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

BÀI TẬP 2

TRÍ TUỆ NHÂN TẠO

Sinh viên: Đỗ Quang Lực - 23520902

Giáo viên hướng dẫn: Lương Ngọc Hoàng

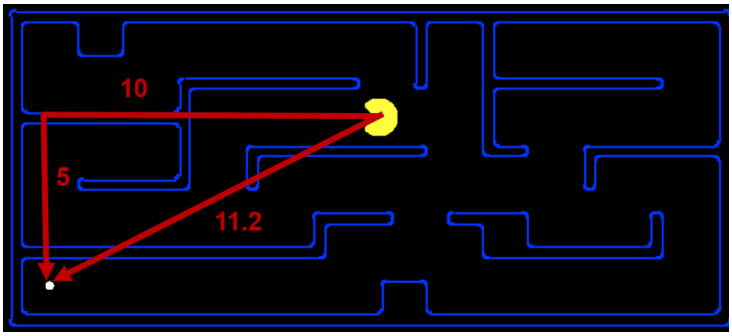
Ngày 30 tháng 3 năm 2025

Mục lục

1 Giới thiệu về Heuristic và thuật toán A*

1.1 Heuristic

Heuristic (hay hàm heuristics) là một phương pháp ước lượng chi phí còn lại để đạt đến mục tiêu trong các thuật toán tìm kiếm. Nó giúp thuật toán đưa ra quyết định nhanh hơn bằng cách đánh giá những bước đi nào có vẻ tiềm năng nhất.
Ví dụ: một heuristic đơn giản có thể là khoảng cách Euclidean hoặc Manhattan từ vị trí hiện tại đến đích.



Hình 1: Heuristic là khoảng cách Manhattan và Euclidean với trò chơi Pac-Man

1.2 Thuật toán A*

A* (A-star) là một thuật toán tìm kiếm tối ưu sử dụng heuristic để tìm đường đi ngắn nhất từ điểm bắt đầu đến điểm đích.
Công thức: A* sử dụng hàm đánh giá:

$$f(n) = g(n) + h(n)$$

Trong đó:

- $g(n)$ là chi phí thực tế từ điểm bắt đầu đến nút n
- $h(n)$ là hàm heuristic, ước lượng chi phí từ n đến mục tiêu.
- $f(n)$ là tổng chi phí ước tính từ điểm bắt đầu đến mục tiêu qua n.

Thuật toán sẽ ưu tiên chọn nút có hàm đánh giá thấp nhất chưa được mở để tìm kiếm lời giải.

1.3 Yêu cầu của bai tập

Tiếp nối bài tập 1, hãy cài đặt thuật toán A* và so sánh hiệu suất của A* với UCS khi giải trò chơi Sokoban.

2 Triển khai heuristic cho thuật toán A*

Ở đây chúng ta sẽ thử triển khai hai hàm heuristic dựa trên khoảng cách manhattan.

2.1 Random match heuristic

Ta tiến hành lọc ra những hộp và vị trí đích trống, ghép ngẫu nhiên từng cặp hộp và vị trí đích rồi tính tổng khoảng cách manhattan giữa các cặp.

```python
def random_match_heuristic(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 += (abs(sortposBox[i][0] - sortposGoals[i][0])) + (abs(sortposBox[i][1] - sortposGoals[i][1]))
    return distance
```

Hình 2: Triển khai ramdom match heuristic

Ưu điểm:

- Hàm có độ phức tạp thấp, dễ triển khai và tính toán nhanh
- Hiệu suất tốt hơn trên các màn chơi: cách tiếp cận này vẫn có thể cung cấp một đánh giá hợp lý về khoảng cách cần di chuyển trên các màn chơi nhỏ (số lượng hộp và đích ít)

Hạn chế:

- Do ghép cặp ngẫu nhiên, tổng chi phí di chuyển có thể cao hơn so với một chiến lược ghép tối ưu.
- Không đảm bảo rằng mỗi hộp được ghép với vị trí đích gần nhất, dẫn đến kết quả kém chính xác.

2.2 Hungarian heuristic

Tương tự như ý tưởng của random match heuristic, tuy nhiên ta sử dụng thuật toán Hungarian. Thuật toán này đảm bảo tổng khoảng cách di chuyển là nhỏ nhất có thể để tối ưu hóa việc ghép cặp hộp - đích.
Ưu điểm:

- Đảm bảo rằng mỗi hộp được ghép với vị trí đích tối ưu, giảm tổng quãng đường di chuyển.
- Tăng độ chính xác của heuristic, giúp thuật toán tìm kiếm hoạt động hiệu quả hơn.

Hạn chế:

- Thuật toán Hungarian có độ phức tạp cao hơn so với phương pháp heuristic ban đầu, có thể làm tăng thời gian tính toán trong các bài toán lớn.

```python
def hungarian_heuristic(posPlayer, posBox):
    from scipy.optimize import linear_sum_assignment
    completes = set(posGoals) & set(posBox)
    sortposBox = list(set(posBox) - completes)
    sortposGoals = list(set(posGoals) - completes)

    if not sortposBox:
        return 0

    cost_matrix = [[abs(bx - gx) + abs(by - gy) for gx, gy in sortposGoals] for bx, by in sortposBox]
    row_ind, col_ind = linear_sum_assignment(cost_matrix)

    return sum(cost_matrix[i][j] for i, j in zip(row_ind, col_ind))
```

Hình 3: Triển khai hungarian heuristc

2.3 Cài đặt chương trình

Tất cả source code đều được lưu trữ trên link github: BT2 - Heuristics & A_star search.

