Διονύσης Χαλεπλής
ics24060

Blocks World

Άσκηση 1η
Τεχνητή Νοημοσύνη

Περιεχόμενα

Εκτέλεση κώδικα

Για να μπορέσει ο χρήστης να εκτελέσει τον κώδικα θα πρέπει να έχει εγκατεστημένη μια πρόσφατη, κατά προτίμηση, έκδοση της γλώσσας python.

Η εκτέλεση γίνεται μέσω της παρακάτω εντολής μέσω τερματικού.

```
python3 bw.py <algorithm> <problem> <output_file>
```

Algorithm option, επιλογή ενός από τους τέσσερις αλγορίθμους αναζήτησης

- depth
- breadth
- astar
- best

Problem option, επιλογή ενός αρχείου προβλήματος από τον κατάλογο input_files

Output_file option, επιλογή οποιουδήποτε ονόματος θέλει ο χρήστης να αποθηκευτεί το αποτέλεσμα του αλγορίθμου που επιλέχθηκε.

Παράδειγμα εντολής για τον αλγόριθμο breadth στο πρόβλημα probBLOCKS-6-0-.pddl και όνομα αρχείου out.txt

```
python3 bw.py breadth input_files/probBLOCKS-6-0.pddl
out.txt
```

Ανάλυση κώδικα

Επεξήγηση τμημάτων κώδικα που εμφανίζονται συχνά στο project

Η κλάση Node χρησιμοποιήθηκε σχεδόν σε όλους τους αλγορίθμους για λόγους ευκολίας μιας και κρατάμε τον κόμβο γονέα, όπου το οποίο μας διευκόλυνε στην εύρεση του μονοπατιού λυσης. Επιπρόσθετα μπορούμε να καταγράφουμε κάθε ‘μετακίνηση’ των κύβων με την ιδιότητα action!

```
class Node:
  def __init__(self, state, parent=None, action=None):
   self.state = state
   self.parent = parent
   self.action = action
```

Η κλάση Node βρίσκεται στην πιο απλή μορφή μιας και για να χρησιμοποιηθεί στους astar ή best θα πρέπει να προστεθούν επιπλέον ιδιότητες, όπως το κόστος βάθους.

Η χρήση φίλτρου όσον αφορά το initial state του προβλήματος.

```
list(filter(None, state))
```

Η εντολή αυτή χρησιμοποιείται για να μικρύνει ένα state το οποίο θα περιέχει άδειες λίστες, π.χ. [[‘A’, ‘B’, ‘C’], [], []] θα γίνει [[‘A’, ‘B’, ‘C’]]. Προτιμήθηκε μια τέτοια προσέγγιση διότι ανεβάζει την απόδοση οποιουδήποτε αλγορίθμου και γλυτώνουμε την δημιουργία άσκοπων παιδιών.

Επεξήγηση Ευρετικής Συνάρτησης

Η χρήση ευρετικής συνάρτησης για τον astar και για τον best.

```
def calculate_misplaced(self, node):
   filter_node = list(filter(None, node.state))
   for i, stack in enumerate(filter_node):
     for j, block in enumerate(stack):
       if block != self.goal_state[0][j]:
         node.misplaced_cost += 1 + (len(stack) - j)
       else:
         for z in range(j-1, -1, -1):
           if self.goal_state[0][z]!=stack[z]:
             node.misplaced_cost+=j-z
       if len(stack)>1:
         node.misplaced_cost+=1
```

Η ευρετική ελέγχει αν το block βρίσκεται στην σωστή θέση και αν όχι αυξάνει το κόστος κατά το μέγεθος του current stack μείων την θέση του block συν ένα.

