



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

11 Φεβρουαρίου 2012

Υλοποίηση αλγορίθμου A*

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 extension module όπως το Psyco έχουμε αντίστοιχο χρόνο εκτέλεσης όπως αν γράφαμε σε μια γλώσσα χαμηλότερου επιπέδου.

2 Επιλογή Δομών Δεδομένων

Στην υλοποίησή μας επιλέξαμε να χρησιμοποιήσουμε λίστες από τούπλες 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.

3 Βασικές συναρτήσεις-τελεστές

Οι συναρτήσεις που σχεδιάστηκαν για την υλοποίηση του αλγορίθμου, όπως παρατίθενται στο 8ο κομμάτι της αναφοράς, είναι οι εξής:

3.1 Ανάγνωση εισόδου

-
-
-

3.2 Ευριστικές μέθοδοι

-
-

3.3 Κυρίως σώμα του A*

-
-
-

3.4 Χώρος καταστάσεων

-
-
-

3.5 Controller

-
-
-

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.

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.

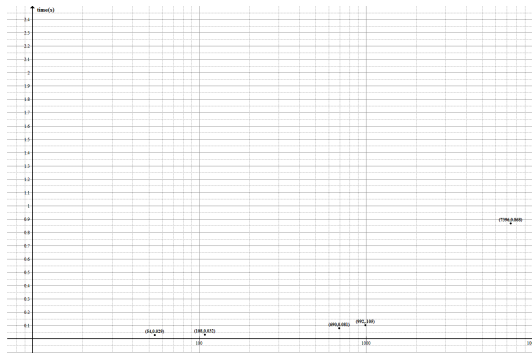
Algorithm 1 Conflict resolution

```
1: procedure resolve(newGrid[x][y], currStep)
2:   if newGrid[x][y] == currStep then
3:      $cost_1 \leftarrow \text{len}(\text{Astar}((x, y), \text{target})) + 1$ 
4:      $path_1 \leftarrow (\text{path}(\text{Astar}((x, y), \text{target})))$ 
5:     newgrid[x][y]  $\leftarrow$  Invalid
6:      $cost_2 \leftarrow \text{len}(\text{Astar}(\text{currPos}, \text{target}))$ 
7:      $path_1 \leftarrow (\text{path}(\text{Astar}(\text{currPos}, \text{target})))$ 
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:       $path \leftarrow path_2$ 
```

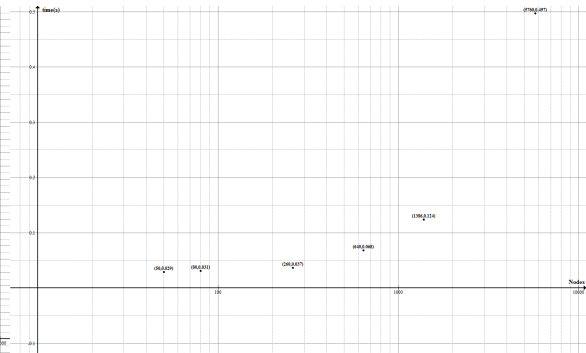
6 Χρόνος εκτέλεσης ανά μέγεθος εισόδου

Testcase	Nodes	Time
1	54	0.029
2	108	0.032
3	690	0.081
4	992	0.105
5	7396	0.868
6	14406	2.383

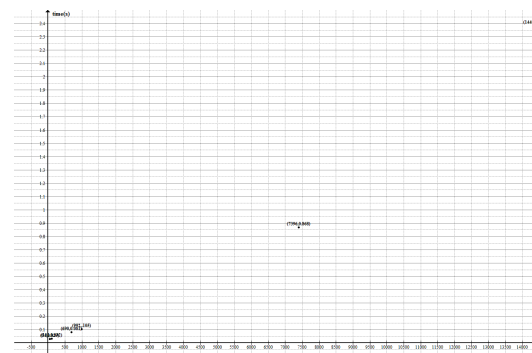
Testcase	Nodes	Time
1	50	0.029
2	80	0.031
3	260	0.037
4	640	0.068
5	5760	0.497
6	1386	0.124



