

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

Υλοποίηση αλγορίθμου Α*

1.1 Παρουσίαση προβλήματος και σκιαγράφηση λύσης

Στην άσκηση αυτή καλούμαστε να υλοποιήσουμε τον αλγόριθμο αναζήτησης Α* 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, παραχωρόντας τη βέλτιστη κίνηση στο 1ο ρομπότ. Με τον τρόπο αυτό, μπορούμε να εκτελέσουμε σειριακά τον αλγόριθμο Α*, 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 extension module όπως το Psyco έχουμε αντίστοιχο 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 στι λίστα του Α* 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 (γ)

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.

Algorithm 1 Conflict resolution

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

1.6 Χρόνος εκτέλεσης ανά μέγεθος εισόδου

5

1.7 Εκτύπωση βημάτων-συγκρούσεων-εναλλακτικών δρόμων σε παράδειγμα εκτέλεσης

6

1.8 Πηγαίος κώδικας

Here be dragons!

• Οι συναρτήσεις που καλούνται για την ανάγνωση της εισόδου:

```
#!/usr/bin/python
    #* File Name : inparser.py
    #* Purpose : Input parsing of state space
    #* Creation Date : 24-12-2011
    #* Last Modified : Fri 10 Feb 2012 04:17:27 PM EET
11
    #* Created By : Greg Liras <gregliras@gmail.com>
12
13
    #* Created By : Vasilis Gerakaris <vqerak@qmail.com>
14
15
16
    #_._._..*/
17
    def retbools(st):
        a=[]
19
20
        for i in st:
            #'X' marks obstacle, 'O' marks open space
```

```
if i == 'X':
22
                a.append(-1)
23
             elif i=='0':
24
25
                a.append(0)
26
         return a
    def parseInput(f):
28
         lines = int(f.readline().split()[0])
29
         robo1_initstate = tuple(map(int,f.readline().split()))
30
         robo2_initstate = tuple(map(int,f.readline().split()))
31
32
         target = tuple(map(int,f.readline().split()))
         text = map(retbools,f.readlines())
33
         return (target,robo1_initstate,robo2_initstate,text)
34
```

• Οι ευριστικές που χρησιμοποιούνται στον αλγόριθμο:

```
#!/usr/bin/python
2
    #/* -.-.-.-.-.
4
    #* File Name : heuristics.py
    #* Purpose :
    #* Creation Date : 09-02-2012
    #* Last Modified : Fri 10 Feb 2012 04:17:27 PM EET
10
11
    #* Created By : Greg Liras <gregliras@gmail.com>
12
13
    #* Created By : Vasilis Gerakaris <vgerak@gmail.com>
14
15
    #_._...*/
16
17
    def manhattanDist(point1,point2):
18
       return abs(point1[0][0]-point2[0])+abs(point1[0][1]-point2[1])
19
20
    def squaredDist(point1,point2):
21
       return (point1[0][0]-point2[0])**2 + (point1[0][1]-point2[1])**2
22
```

