



ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΜ&ΜΥ
Τεχνητή Νοημοσύνη
1^η Άσκηση
Ακ. έτος 2011-2012

Γερακάρης Βασίλης Α.Μ.: 03108092
Λύρας Γρηγόρης Α.Μ.: 03109687

9 Φεβρουαρίου 2012

Υλοποίηση αλγορίθμου A*

1.1 Παρουσίαση προβλήματος και σκιαγράφηση λύσης

Στην άσκηση αυτή καλούμαστε να υλοποιήσουμε τον αλγόριθμο αναζήτησης A* με σκοπό να οδηγηθούν με βέλτιστο τρόπο 2 ρομπότ σε ένα προκαθορισμένο σημείο συνάντησης, αποφεύγοντας τις πιθανές συγκρούσεις.

Έχοντας ως δεδομένο ότι τα δύο ρομπότ έχουν γνώση της θέσης και του πλάνου του άλλου ρομπότ, καθώς και πλήρη γνώση της αίθουσας, επιλέξαμε να κάνουμε μια παρατήρηση /παραδοχή που διευκολύνει σημαντικά την δομή του προγράμματός μας:

Αντί τα 2 ρομπότ να πραγματοποιούν τις κινήσεις/επιλογές τους εναλλάξ, θεωρούμε, χωρίς βλάβη της γενικότητας, ότι στις συγκρούσεις το 2ο ρομπότ θα έχει μια 'nice' συμπεριφορά, παραχωρώντας τη βέλτιστη κίνηση στο 1ο ρομπότ. Με τον τρόπο αυτό, μπορούμε να εκτελέσουμε σειριακά τον αλγόριθμο A*, πρώτα για το 1ο ρομπότ και στη συνέχεια για το 2ο, έχοντας ήδη δεδομένες τις κινήσεις του 1ου.

Ο χώρος των καταστάσεων μας δίνεται ως ένα grid με O (στις θέσεις που επιτρέπεται η κίνηση) και X (στις θέσεις όπου βρίσκονται εμπόδια). Η βασική ιδέα πίσω από τον αλγόριθμό μας είναι ότι οι θέσεις με X κωδικοποιούνται ως '-1', οι θέσεις με O με '0', ενώ η θέση στην οποία θα βρίσκεται το 1ο ρομπότ στο k-οστό βήμα με 'k'.

Με τον τρόπο αυτό, μπορούμε να περιορίσουμε τις επιλογές του 2ου ρομπότ σε κάθε βήμα, αναγκάζοντας το να ψάξει εναλλακτικό μονοπάτι ή να μείνει στάσιμο γι'αυτό το βήμα (αν το συμφέρει). Καταφέρνουμε έτσι να αποφύγουμε τις πιθανές συγκρούσεις, ενώ ταυτόχρονα δε θυσιάζουμε τη βελτιστότητα της λύσης μας.

Για την υλοποίηση του παραπάνω αλγορίθμου, επιλέχθηκε ως κατάλληλη γλώσσα η Python, επειδή μας δίνει τη δυνατότητα να επικεντρωθούμε στα πλέον σημαντικά κομμάτια του προβλήματος (αλγόριθμος και στρατηγική αναζήτησης), να γράψουμε πυκνό και ευανάγνωστο κώδικα, ενώ με τη χρήση ενός Just-in-time compiler όπως ο Psycho έχουμε ίδιο χρόνο εκτέλεσης όπως αν γράφαμε σε μια γλώσσα χαμηλότερου επιπέδου.

1.2 Επιλογή Δομών Δεδομένων

Στην υλοποίησή μας επιλέξαμε να χρησιμοποιήσουμε λίστες λιστών της Python, όπου ο κάθε κόμβος στη λίστα του A* είχε 4 στοιχεία:

1. Το άθροισμα ευριστικής και κόστους ($h + c$)
2. Το κόστος (c)
3. Την τετμημένη του σημείου (x)
4. Την τεταγμένη του σημείου (y)

Επιπλέον, χρησιμοποιείται ένα λεξικό (dict) προκειμένου να πραγματοποιηθεί η αναδόμηση της επιλεγμένης διαδρομής ενός ρομπότ. Τα dictionaries είναι associative arrays, ένας τύπος αντικειμένων που μοιάζει με λίστα, αλλά όπου σε μοναδικά κλειδιά αντιστοιχίζονται (όχι απαραίτητα μοναδικές) τιμές.