Το τρίτο foor-lop εκτελείτε αν το block βρίσκεται στην σωστή θέση και ελέγχει αν τα από κάτω blocks του είναι στην σωστή θέση, αν όχι το current_state τρώει κλιμακούμενη ποινή.
Για παράδειγμα:

- init_state = [[a,b,c], [d]]
- goal_state = [[d,a,c],[b]]

Έστω ελέγχουμε το block **c** βλέπουμε οτι βρίσκεται στην θέση 2 στο init_state, (θέση 2 προκύπτει αν αρχίσουμε την αρίθμηση από το μηδέν),
όπου είναι και η σωστή θέση εμείς όμως θέλουμε να προτιμησουμε μια λύση που δεν θα έχει από κάτω λάθος blocks γιατί μετά πάλι θα χρειαστούμε extra κινήσεις για να τα ξαναβγάλουμε. Στην συγκεκριμένη περίπτωση το state αυτό θα φάει ποινή 1 για το block b και 2 για το block a. Συμπεραίνουμε ότι όσο πιο κάτω στο stack πάμε τόσο μεγαλύτερη η ποινή.

Τέλος, υπάρχει και ένας επιπλέον έλεγχος όπου ουσιαστικά αν το stack έχει μέγεθος μεγαλύτερο του ένα τρώει ποινή συν 1.

Μετρικές

Σε αυτήν την ενότητα θα παρουσιαστούν μόνο στιγμιότυπα από τα μεγαλύτερα προβλήματα που μπορεί να επιλύσει αποτελεσματικά ο κάθε αλγόριθμος. Συγκεκριμένα, τα προβλήματα κυμαίνονται από το probBLOCKS-4-0.pddl μέχρι το probBLOCKS-60-1.pddl. Για παράδειγμα, αν ένας αλγόριθμος μπορεί να επιλύσει μέχρι το probBLOCKS-11-0.pddl, θα παρουσιαστεί μόνο το στιγμιότυπο από αυτό το πρόβλημα. Στο τέλος, θα παρουσιαστεί ένας αναλυτικός πίνακας αποδόσεων για όλα τα προβλήματα που δοκιμάστηκαν.

Depth-First-Search

Ο αλγόριθμος σε βάθος λύνει αποδοτικά μέχρι και το probBLOCKS-8-0.pddl

```
▶time python3 bw.py depth input_files/probBLOCKS-8-0.pddl out.txt
Init state = [['E', 'G', 'A'], ['F', 'H', 'D'], ['B'], ['C'], [], [], [], []]
Goal state = [['B', 'G', 'A', 'C', 'H', 'E', 'F', 'D']]

python3 bw.py depth input_files/probBLOCKS-8-0.pddl out.txt  2.43s user 0.37s system 98% cpu 2.841 total

Exercises/Exercise_1/src
▶cat out.txt | wc -l
1064
```

Επίδοση 2.5secs και λύση με 1064 βήματα.

Breadth-First-Search

Ο αλγόριθμος σε πλάτος λύνει αποδοτικά μέχρι και το probBLOCKS-8-0.pddl

```
Exercises/Exercise_1/src
▶time python3 bw.py breadth input_files/probBLOCKS-8-0.pddl out.txt
Init state = [['E', 'G', 'A'], ['F', 'H', 'D'], ['B'], ['C'], [], [], [], []]
Goal state = [['B', 'G', 'A', 'C', 'H', 'E', 'F', 'D']]

python3 bw.py breadth input_files/probBLOCKS-8-0.pddl out.txt  6.71s user 0.07s system 99% cpu 6.798 total

Exercises/Exercise_1/src
▶cat out.txt | wc -l
9
```

Επίδοση 7secs και λύση με 9 βήματα.