• Το κυρίως σώμα του Α*:

```
#!/usr/bin/python
    #/* -.-.-.-.-.
2
3
    #* File Name : astar.py
4
    #* Purpose : Main body of A* algorithm
7
    #* Creation Date : 24-12-2011
    #* Last Modified : Fri 10 Feb 2012 17:50:59 EET
10
11
    #* Created By : Greg Liras <gregliras@gmail.com>
13
    #* Created By : Vasilis Gerakaris <vgerak@gmail.com>
14
15
16
    17
18
    def nextNodes((a,b)):
19
        return [((a-1,b),(a,b)),((a,b-1),(a,b)),((a+1,b),(a,b)),((a,b+1),(a,b))]
20
21
22
    def putinlist(starque,(h,c,xy)):
        if not starque:
23
            starque append((h,c,xy))
24
25
            return starque
26
        for i in range(len(starque)):
27
            (sh,sc,sxy) = starque[i]
            if sxy == xy:
29
30
                starque.pop(i)
                (sh,sc,sxy) = min((sh,sc,sxy),(h,c,xy))
31
                starque.append((sh,sc,sxy))
32
33
                return starque
34
    def astar(startpoint,finishpoint,grid,heuristic,robotID=1):
35
```

```
36
         #ancestors is a dictionary which stores the ancestors of each point
37
         #this will be used in the end to rebuild the path
38
         passedlist=[]
39
40
         passedlist.append(startpoint)
         #passedlist contains nodes that have been processed already
41
         starque=[]
42
         #num of rows
43
         sizex=len(grid)
44
         #num of columns
45
46
         sizey=len(grid[0])
         possible = map(lambda x:(heuristic(x,finishpoint)+1,1,x),nextNodes(startpoint))
47
         #each point has these characteristics
48
         \#(heuristic, cost, ((x,y), father))
         for (h,c,((x,y),father)) in possible:
50
              #checking for bounds and then checking if grid[x][y] == True
51
              #now should be grid[x][y]==0 ??
52
              if x \ge 0 and y \ge 0 and x < sizey and y < sizex and grid[x][y] != -1:
53
54
                  passedlist.append((x,y))
55
                  ancestors[(x,y)]=father
                  starque.append((h,c,(x,y)))
56
         ind = starque.index(min(starque))
58
         (h,c,(x,y)) = starque.pop(ind)
59
          #Conflict detection on first step
60
         while(grid[x][y] == c + 1):
61
              print "Conflict in [%d,%d] on step %d , Robot #%d recalculating.." %(x,y,c,robotID)
62
              starque.append((h+1,c+1,(x,y)))
63
              ind = starque.index(min(starque))
64
              #Consideration of alternative path
              print "Robot #%d trying.." %(robotID),starque[ind][2][0],starque[ind][2][1]
66
67
              (h,c,(x,y)) = starque.pop(ind)
68
         #ind = starque.index(min(starque))
69
         #find index of touple with the lowest heuristic+cost
70
71
         #nxt = starque.pop(ind)
         nxt = (h,c,(x,y))
72
         #remove if from the queue
73
         #and store it in nxt
74
         \#nxt = (minh, c, ((x,y), father))
75
         current = nxt[2]
76
         currentCost = nxt[1]
77
         #current cost is the cost so far that is stored in nxt
78
         #goal = zip( *nextNodes( finishpoint ) )[0]
79
         #print goal
80
81
         while(current!=finishpoint):
82
              #until you find the end
83
84
              possible = map(lambda x:(heuristic(x,finishpoint)+currentCost+1,currentCost+1,x),nextNodes(current))
              #find the next possible list
85
              for (h,c,((x,y),father)) in possible:
86
                  if x \ge 0 and y \ge 0 and x < sizex and y < sizey and grid[x][y] != -1:
                      #check what is in possible list, make sure its same
88
                      \#starque = putinlist(starque, (h, c, (x, y)))
90
                      if(x,y) not in passedlist:
91
                          passedlist.append((x,y))
                          #if I havend passed this so far
92
                          #then store it and insert the coordinates in ancestors dictionary
93
0.4
                          ancestors[(x,y)]=father
                          starque.append((h,c,(x,y)))
                      #if I have passed this already then I can reach this with a lower cost
96
                      #so i don't need to save x,y
97
98
99
              ind = starque.index(min(starque))
              (h,c,(x,y)) = starque.pop(ind)
100
              #Conflict detection for all steps
101
              while(grid[x][y] == c + 1):
102
                  print "Conflict in [%d,%d] on step %d , Robot #%d recalculating.." %(x,y,c,robotID)
                  starque.append((h+1,c+1,(x,y)))
104
105
                  ind = starque.index(min(starque))
                  #Consideration of alternative paths
106
                  print "Robot #%d trying.." %(robotID),starque[ind][2][0],starque[ind][2][1]
107
                  (h,c,(x,y)) = starque.pop(ind)
109
110
```

```
#find index of touple with the lowest heuristic+cost
111
              nxt = (h,c,(x,y))
112
113
              #remove if from the queue and store it in nxt
              \#nxt = (minh, c, ((x,y), father))
114
115
              current = nxt[2]
              currentCost = nxt[1]
116
              #current cost is the cost so far that is stored in nxt
117
          print "Found it after %d expanded nodes" %len(passedlist)
118
119
          finalists=[]
120
121
          i =finishpoint
          finalists.append(i)
122
          \#starting from the end build the path list
123
          #following the directions in the ancestors dictionary
124
          while i !=startpoint:
125
126
              i = ancestors[i]
127
              finalists.append(i)
          #reverse the path so it starts from the beginning
128
129
          finalists.reverse()
130
          #put it in the queue
          return (finalists, len(passedlist))
131
```