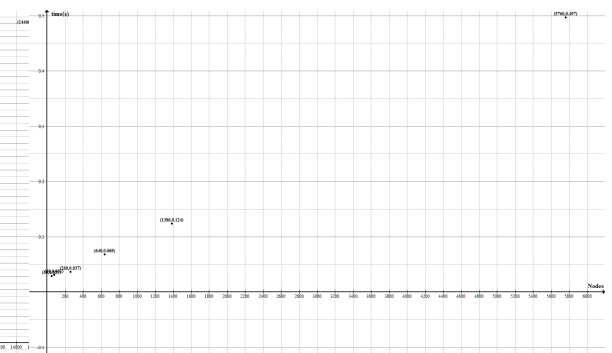
Σχήμα 1: Admissible (Σχήμα 7)



Σχήμα 2: Non Admissible (Σχήμα 8)



Σχήμα 3: Admissible (Σχήμα 9)




Σχήμα 4: Non Admissible (Σχήμα 10)

7 Εκτύπωση βημάτων-συγκρούσεων-εναλλακτικών δρόμων σε παράδειγμα εκτέλεσης


```
master@gentoo:AI/Programs/roboLab (*)
>>= ./controller.py testcases/test1 A
Using Manhattan Distance as admissible heuristic

LEGEND:
Robot1 path
Robot2 path
Joined path

===== Robot 2 plays 'nice' =====
Found it after 13 expanded nodes
Conflict in [4,2] on step 2 , Robot #2 recalculating..
Robot #2 trying.. 5 3
Conflict in [4,3] on step 3 , Robot #2 recalculating..
Robot #2 trying.. 5 4
Conflict in [4,4] on step 4 , Robot #2 recalculating..
Robot #2 trying.. 5 5
Conflict in [4,5] on step 5 , Robot #2 recalculating..
Robot #2 trying.. 4 2
Found it after 14 expanded nodes


Vasilis Gerakaris - Gregory Liras
Max length in steps: 5
Robot 1 took: 5 steps
Robot 2 (nice) took: 5 steps

===== Robot 1 plays 'nice' =====
Found it after 14 expanded nodes
Conflict in [4,2] on step 2 , Robot #1 recalculating..
Robot #1 trying.. 4 2
Found it after 13 expanded nodes


Vasilis Gerakaris - Gregory Liras
Max length in steps: 5
Robot 1 (nice) took: 5 steps
Robot 2 took: 5 steps

===== RESULT =====
Both strategies yield same result
Total nodes considered: 54
```

Σχήμα 5: Testcase 1

8 Πηγαίος κώδικας

- Οι συναρτήσεις που καλούνται για την ανάγνωση της εισόδου:

```

1 #!/usr/bin/python
2 /* .....*/
3 #
4 ** File Name : inparser.py
5 #
6 ** Purpose : Input parsing of state space
7 #
8 ** Creation Date : 24-12-2011
9 #
10 ** Last Modified : Sat 11 Feb 2012 04:15:25 AM EET
11 #
12 ** Created By : Greg Liras <gregliras@gmail.com>
13 #
14 ** Created By : Vasilis Gerakaris <vgerak@gmail.com>
15 #
16 #.....*/
17
18 import heuristics
19 from sys import argv,exit
20 def retbools(st):
21     a=[]
22     for i in st:
23         #'X' marks obstacle, 'O' marks open space
24         if i == 'X':
25             a.append(-1)
26         elif i=='O':
27             a.append(0)
28     return a
29
30 def parseInput(f):
31     lines = int(f.readline().split()[0])
32     robo1_initstate = tuple(map(int,f.readline().split()))
33     robo2_initstate = tuple(map(int,f.readline().split()))
34     target = tuple(map(int,f.readline().split()))
35     text = map(retbools,f.readlines())
36     return (target,robo1_initstate,robo2_initstate,text)
37
38 def parseUser():
39     if len(argv) < 3:
40         print "Usage: %s <inputfile> <mode (A/N)>" %argv[0]
41         exit(-1)
42     modeCheck = argv[2]
43     while modeCheck != "A" and modeCheck != "N" and modeCheck != "a" and modeCheck != "n":
44         print "Wrong mode, choose A for admissible heuristic or N for non-admissible"
45         modeCheck = raw_input('Enter A or N --->')
46     if modeCheck == "A" or modeCheck == 'a':
47         print "Using Manhattan Distance as admissible heuristic"
48         heuristic = heuristics.manhattanDist
49     else:
50         print "Using Square Distances as non-admissible heuristic"
51         heuristic = heuristics.squaredDist
52     try:
53         f=open(argv[1],"r")
54     except IOError:
55         print "The file you have entered does not exist"
56         exit(-1)
57     (target,r1,r2,initialfield) = parseInput(f)
58     f.close()
59     return (target,r1,r2,initialfield,heuristic)