1.3 Βασικές συναρτήσεις-τελεστές

2

1.4 Ευριστικές μέθοδοι

- Δεδομένου ότι δουλεύουμε σε δισδιάστατο χώρο με μόνο οριζόντια και κάθετη κίνηση (όχι διαγώνια), επιλέχθηκε ως πιο έγκυρος και αποδοτικός υποεκτιμητής (admissible heuristic) η απόσταση Manhattan:

$$ManhDist = |x - x_T| + |y - y_T|$$

Η χρήση υποεκτιμητή μας εγγυάται τη βελτιστότητα της λύσης.

- Ένας λειτουργικός υπερεκτιμητής (non-admissible heuristic), είναι το άθροισμα των τετραγώνων των αποστάσεων από τον προορισμό:

$$SqDist = (x - x_T)^2 + (y - y_T)^2$$

Αν χρησιμοποιήσουμε ένα υπερεκτιμητή, θα επεκτείνουμε πιθανώς πολύ λιγότερους κόμβους κατά την αναζήτηση μας. Το χρονικό κέρδος αυτό όμως αντισταθμίζεται με την πιθανότητα να μην είναι βέλτιστη η λύση που προκύπτει, αφού ορισμένοι βέλτιστοι κόμβοι αποφεύγονται λόγω του αυξημένου συνολικού κόστους που εισάγεται με τον υπερεκτιμητή.

1.5 Έλεγχος και επίλυση συγκρούσεων

Όπως εξηγήσαμε πριν, ο αλγόριθμός μας εκτελείται πρώτα για το ένα ρομπότ, και στη συνέχεια για το δεύτερο, σε ένα τροποποιημένο grid. Στην περίπτωση που στο μέτωπο αναζήτησης του 2ου για το k-οστό βήμα είναι η θέση που πήγε το 1ο ρομπότ στο k βήμα, ανιχνεύεται πιθανή σύγκρουση. Το 2ο ρομπότ έχει πλέον 2 επιλογές:

1. Να περιμένει (wait) για ένα γύρο, επιτρέποντας στο 1ο να προχωρήσει και έπειτα ακολουθώντας το (μιας και οι επιλογές του 1ου είναι εξ'ορισμού βέλτιστες), δίνοντας στο σημείο της σύγκρουσης κόστος $c + 1$
2. Να αναζητήσει κάποια εναλλακτική πορεία προς το σημείο συνάντησης.

Από τις 2 αυτές επιλογές, τελικά θα διαλέξει αυτή που έχει το λιγότερο κόστος (σε αριθμό βημάτων)

Εκτελώντας τον αλγόριθμο A^* 2 φορές, ώστε στη μία το ρομπότ A να παραχωρεί προτεραιότητα στις συγκρούσεις, ενώ στη 2η το B να παραχωρεί προτεραιότητα και επιλέγοντας αυτό που παράγει μικρότερο αριθμό βημάτων τερματισμού, καταλήγουμε στο τελικά βέλτιστο αποτέλεσμα της αναζήτησης.

```

1: procedure resolve(newGrid[x][y], currStep)
2:   if newGrid[x][y] == currStep then
3:      $cost_1 \leftarrow (len(Astar((x, y), target)) + 1)$ 
4:      $path_1 \leftarrow (path(Astar((x, y), target)))$ 
5:     newgrid[x][y]  $\leftarrow Invalid$ 
6:      $cost_2 \leftarrow len(Astar(currPos, target))$ 
7:      $path_1 \leftarrow (path(Astar((currPos, target)))$ 
8:     if  $cost_1 < cost_2$  then
9:        $cost \leftarrow cost_1$ 
10:       $path \leftarrow path_1$ 
11:   else
12:      $cost \leftarrow cost_2$ 
13:      $path \leftarrow path_2$ 

```

5

6

Here be dragons!

- Οι συναρτήσεις που καλούνται για την ανάγνωση της εισόδου:

3

```

19         #'X' marks obstacle, 'O' marks open space
20         if i == 'X':
21             a.append(-1)
22         elif i=='O':
23             a.append(0)
24     return a
25
26 def parseInput(f):
27     lines = int(f.readline().split()[0])
28     target = tuple(map(int,f.readline().split()))
29     robo1_initstate = tuple(map(int,f.readline().split()))
30     robo2_initstate = tuple(map(int,f.readline().split()))
31     text = map(retbools,f.readlines())
32     return (target,robo1_initstate,robo2_initstate,text)

