Cơ sở trí tuệ nhân tạo :

Đồ án 1: Tìm kiếm Heuristic với A*

Nguyễn Sĩ Hùng (1612226) - Đoàn Minh Hiếu (1612198)

Ngày 27 tháng 10 năm 2018

Mục lục

1 Thông tin & Công việc & Mức độ hoàn thành **2**
1.1 Thông tin thành viên 2
1.2 Phân chia công việc Mức độ hoàn thành 2
1.3 Mức độ hoàn thành đồ án 2
1.4 Những vấn đề chưa thực hiện được 2
1.4.1 Giao diện đồ hoạ ARA 2

2 Thuật toán A* **3**
2.1 Nội dung thuật toán 3
2.2 Cấu trúc dữ liệu và cài đặt 3
2.3 Kết quả với các bộ test 4
2.4 Đề xuất heuristic khác cho thuật toán A* 6

3 Thuật toán ARA* **7**

4 Sơ đồ biểu diễn hệ thống phần mềm **8**

5 Tài liệu tham khảo **9**

1 Thông tin & Công việc & Mức độ hoàn thành

1.1 Thông tin thành viên

Thông tin thành viên	
Họ tên	Mã số sinh viên
Nguyễn Sĩ Hùng	1612226
Đoàn Minh Hiếu	1612198

1.2 Phân chia công việc Mức độ hoàn thành

Họ tên	Công việc	Mức độ hoàn thành
Đoàn Minh Hiếu	Tìm hiểu và code thuật toán A* và ARA	100%
	Đề xuất Heuristic	100%
Nguyễn Sĩ Hùng	Minh hoạ thuật toán A* và ARA	90%
Nguyễn Sĩ Hùng, Đoàn Minh Hiếu	Viết báo cáo	100%

1.3 Mức độ hoàn thành đồ án

Yêu cầu	Công việc	Mức độ hoàn thành
Yêu cầu 1	Cài đặt thuật toán A*	100%
	Chạy chương trình tham số dòng lệnh	100%
Yêu cầu 2	Tự đề xuất heuristic và chứng minh	100%
	Tìm hiểu thuật toán ARA	100%
	Minh hoạ thuật toán A* bằng giao diện đồ hoạ	100%
	Minh hoạ thuật toán ARA bằng giao diện đồ hoạ	80%

1.4 Những vấn đề chưa thực hiện được

1.4.1 Giao diện đồ hoạ ARA

Chương trình dòng lệnh đã có chức năng tìm kiếm với một khoảng thời gian bất kì nhưng giao diện đồ hoạ chưa có chức năng này.

2 Thuật toán A*

2.1 Nội dung thuật toán

A* là thuật toán tìm đường đi từ một nút khởi đầu (Start) tới nút đích (Goal) cho trước. Nó thuộc nhóm thuật toán tìm kiếm có thông tin (informed search).

Ý tưởng: Kết hợp ý tưởng của thuật toán UCS (Uniform Cost Search) với Greedy Search, nên A* là thuật toán tìm đường đầy đủ và tối ưu nhất. Thuật toán này đánh giá một nút dựa trên chi phí đi từ nút gốc đến nút đó – $g(n)$ cộng với chi phí từ nút đó đến đích – $h(n)$.

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

Vì

$g(n)$: chi phí đường đi từ nút gốc đến nút n

$h(n)$: ước lượng đường đi ngắn nhất từ nút n đến đích

Nên: $f(n)$: ước lượng chi phí của lời giải tốt nhất qua n. Do đó, để tìm đường đi ngắn nhất thì ta cứ lấy nút có f(n) nhỏ nhất đến khi tới đích.