```

- 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 : Fri 10 Feb 2012 04:17:27 PM EET
```



```

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

```

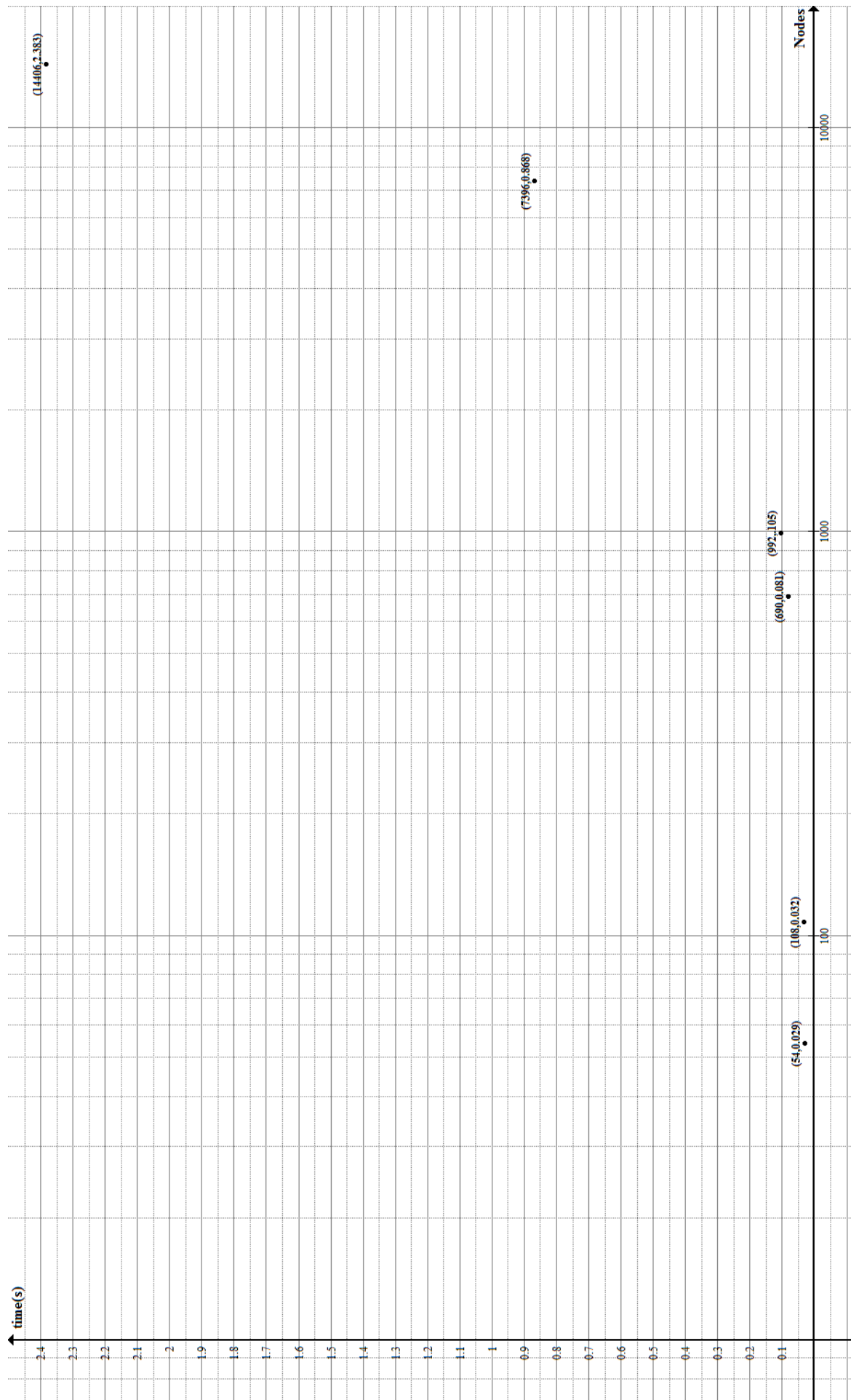
- Ο controller που καλεί τις παραπάνω συναρτήσεις για να παραχθεί το τελικό αποτέλεσμα:

[illegible]

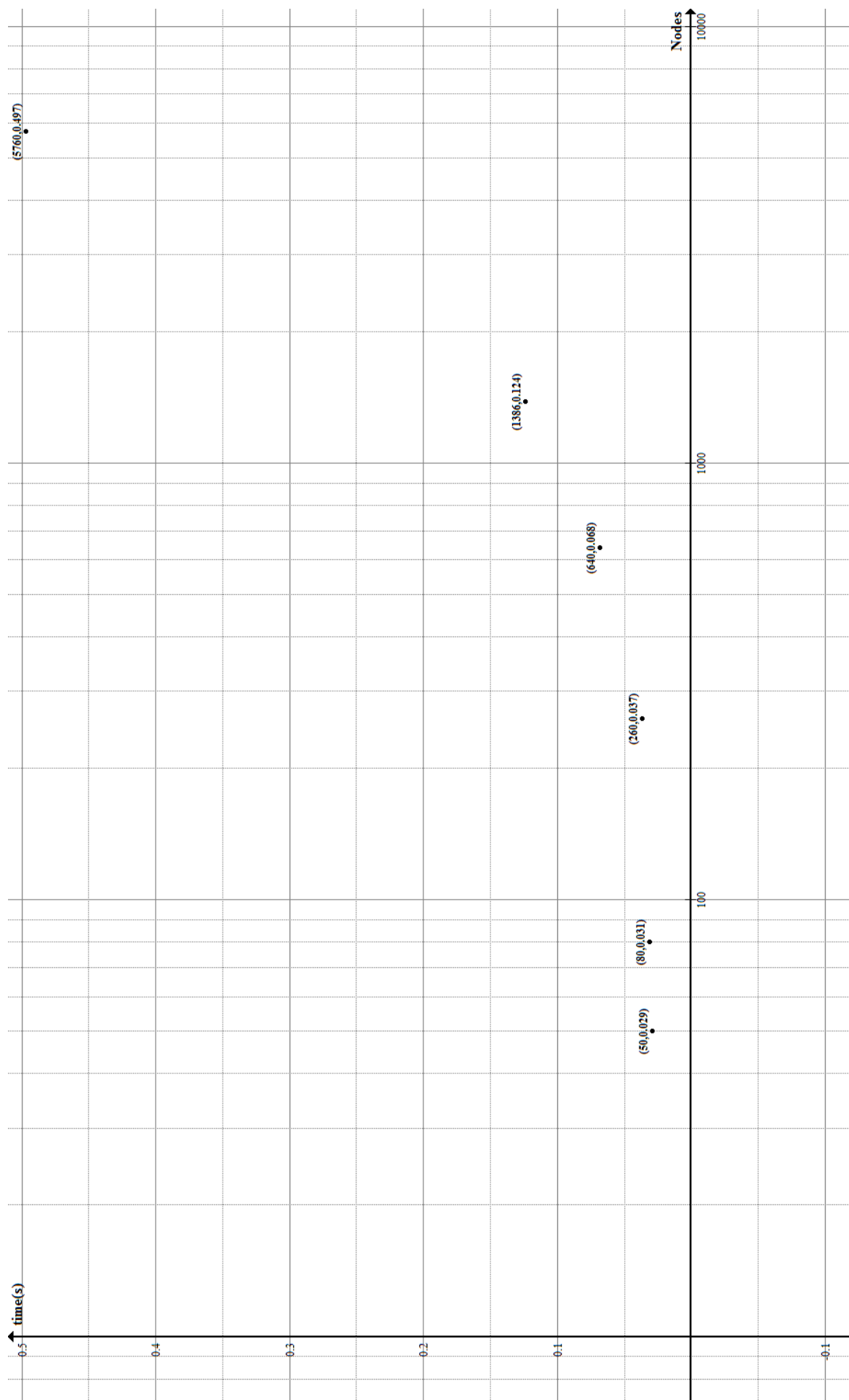
```