```

- Οι ευριστικές που χρησιμοποιούνται στον αλγόριθμο:

```

1  #!/usr/bin/python
2  #/* -.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.
3  #
4  ## File Name : heuristics.py
5  #
6  ## Purpose :
7  #
8  ## Creation Date : 09-02-2012
9  #
10 ## Last Modified : Thu 9 Feb 2012 22:09:12 EET
11 #
12 ## Created By : Greg Liras <gregliras@gmail.com>
13 #
14 #_._._._._._._._._._._._._._._._._.* /
15
16 def manhattanDist(point1,point2):
17     return abs(point1[0][0]-point2[0])+abs(point1[0][1]-point2[1])
18
19 def squaredDist(point1,point2):
20     return (point1[0][0]-point2[0])**2 + (point1[0][1]-point2[1])**2

```

- Το κυρίως σώμα του A^* :

```
1  #!/usr/bin/python  
2  #!/* -. .-. .-. .-. .-. .-. .-. .-. .-. .-. .-. .-. .-. .-. .-. .-  
3  #  
4  ** File Name : astar.py  
5  #  
6  ** Purpose :  
7  #  
8  ** Creation Date : 24-12-2011  
9  #  
10 ** Last Modified : Thu 09 Feb 2012 06:52:30 PM EET  
11 #
```

```

12  ## Created By : Greg Liras <gregliras@gmail.com>
13  #
14  #_._._._._._._._._._._._._._._._._.*_/
15
16
17  def nextNodes((a,b)):
18      return [(a-1,b),(a,b)),((a,b-1),(a,b)),((a+1,b),(a,b)),((a,b+1),(a,b))]
19
20  def putinlist(starque,(h,c,xy)):
21      if not starque:
22          starque.append((h,c,xy))
23      return starque
24
25  for i in range(len(starque)):
26      (sh,sc,sxy) = starque[i]
27      if sxy == xy:
28          starque.pop(i)
29          (sh,sc,sxy) = min((sh,sc,sxy),(h,c,xy))
30          starque.append((sh,sc,sxy))
31      return starque
32
33  def astar(startpoint,finishpoint,grid,heuristic):
34      ancestors={}
35      #ancestors is a dictionary which stores the ancestors of each point
36      #this will be used in the end to rebuild the path
37      passedlist=[]
38      passedlist.append(startpoint)
39      #passedlist contains nodes that have been processed already
40      starque=[]
41      #num of rows
42      sizex=len(grid)
43      #num of columns
44      sizey=len(grid[0])
45      possible = map(lambda x:(heuristic(x,finishpoint)+1,1,x),nextNodes(startpoint))
46      #each point has these characteristics
47       #(heuristic,cost,((x,y),father))
48      for (h,c,((x,y),father)) in possible:
49          #checking for bounds and then checking if grid[x][y]==True
50          #now should be grid[x][y]==0 ??
51          if x >= 0 and y >= 0 and x < sizex and y < sizey and grid[x][y] != -1:
52              passedlist.append((x,y))
53              ancestors[(x,y)]=father
54              starque.append((h,c,(x,y)))
55
56      ind = starque.index(min(starque))
57      (h,c,(x,y)) = starque.pop(ind)
58      while(grid[x][y] == c + 1 ):
59          print "Conflict in [" ,x," ," ,y," ] on step", c, ", Robot #2 recalculating.."
60          starque.append((h+1,c+1,(x,y)))

```

```

61     ind = starque.index(min(starque))
62     print "Robot #2 trying..",starque[ind][2][0],starque[ind][2][1]
63     (h,c,(x,y)) = starque.pop(ind)
64
65     #ind = starque.index(min(starque))
66     #find index of tuple with the lowest heuristic+cost
67     #nxt = starque.pop(ind)
68     nxt = (h,c,(x,y))
69     #remove if from the queue
70     #and store it in nxt
71     #nxt = (minh,c,((x,y),father))
72     current = nxt[2]
73     currentCost = nxt[1]
74     #current cost is the cost so far that is stored in nxt
75     #goal = zip( *nextNodes( finishpoint ) )[0]
76     #print goal
77
78     while(current!=finishpoint):
79         #untill you find the end
80         possible = map(lambda x:(heuristic(x,finishpoint)+currentCost+1,currentCost
81         #find the next possible list
82         for (h,c,((x,y),father)) in possible:
83             if x >= 0 and y >= 0 and x < sizex and y < sizey and grid[x][y] != -1:
84                 #check what is in possible list, make sure its sane
85                 #starque = putinlist(starque,(h,c,(x,y)))
86                 if(x,y) not in passedlist:
87                     passedlist.append((x,y))
88                     #if i havend passed this so far
89                     #then store it and insert the coordinates in ancestors dictio
90                     ancestors[(x,y)]=father
91                     starque.append((h,c,(x,y)))
92                     #if i have passed this already then i can reach this with a lower
93                     #so i don't need to save x,y
94
95     ind = starque.index(min(starque))
96     (h,c,(x,y)) = starque.pop(ind)
97     while(grid[x][y] == c + 1 ):
98         print "Conflict in [",x,"",y,"] on step", c, ", Robot #2 recalculating
99         starque.append((h+1,c+1,(x,y)))
100        ind = starque.index(min(starque))
101        print "Robot #2 trying..",starque[ind][2][0],starque[ind][2][1]
102        (h,c,(x,y)) = starque.pop(ind)
103
104
105        #find index of tuple with the lowest heuristic+cost
106        nxt = (h,c,(x,y))
107        #remove if from the queue
108        #and store it in nxt
109        #nxt = (minh,c,((x,y),father))