Để đảm bảo A* là lời giải tối ưu thì hàm heuristic h(n) phải chấp nhận được (admissable). Một hàm heuristic chấp nhận được khi và chỉ khi giá trị của nó tại một nốt không bao giờ vượt quá chi phí đến đích thực sự tại nốt đó. Trong yêu cầu 1, ta sử dụng heuristic là khoảng cách euclid từ nút đến đích (goal).

```
#heuristic
import math
def h(a,b):  #euclid distance heuristic
  (x1,y1) =a
  (x2,y2)=b
  return math.sqrt((x2-x1)**2+(y2-y1)**2)
```

2.2 Cấu trúc dữ liệu và cài đặt

Cấu trúc dữ liệu:

+ Dùng 1 hàng đợi có độ ưu tiên frontier=PriorityQueue()

+ Dùng 1 dictionary cost_so_far = tính chi phí từ start đến nút đang xét ($g(n)$)

+ Dùng 1 dictionary come_from= để truy vết ra đường đi.

Thứ tự mở rộng:

1	2	3
8		4
7	6	5

Dùng hàm neighbor để xét các nút kế tiếp từ nút đang xét:

```
def neighbor(robot):
  (x,y)=robot
  xplus=[-1,-1,-1,0,1,1,1,0]
  yplus=[-1,0,1,1,1,0,-1,-1]
  listNeighbor = []
  for i in range(8):
    xx=x+xplus[i]
    yy=y+yplus[i]
    if 0<=xx<sizeOfGrid and 0<=yy<sizeOfGrid and grid[xx][yy]!='1':
      listNeighbor.append((xx,yy))
  return listNeighbor
```

Thuật toán sẽ pop phần tử từ frontier tức là nó lấy ra nút có f(x) nhỏ nhất để đi bước tiếp theo.
Từ các nút neighbor của nút hiện tại. Ta push nó vào frontier và vì frontier là PriorityQueue nên nó sẽ sort lại để khi pop từ frontier ra ta luôn có được nút với f(x) tại đó nhỏ nhất. Đến khi nào ta pop được goal, nghĩa là đường đi tối ưu đã được tìm thấy.

```
while not frontier.emtpy():
    current=frontier.get()
    if current==goal:
      break

    for next in neighbor(current):
      new_cost = cost_so_far[current]+1 #chi tinh tren edges thoi
      if next not in cost_so_far or new_cost<cost_so_far[next]:
        cost_so_far[next]=new_cost
        value = new_cost+h(goal,next)
        frontier.put(next,value)
        came_from[next]=current
```

2.3 Kết quả với các bộ test

Kết quả của giải thuật A* đã cài đặt qua 3 bộ test đặc trưng:
+ Bộ test 1: Tồn tại 1 đường đi từ Start đến Goal
input:
7
0 0
6 5
0 0 0 0 0 0 0
1 0 1 1 1 0 0
1 0 1 1 1 0 0
0 0 1 0 0 0 0
0 1 1 0 1 0 0
0 0 0 0 1 0 1
1 1 1 1 1 0 1

output:

11
(0, 0) (1, 1) (0, 2) (0, 3) (0, 4) (1, 5) (2, 5) (3, 5) (4, 5) (5, 5) (6, 5)
S - x x x - -
o x o o o x -
o - o o o x -
- - o - - x -
- o o - o x -
- - - - o x o
o o o o o G o
+ Bộ test 2: Không có đường đi từ Start đến Goal.
Input:
7
0 0
6 5
0 0 0 0 0 0 0
1 0 1 1 1 0 0
1 0 1 1 1 0 0
0 0 1 0 0 0 0
0 1 1 0 1 0 0
0 0 0 0 1 1 1
1 1 1 1 1 0 1
Output:
-1
+ Bộ test 3: Có đường đi và phức tạp.
input:
7
0 0
6 5
0 0 0 0 0 0 1
1 0 1 1 1 0 1
1 0 1 1 1 1 1
0 0 1 0 0 1 1
0 1 1 0 1 0 1
0 0 0 0 1 0 1
1 1 1 1 1 0 1

output:
12
(0, 0) (1, 1) (2, 1) (3, 1) (4, 0) (5, 1) (5, 2) (4, 3) (3, 4) (4, 5) (5, 5) (6, 5)
S - - - - - o
o x o o o - o
o x o o o o o
- x o - x o o
x o o x o x o
- x x - o x o
o o o o o G o