3 Bảng thống kê

Dưới đây là bảng thống kê về thời gian chạy và số nút đã được mở

Màn chơi	UCS		A*(random match)		A* (hungarian)	
	Thời gian (s)	Số nút	Thời gian (s)	Số nút	Thời gian (s)	Số nút
1	0.05622	720	0.01163	122	0.80828	91
2	0.00463	64	0.00301	39	0.02447	43
3	0.08549	509	0.00759	54	0.07157	116
4	0.00411	55	0.00201	29	0.02075	29
5	78.60527	357203	0.09268	485	0.57979	756
6	0.01173	250	0.01305	208	0.07454	214
7	0.58386	6046	0.07374	715	0.87154	1775
8	0.23324	2383	0.26024	2352	1.07354	2353
9	0.00864	74	0.00550	42	0.01472	26
10	0.01611	218	0.01777	198	0.07727	205
11	0.01746	296	0.02078	284	0.08566	289
12	0.09798	1225	0.05176	563	0.25686	625
13	0.19876	2342	0.16870	1699	0.80951	1870
14	3.41742	26352	1.09971	8108	7.25570	13000
15	0.32082	2505	0.32475	2183	1.32702	2249
16	17.67789	57275	0.30916	1286	10.69800	9499
17	...	...	...	...	...	...
18	...	...	...	...	...	...

Bảng 1: So sánh UCS, A*(ramdon match) và A*(hungarian)

Chú thích: những ô ... là những ô mà chạy thuật toán không tìm ra được lời giải.

4 Nhận xét

4.1 Về thời gian

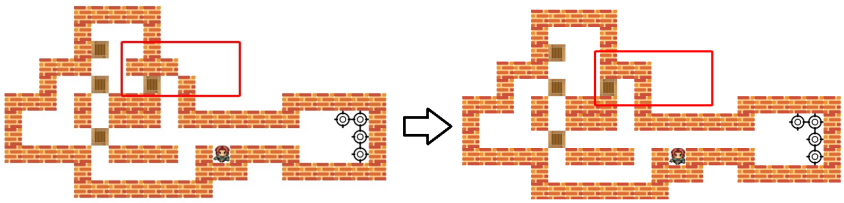
- UCS có thời gian chạy lớn nhất trong hầu hết các trường hợp do không sử dụng heuristic. Một số màn chơi có thời gian tính toán rất lớn (ví dụ: màn 5 và màn 17).
- A* với random match heuristic giảm đáng kể thời gian so với UCS. Tuy nhiên, do heuristic không tối ưu nên thời gian vẫn có thể tăng ở một số bài toán phức tạp.
- A* với heuristic Hungarian có thời gian chạy biến động: thấp hơn UCS nhưng đôi khi cao hơn heuristic ghép ngẫu nhiên. Điều này do chi phí tính toán thuật toán Hungarian có thể làm tăng thời gian xử lý, đặc biệt ở các màn đơn giản.

4.2 Về số nút được mở

- UCS mở số lượng nút rất lớn, đặc biệt ở các màn chơi phức tạp.
- A* với random match heuristic giảm số nút mở đáng kể so với UCS, chứng tỏ heuristic giúp định hướng tìm kiếm tốt hơn.
- A* với heuristic Hungarian mặc dù giúp tối ưu việc ghép cặp hộp và đích, nhưng thực tế lại mở nhiều nút hơn so với heuristic ghép ngẫu nhiên. Nguyên nhân có thể do heuristic này đôi khi đánh giá chưa tốt trong các trạng thái trung gian, dẫn đến việc mở rộng nhiều trạng thái hơn trước khi tìm thấy lời giải.

4.3 Nhận xét khác

Nhận thấy rằng màn 17 là màn không có lời giải vì bản đồ có vấn đề nên em có chỉnh lại bản đồ 17 để có đáp án.



Hình 4: Màn 17 trước và sau khi chỉnh

Màn 17 khi thử cho random match heuristic làm cost thì nhanh chóng tìm được 1 lời giải. Tuy không nhanh bằng DFS ở bài tập trước nhưng độ dài lời giải đưa ra ngắn hơn nhiều.

```
Number of nodes opened: 1712
Length path: 212
Runtime of astar: 1.39869 second.
```

Hình 5: Kết quả màn 17 với cost là hàm random match heuristic

Tương tự với hungarian heuristic đối với màn 17.

```
Number of nodes opened: 2194
Length path: 218
Runtime of astar: 3.36924 second.
```

Hình 6: Kết quả màn 17 với cost là hàm hungarian heuristic

Ngoài ra, với màn 5 thì số bước đi của UCS là 20 (ở bài tập 1), tuy nhiên ở A* (random match heuristic) lại đưa ra lời giải với độ dài là 22.

```
Number of nodes opened: 485
Length path: 22
Runtime of astar: 0.33277 second.
['u', 'l', 'u', 'R', 'd', 'r', 'U', 'U', 'U', 'L', 'u',
```

Hình 7: Kết quả màn 17 với cost là hàm hungarian heuristic

Nguyên nhân là do trong một số trường hợp, việc di chuyển một hộp có thể làm thay đổi đáng kể cách ghép hộp-đích tối ưu. Khi đó, chi phí heuristic có thể giảm đột ngột, vi phạm tính nhất quán (consistency) của thuật toán A*. Do đó có thể kết luận rằng lời giải đưa ra của A* không chắc sẽ là lời giải tối ưu.