ liệu về hiệu suất của ba thuật toán Euclidean, Manhattan và Chebysev. Kết quả cho thấy rằng, trên hầu hết các màn chơi, ba thuật toán này có hiệu suất gần như tương đương nhau, với số bước di chuyển và thời gian tìm kiếm không có sự chênh lệch đáng kể.

Tuy nhiên, có một vài trường hợp đặc biệt: tại màn chơi 16, trong đó thuật toán Euclidean đã có hiệu suất tốt hơn so với Manhattan; tại các màn chơi 1, 5 và 16 thì thuật toán Chebysev có hiệu suất tốt hơn hẳn so với hai thuật toán còn lại. Cụ thể, trong màn chơi 1, số bước di chuyển của thuật toán Euclidean và Manhattan là 13, trong khi đó thuật toán Chebysev là 12; trong màn chơi 5, số bước di chuyển của thuật toán Euclidean và Manhattan là 22, trong khi đó thuật toán Chebysev là 20; trong màn chơi 16, khi dùng thuật toán Chebysev thì nhân vật Sokoban sẽ di chuyển 34 bước, ngắn hơn so với 36 của Euclidean và 42 của Manhattan. Điều này cho thấy rằng, trong một số trường hợp cụ thể, việc sử dụng thuật toán Chebysev có thể đem lại kết quả tốt hơn so với Manhattan và Euclidean.

2 Nhận xét về 2 thuật toán tìm đường: A* và UCS

2.1 Tổng quan về 2 thuật toán và so sánh dựa trên thực nghiệm

Uniform-Cost Search (UCS) là một thuật toán tìm kiếm trên đồ thị không sử dụng bất kỳ kiến thức cụ thể về vấn đề. Nó mở rộng nút có chi phí thấp nhất và thực hiện điều này theo mọi hướng vì không có thông tin về mục tiêu được cung cấp. Có thể coi UCS như một hàm $f(n) = g(n)$, trong đó $g(n)$ là chi phí đường đi ("path cost"), tức là một hàm gán một chi phí số cho một đường đi dựa trên tiêu chí hiệu suất, ví dụ như khoảng cách tính bằng kilômét hoặc số lần di chuyển. Đây đơn giản là chi phí để đến nút n.

Best-First Search (Greedy search) là một thuật toán tìm kiếm thông tin, trong đó hàm chi phí tổng cộng $f(n)$ của mỗi nút n được xác định bởi $h(n)$, là giá trị của hàm heuristic ước lượng chi phí từ nút n đến mục tiêu. Best First Search chọn các hành động tiếp theo dựa trên ước lượng này mà không xem xét chi phí thực sự để đến được nút đó.

A* Search là sự kết hợp của UCS và Best-First Search. Trong A*, hàm chi phí tổng cộng $f(n)$ của mỗi nút n được xác định bởi $f(n) = g(n) + h(n)$, tức là tổng của chi phí đến nút n từ nút gốc và ước lượng chi phí từ nút n đến mục tiêu. A* chọn các hành động tiếp theo dựa trên sự kết hợp này, giúp tối ưu hóa việc tìm kiếm đường đi.

Dưới đây là bảng thực nghiệm giữa UCS và A trong các màn chơi của trò chơi Sokoban:*

Level	UCS	A*(chebysev)
1	12	12
2	9	9
3	15	15
4	7	7
5	20	20
6	19	19
7	21	21
8	97	97
9	8	8
10	33	33
11	34	34
12	23	23
13	31	31
14	23	23
15	105	105
16	34	34
17	NG	NG
18	DNF	DNF

Bảng 3: Thống kê số bước di chuyển (*steps*)

Level	UCS	A*(chebysev)
1	0.057	0.021
2	0.005	0.003
3	0.104	0.006
4	0.005	0.002
5	144.9	0.065
6	0.012	0.013
7	0.636	0.184
8	0.243	0.273
9	0.010	0.003
10	0.015	0.015
11	0.016	0.018
12	0.093	0.071
13	0.191	0.186
14	3.152	1.672
15	0.323	0.340
16	21.24	0.363
17	NG	NG
18	DNF	DNF

Bảng 4: Thống kê thời gian tìm ra đường đi (*seconds*)