```



```

30 def flushgrid(grid):
31     global lgrid
32     lgrid = []
33     for i in grid:
34         lgrid.append(map(revertMap,list(i)))
35
36
37 def designpath(color,(sx,sy),(fx,fy),finalists):
38     global lgrid
39     for (x,y) in finalists:
40         if ( lgrid[x][y].startswith("\033")):
41             lgrid[x][y] = "\033["+joinedColor+"m*\033[0m"
42         else:
43             lgrid[x][y]="\033["+color+"m*\033[0m"
44     lgrid[sx][sy]="S"
45     lgrid[fx][fy]="F"
46
47 def printpath():
48
49     print "\033[47m"+(" "*(len(lgrid[0])+2))+"\033[0m"
50     for i in lgrid:
51         print "\033[47m \033[0m"+" ".join(i)+"\033[47m \033[0m"
52     print "\033[47m\033[1;34m"+(" "*(len(lgrid[0])+2-len(names)))+names+"\033[0m"
53
54 def printgrid((cx,cy),(fx,fy),grid):
55     global lgrid
56     if not lgrid:
57         for i in grid:
58             lgrid.append(map(revertMap,i))
59     lgrid[cx][cy]="@"
60     lgrid[fx][fy]="#"
61
62     for i in lgrid:
63         print " ".join(i)
64     print "----"
65
66 def modifygrid(finalists,grid):
67     i=1
68     for (x,y) in finalists:
69         grid[x][y]=i
70         i+=1
71     return grid

```

- Ο controller που καλεί τις παραπάνω συναρτήσεις για να παραχθεί το τελικό αποτέλεσμα:

```
1 #!/usr/bin/python
2 #/* -.-.-.-.-.
3 #
4 /* File Name : controler.py
```

```

5 #
6 ## Purpose : 1st assignment in Artificial Intelligence
7 #
8 ## Creation Date : 24-12-2011
9 #
10 ## Last Modified : Thu 9 Feb 2012 19:00:43 EET
11 #
12 ## Created By : Greg Liras <gregliras@gmail.com>
13 #
14 #_._._._._._._._._._._._._._._._._._._._._.* /
15
16 import sys
17
18 from inparser import parseInput
19 from astar import astar
20 import heuristics
21 from grids import flushgrid, printpath, designpath, modifygrid
22 #import psyco
23
24 #psyco.full()
25
26 def main():
27     if len(sys.argv) < 3:
28         print "Usage: %s <inputfile> <mode (A/N)>" %sys.argv[0]
29         return -1
30     f=open(sys.argv[1],"r")
31     (target,r1,r2,field) = parseInput(f)
32     f.close()
33     if sys.argv[2] == "A":
34         print "Using Manhattan Distance as admissible heuristic"
35         heuristic = heuristics.manhattanDist
36     elif sys.argv[2] == "N":
37         print "Using Square Distances as non-admissible heuristic"
38         heuristic = heuristics.squaredDist
39     else:
40         print "Wrong mode, choose A for admissible heuristic or N for non-admissible"
41         return -1
42     finalists1 = astar(r1,target,field,heuristic)
43     field = modifygrid(finalists1,field)
44     finalists2 = astar(r2,target,field,heuristic)
45     flushgrid(field)
46     designpath("1;34",r1,target,finalists1)
47     designpath("1;33",r2,target,finalists2)
48     printpath()
49     print "Max length in steps:", max(len(finalists1),len(finalists2))
50     print "finalists 1 len: ", len(finalists1)
51     print "finalists 2 len: ", len(finalists2)
52
53 if __name__=="__main__":

```