• Οι συναρτήσεις που σχετίζονται με την εμφάνιση και τροποποίηση του χώρου καταστάσεων:

```
#!/usr/bin/python
     #/* -.-.-.-.-.
     \#* File Name : grids.py
4
     #* Purpose : Functions related with grid modifying & output
6
     #* Creation Date : 24-12-2011
     #* Last Modified : Fri 10 Feb 2012 04:17:27 PM EET
10
11
     #* Created By : Greg Liras <gregliras@gmail.com>
12
     #* Created By : Vasilis Gerakaris <vgerak@gmail.com>
14
15
16
     #_._...*/
17
18
    lgrid=[]
     blockChar = unichr(0x258A)
19
     joinedColor = "0:32"
20
     names = "Vasilis Gerakaris - Gregory Liras"
21
22
     #Choose what will be printed depending on element in position
23
24
     def revertMap(b):
       if b=="0" or b=="#":
25
26
            return b
        elif b>=0:
27
            return " "
                                         #unused space gets blankspace
28
         elif b<0:
           return "\033[41m \033[0m"
                                        #obstacle gets red solid box
30
31
        else:
32
            return b
33
34
     def flushgrid(grid):
        global lgrid
35
        lgrid = []
36
37
         for i in grid:
            lgrid.append(map(revertMap,list(i)))
38
39
40
     def designpath(color,(sx,sy),(fx,fy),finalists):
         global lgrid
41
42
         for (x,y) in finalists:
             if ( lgrid[x][y].startswith("\033")):
                                                         #if both robots use position, color with green
43
                lgrid[x][y] = "\033["+joinedColor+"m*\033[0m"
44
45
                                                         #else keep designated robot color
                lgrid[x][y]="\033["+color+"m*\033[0m"
46
         lgrid[sx][sy]="S"
47
         lgrid[fx][fy]="F"
48
49
50
     def printpath():
        print "\033[47m"+(" "*(len(lgrid[0])+2))+"\033[0m"
51
        for i \underline{in} lgrid:
52
```

```
print "\033[47m \033[0m"+"".join(i)+"\033[47m \033[0m"
53
         print "\033[47m\033[1;34m"+(" "*(len(lgrid[0])+2-len(names)))+names+"\033[0m"
54
55
     def printgrid((cx,cy),(fx,fy),grid):
56
         global lgrid
57
         if not lgrid:
58
             for i in grid:
59
                 lgrid.append(map(revertMap,i))
60
         lgrid[cx][cy]="@"
61
         lgrid[fx][fy]="#"
62
63
         for i in lgrid:
64
         print "".join(i)
print "----"
65
67
     def modifygrid(finalists,grid):
68
69
         for (x,y) in finalists:
70
71
             grid[x][y]=i
                                   #robot 1 marks its steps on the grid to
             i+=1
                                   #be unusable (on same turn) by robot 2
72
73
         return grid
```