2.4 Đề xuất heuristic khác cho thuật toán A*

Heuristic đề xuất:

$$h(n) = \frac{dx}{2} + \frac{dy}{2}$$

Trong đó:
$dx = |x_{goal} – x_n|$
$dy = |y_{goal} – y_n|$

Một hàm heuristic chấp nhận được khi và chỉ khi giá trị của nó tại một nốt không bao giờ vượt quá chi phí đến đích thực sự tại nốt đó.
Gọi h*(n) là chi phí nhỏ nhất đến đích (goal) thực tế của nốt n.
Ta cần chứng minh $h(n) \leq h*(n)$
Nút hiện tại $S(n) = (x_n, y_n)$
Nút đích $S(goal) = (x_{goal}, y_{goal})$
Ta chứng minh trường hợp $x_n < x_{goal}, y_n < y_{goal}$.
Các trường hợp khác chứng minh tương tự.
Khi đó:

$$h(n) = \frac{x_{goal} - x_n}{2} + \frac{y_{goal} - y_n}{2}$$

Theo đề bài:

$$h*(n) = max(x_{goal} - x_n, y_{goal} - y_n)$$

(vì mỗi bước chéo được tính là 1)
Khi $x_{goal} - x_n \geq y_{goal} - y_n$
$\Rightarrow h*(n) = x_{goal} - x_n$

$$h(n) = \frac{x_{goal} - x_n}{2} + \frac{y_{goal} - y_n}{2} \leq 2\frac{x_{goal} - x_n}{2} = x_{goal} - x_n = h*(n)$$

Khi $y_{goal} - y_n > x_{goal} - x_n$
$\Rightarrow h*(n) = y_{goal} - y_n$

$$h(n) = \frac{x_{goal} - x_n}{2} + \frac{y_{goal} - y_n}{2} < 2\frac{y_{goal} - y_n}{2} = y_{goal} - y_n = h*(n)$$

(chứng minh xong)

3 Thuật toán ARA*

ARA* là viết tắt của Anytime Repair A*. Thuật toán này có được từ việc chỉnh sửa A*.
ARA* dùng để xử lý trường hợp người dùng muốn giới hạn thời gian tìm kiếm của A*, nghĩa là phải trả về kết quả đường đi (không cần phải tối ưu) trong khoảng thời gian cho phép.
Trong thuật toán A* ta có:

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

Khi ta thêm một hằng số E $>$ 1 vào trước h(x): f(x) = g(x)+Eh(x) thì thuật toán A* với f(x) mới sẽ chạy nhanh hơn. Nó thiên hơn về phía thuật toán tham lam. Khi ta xuất phát ở 1 giá trị E lớn và sau đó giảm dần với mong muốn sẽ tìm được đường đi tối ưu hơn. Nếu ta lặp lại thuật toán A* nhiều lần với các giá trị E khác nhau thì sẽ lãng phí các kết quả đã mở rộng được từ lần trước đó, dẫn tới thời gian để tìm được đường tối ưu tăng đáng kể. Vậy ta cần phải đảm bảo là các ô chỉ mở rộng đúng 1 lần. Mở rộng ở đây là khi nào chúng ta pop giá trị từ queue.
Ta sử dụng 3 queue:
– OPEN: lưu các trạng thái sắp được mở rộng. Trạng thái nào có fvalue nhỏ nhất sẽ được pop ra
– CLOSED: lưu các trạng thái đã mở rộng
– INCONS: lưu các trạng thái local inconsistency cho lần chạy tiếp theo
Trạng thái local inconsistency là trạng thái mà khi giá trị nó thay đổi sẽ dẫn đến giá trị của các trạng thái khác thay đổi theo
Mã giả:

```
procedure fvalue(s)
01 return g(s) + ε * h(s);

procedure ImprovePath()
02 while(fvalue(s_goal) > min_{s∈OPEN}(fvalue(s)))
03   remove s with the smallest fvalue(s) from OPEN;
04   CLOSED = CLOSED ∪ {s};
05   for each successor s' of s
06     if s' was not visited before then
07       g(s') = ∞;
08     if g(s') > g(s) + c(s, s')
09       g(s') = g(s) + c(s, s');
10       if s' ∉ CLOSED
11         insert s' into OPEN with fvalue(s');
12       else
13         insert s' into INCONS;
```

```
procedure Main()
01' g(s_goal) = ∞; g(s_start) = 0;
02' OPEN = CLOSED = INCONS = ∅;
03' insert s_start into OPEN with fvalue(s_start);
04' ImprovePath();
05' ε' = min(ε, g(s_goal)/ min_{s∈OPEN∪INCONS}(g(s)+h(s)));
06' publish current ε'-suboptimal solution;
07' while ε' > 1
08'   decrease ε;
09'   Move states from INCONS into OPEN;
10'   Update the priorities for all s ∈ OPEN according to fvalue(s);
11'   CLOSED = ∅;
12'   ImprovePath();
13'   ε' = min(ε, g(s_goal)/ min_{s∈OPEN∪INCONS}(g(s)+h(s)));
14'   publish current ε'-suboptimal solution;
```