Level	UCS	A*(chebysev)
1	818	261
2	82	39
3	677	54
4	89	35
5	687417	386
6	250	231
7	7110	1835
8	2425	2432
9	105	34
10	230	204
11	300	288
12	1266	769
13	2382	1997
14	27440	12948
15	2519	2360
16	70366	1533
17	NG	NG
18	DNF	DNF

Bảng 5: Thống kê số nút được mở ra (*Expanded nodes*)

Chú thích: NG - Không có lời giải, DNF - Chưa hoàn thành vì thời gian chạy quá lâu

Qua 3 bảng thực nghiệm trên: *Bảng 3, Bảng 4, Bảng 5*: có thể thấy số bước đi gần như tương đương nhau, kể cả trên các màn chơi khó hay dễ. Do đó, chúng ta sẽ so sánh về thời gian và số nút được mở ra.

Điểm khác biệt chính giữa UCS (*Uniform Cost Search*) và thuật toán A* đó là trong việc lựa chọn nút tiếp theo để mở rộng. Dưới đây là một số điểm khác biệt quan trọng:

Uniform Cost Search (UCS):

- UCS chọn nút tiếp theo dựa trên chi phí để đến nút đó, mà không quan tâm đến mục tiêu cuối cùng.
- Điều này có nghĩa là UCS có thể mở rất nhiều nút trong exploratory set, mở các nút ở xa mục tiêu trước khi đến được nút mục tiêu.

Thuật toán A* (A-star):

- A* sử dụng một hàm heuristic (heuristic function) để ước lượng chi phí từ nút hiện tại đến mục tiêu.
- Khi chọn nút tiếp theo, A* kết hợp chi phí thực tế từ nút bắt đầu đến nút hiện tại với ước lượng từ nút hiện tại đến mục tiêu thông qua hàm heuristic.
- Nhờ vào heuristic, A* thường có thể tránh mở rất nhiều nút không cần thiết và tập trung vào các nút tiềm năng gần mục tiêu.

Vì UCS không sử dụng hàm heuristic, nó có thể mở rất nhiều nút trong exploratory set, kể cả những nút ở xa mục tiêu. Điều này dẫn đến việc UCS thường mất thời gian hơn và duyệt qua số lượng nút lớn hơn so với A*. Điều này đặc biệt đáng chú ý trên các bản đồ khó, nơi A* có thể hiệu quả hơn nhiều so với UCS. Tuy nhiên, trên các bản đồ cực kì đơn giản (với số bước cực kì ít) thì sự khác biệt giữa thời gian chạy của UCS và A* có thể không lớn và có thể nói hiệu suất khi dùng hai thuật toán là tương đương nhau.

2.2 Nhận xét về lời giải của thuật toán A*

Thuật toán BFS (Breadth-First Search) được biết đến là tìm ra đường đi với số bước đi ít nhất từ nút ban đầu đến mục tiêu. Bằng cách so sánh số bước đi của lời giải từ A* với BFS, chúng ta có thể rút ra nhận xét về hiệu suất của A*. Trong một số trường hợp, nếu lời giải từ A* có số bước đi lớn hơn so với BFS, điều này có thể cho thấy rằng A* không đạt được hiệu suất tối ưu. Điều này có thể xảy ra khi hàm heuristic không đảm bảo admissible hoặc consistent, dẫn đến việc A* bị dẫn hướng vào các nút không tối ưu hoặc lựa chọn không tốt. Dưới đây là nhận xét về hiệu suất của A* trên từng màn chơi cụ thể:

Dựa vào *Bảng 6*: bảng thống kê số bước đi của BFS và A*, chúng ta có thể nhận xét về sự tối ưu của A* như sau:

- **Không tối ưu**: Không có trường hợp nào
- **Tối ưu**: Trong tất cả các màn chơi , kết quả của thuật toán A* là giống so với BFS, cho thấy rằng khi dùng hàm heuristic *chebysev* cho thuật toán A* thì sẽ đạt được hiệu suất tối ưu trong các màn chơi này.

Level	BFS	A*(chebysev)
1	12	12
2	9	9
3	15	15
4	7	7
5	20	20
6	19	19
7	21	21
8	97	97
9	8	8
10	33	33
11	34	34
12	23	23
13	31	31
14	23	23
15	105	105
16	34	34
17	NG	NG
18	DNF	DNF

Bảng 6: Thống kê số bước di chuyển giữa 2 thuật toán BFS(Breadth-First Search) và A*(euclidean)