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

```
#!/usr/bin/python
     #/* -.-.-.-.-.
2
     #* File Name : controler.py
4
     #* Purpose : 1st assignment in Artificial Intelligence
6
     #* Creation Date : 24-12-2011
    #* Last Modified : Fri 10 Feb 2012 04:44:06 PM EET
10
11
    #* Created By : Greg Liras <gregliras@gmail.com>
12
     #* Created By : Vasilis Gerakaris <vgerak@gmail.com>
14
15
16
     17
18
    import sys
19
20
    from inparser import parseInput
    from astar import astar
21
    import heuristics
22
    from grids import flushgrid,printpath,designpath,modifygrid
23
24
    def main():
25
26
        if len(sys.argv) < 3:
            print "Usage: %s <inputfile> <mode (A/N)>" %sys.argv[0]
27
            return -1
28
        f=open(sys.argv[1],"r")
        (target,r1,r2,initialfield) = parseInput(f)
30
31
        f.close()
        field = map(list,initialfield)
32
        modeCheck = sys.argv[2]
33
        while modeCheck != "A" and modeCheck != "N" and modeCheck != "a" and modeCheck != "n":
34
            print "Wrong mode, choose A for admissible heuristic or N for non-admissible"
35
            modeCheck = raw_input('Enter A or N --->')
36
        if modeCheck == "A" or modeCheck == 'a':
37
            print "Using Manhattan Distance as admissible heuristic"
38
            heuristic = heuristics.manhattanDist
39
40
            print "Using Square Distances as non-admissible heuristic"
41
42
            heuristic = heuristics.squaredDist
        print "\nLEGEND:"
43
        print "\033[1;34m Robot1 path \n\033[1;33m Robot2 path \n\033[0;32m Joined path \033[0m"
44
        print "\n \033[0;34m ====== Robot 2 plays 'nice' ====== \033[0m"
        total=0
46
        (finalists1,nodes) = astar(r1,target,field,heuristic,1)
47
        total += nodes
        field = modifygrid(finalists1,field)
49
50
        (finalists2,nodes) = astar(r2,target,field,heuristic,2)
        total += nodes
51
        flushgrid(field)
52
```

```
designpath("1;34",r1,target,finalists1)
53
         designpath("1;33",r2,target,finalists2)
54
         printpath()
         max1 = max(len(finalists1),len(finalists2))
56
         print "Max length in steps:", max1-1
print "\t Robot 1 took:\t\t", len(finalists1)-1, " steps"
57
         print "\t Robot 2 (nice) took:\t", len(finalists2)-1, " steps"
59
         flushgrid(field)
60
         print "\n \033[0;34m ====== Robot 1 plays 'nice' ====== \033[0m"
61
         field = map(list,initialfield)
62
         modeCheck = sys.argv[2]
63
         (finalists2,nodes) = astar(r2,target,field,heuristic,2)
64
         total += nodes
65
         field = modifygrid(finalists2,field)
         (finalists1,nodes) = astar(r1,target,field,heuristic,1)
67
         total += nodes
68
69
         flushgrid(field)
         designpath("1;34",r1,target,finalists1)
70
         designpath("1;33",r2,target,finalists2)
71
         printpath()
72
         max2 = max(len(finalists1),len(finalists2))
73
         print "Max length in steps:", max2-1
75
         print "\t Robot 1 (nice) took:\t", len(finalists1)-1, " steps"
         print "\t Robot 2 took:\t\t", len(finalists2)-1, " steps"
76
         print "\n ===== RESULT ====="
77
         if max1 < max2:</pre>
78
             print "1st strategy (Robot 2 plays nice) is optimal"
79
         elif max2 < max1:</pre>
            print "2nd strategy (Robot 1 plays nice) is optimal"
81
         else:
            print "Both strategies yield same result"
83
         print "Total nodes considered:", total
84
85
86
     if __name__=="__main__":
87
         main()
```