50     total += nodes
51     flushgrid(field)
52     designpath("1;34",r1,target,finalists1)
53     designpath("1;33",r2,target,finalists2)
54     printpath()
55     max2 = max(len(finalists1),len(finalists2))
56     print "Max length in steps:", max2-1
57     print "\t Robot 1 (nice) took:\t", len(finalists1)-1, " steps"
58     print "\t Robot 2 took:\t\t", len(finalists2)-1, " steps"
59     print "\n ===== RESULT ====="
60     if max1 < max2:
61         print "1st strategy (Robot 2 plays nice) is optimal"
62     elif max2 < max1:
63         print "2nd strategy (Robot 1 plays nice) is optimal"
64     else:
65         print "Both strategies yield same result"
66     print "Total nodes considered:", total
67
68
69 if __name__=="__main__":
70     main()

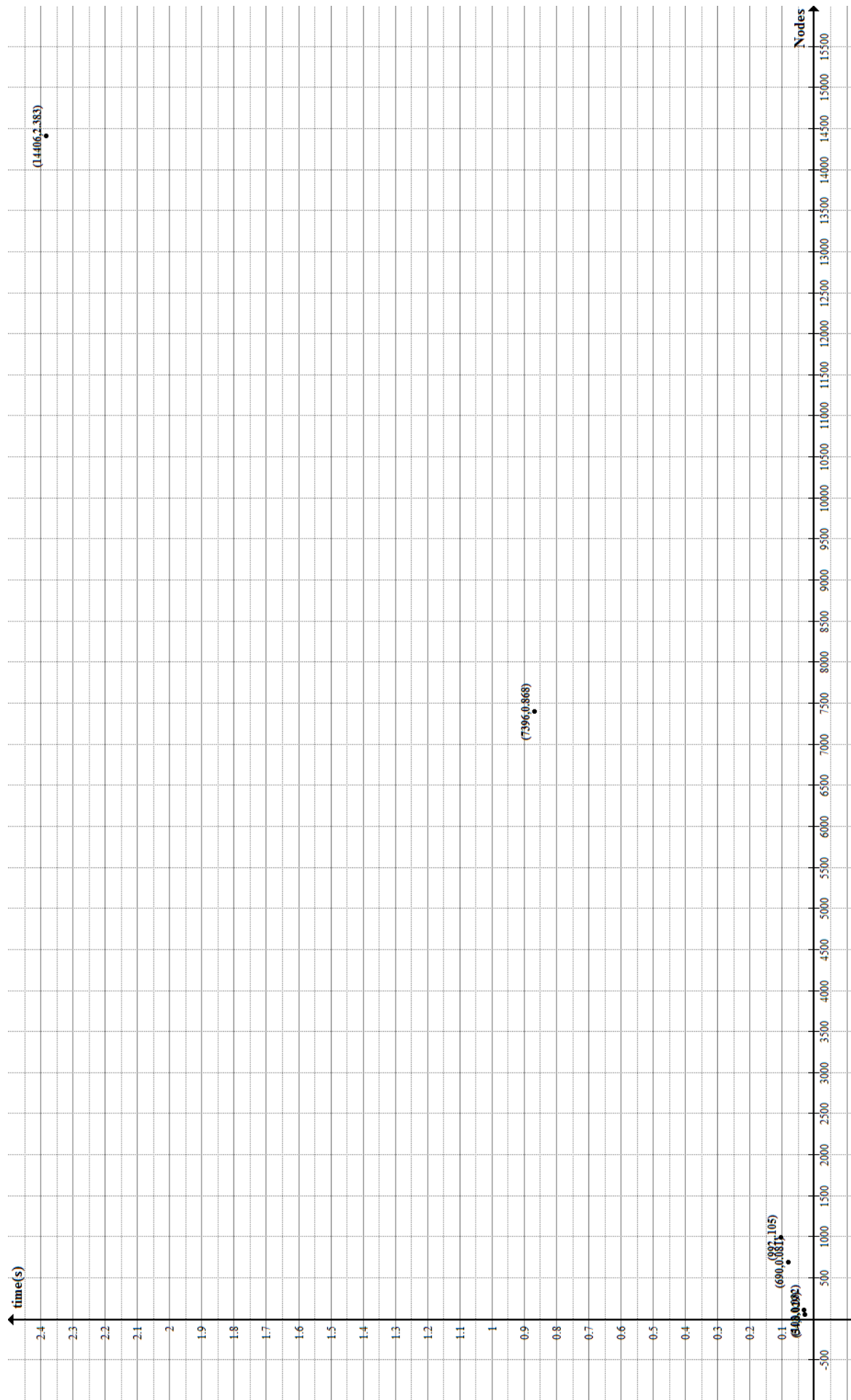
```



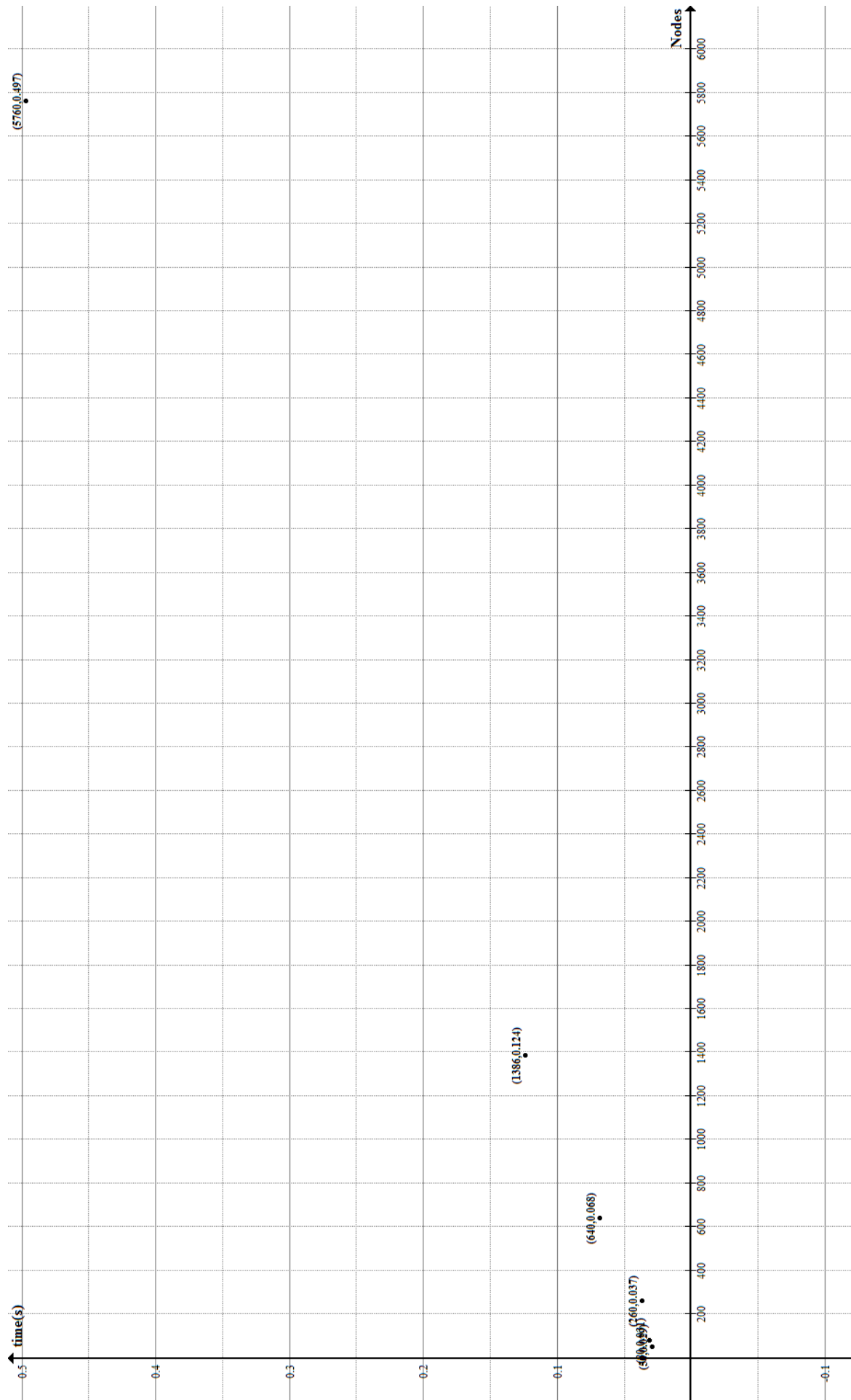
Σχήμα 7: Admissible heuristic



Σχήμα 8: Admissible heuristic



Σχήμα 9: Admissible heuristic



Σχήμα 10: Admissible heuristic