Figure 3: ARA*

Ở lần chạy với E tiếp theo, ta merge các state trong OPEN và INCONS lại với fvalue ứng với E tiếp theo và để vào OPEN cho lần chạy tiếp theo mở rộng nhằm tìm ra đường đi tối ưu hơn khi còn thời gian chạy.

4 Sơ đồ biểu diễn hệ thống phần mềm

Phần	Hàm	Công dụng
Mylibrary	def h(a,b)	heuristic euclid
	def fvalue	Hàm tính f
	def neighbor(robot,sizeOfGrid,grid)	Tính các ô lân cận có thể đi tới được trong 1 grid
	def reconstruct_path(came_from, start, goal)	Lần vết lại đường đi
Priority Queue	def empty(self)	Kiểm tra hàng đợi ưu tiên có rỗng hay không
	def put(self, item, value)	Thêm phần tử có giá trị (value, item) vào đầu hàng đợi ưu tiên
	def get(self)	Lấy giá trị của phần tử cuối cùng của hàng đợi ưu tiên
	def topValue(self)	Lấy giá trị value của phần tử đầu tiên
	def topItem(self)	Lấy giá trị item của phần tử đầu tiên
A*	def Astar_search(start, goal, sizeOfGrid, grid)	Thực hiện tìm kiếm A*
ARA	def ARA(start, goal, E, sizeOfGrid, grid, interval)	Thực hiện tìm kiếm ARA
GUI	def initButton(self)	Thêm các nút (Previous, Return, Continue, Start, Continue) cũng như các radiobutton (ARA, A*) vào giao diện đồ hoạ
	def initCanvas(self)	Thiết lập các ô và gán các hàm xử lý
	def onCreateBarrier(self, event), def onDeleteBarrier(self, event), def onCreateGoal(self, event), def onCreateStart(self, event), def onResize(self, event)	Các hàm xử lý trong initCanvas như xử lý thêm, xoá điểm bắt đầu, kết thúc, vật chắn, thu phóng
	def setColor(self, x, y, colour), def getColor(self, x, y)	Lấy, gán colour cho ô (x,y)
	def prevAction(self, event), def continueAction(self, event), def startAction(self, event), def returnAction(self, event)	Xử lý các nút bấm
	Các hàm của file ARA, A* hay Mylibrary	Được viết lại cho phù hợp với yêu cầu đồ hoạ

5 Tài liệu tham khảo

1. Introduction to A* from Red Blob Games
2. Paper ARA* của Maxim Likhachev, Geoff Gordon and Sebastian Thrun
3. Tkinter by example