A* search

Ο αλγόριθμος a* λύνει αποδοτικά μέχρι και το τελευταίο πρόβλημα.

```
Exercises/Exercise_1/src
▸ time python3 bw.py astar input_files/probBLOCKS-60-1.pddl out.txt
Init state = [['I', 'T1', 'L', 'C2', 'K1', 'O1', 'J', 'Y1', 'M', 'M1'], ['E1', 'Z1', 'V', 'A', 'W1', 'C', 'E2', 'V1', 'H2', 'R1', 'D2', 'R', 'F1', 'T', 'N', 'B1'], ['N1'
, 'X', 'P', 'S', 'B', 'H', 'G1', 'S1', 'C1', 'W', 'U', 'G2', 'P1', 'F', 'O', 'Y', 'L1', 'X1', 'B2', 'Z', 'I1', 'K', 'J1', 'G', 'A1', 'D', 'Q1', 'A2', 'U1', 'E'], ['F2',
'D1'], ['H1'], ['Q'], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [],
 [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], []]
Goal state = [['E2', 'O', 'D', 'V1', 'Y', 'K', 'D1', 'Z1', 'S1', 'A1', 'Q1', 'H2', 'C2', 'M', 'Q', 'U', 'U1', 'R', 'B1', 'W1', 'E', 'L', 'T', 'I1', 'X1', 'Y1', 'L1', 'G2
', 'F', 'V', 'H1', 'H', 'E1', 'A', 'S', 'G1', 'P', 'J1', 'J', 'A2', 'K1', 'C1', 'C', 'F1', 'N1', 'P1', 'T1', 'O1', 'M1', 'B', 'F2', 'W', 'I', 'N', 'Z', 'D2', 'R1', 'X',
'B2', 'G']]

python3 bw.py astar input_files/probBLOCKS-60-1.pddl out.txt  25.17s user 0.35s system 99% cpu 25.562 total

Exercises/Exercise_1/src
▸ cat out.txt | wc -l
113
```

Επίδοση 25secs και λύση με 113 βήματα.

Best-First-Search

Ο αλγόριθμος best λύνει αποδοτικά μέχρι και το τελευταίο πρόβλημα μιας και μοιράζονται την ίδια ευρετική.

```
Exercises/Exercise_1/src
▸ time python3 bw.py best input_files/probBLOCKS-60-1.pddl out.txt
Init state = [['I', 'T1', 'L', 'C2', 'K1', 'O1', 'J', 'Y1', 'M', 'M1'], ['E1', 'Z1', 'V', 'A', 'W1', 'C', 'E2', 'V1', 'H2', 'R1', 'D2', 'R', 'F1', 'T', 'N', 'B1'], ['N1'
, 'X', 'P', 'S', 'B', 'H', 'G1', 'S1', 'C1', 'W', 'U', 'G2', 'P1', 'F', 'O', 'Y', 'L1', 'X1', 'B2', 'Z', 'I1', 'K', 'J1', 'G', 'A1', 'D', 'Q1', 'A2', 'U1', 'E'], ['F2',
'D1'], ['H1'], ['Q'], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [],
 [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], []]
Goal state = [['E2', 'O', 'D', 'V1', 'Y', 'K', 'D1', 'Z1', 'S1', 'A1', 'Q1', 'H2', 'C2', 'M', 'Q', 'U', 'U1', 'R', 'B1', 'W1', 'E', 'L', 'T', 'I1', 'X1', 'Y1', 'L1', 'G2
', 'F', 'V', 'H1', 'H', 'E1', 'A', 'S', 'G1', 'P', 'J1', 'J', 'A2', 'K1', 'C1', 'C', 'F1', 'N1', 'P1', 'T1', 'O1', 'M1', 'B', 'F2', 'W', 'I', 'N', 'Z', 'D2', 'R1', 'X',
'B2', 'G']]

python3 bw.py best input_files/probBLOCKS-60-1.pddl out.txt  25.45s user 0.45s system 99% cpu 25.945 total

Exercises/Exercise_1/src
▸ cat out.txt | wc -l
113
```

Επίδοση 25.5secs και λύση με 113 βήματα.

Πίνακας Μετρικών

Seconds/Steps	Depth	Breadth
probBLOCKS-4-0.pddl	0.02 secs / 6 steps	0.03 secs / 3 steps
probBLOCKS-5-2.pddl	0.07 / 241	0.06 / 13
probBLOCKS-6-0.pddl	0.05 / 24	0.05 / 12
probBLOCKS-7-0.pddl	15.76 / 108.652	1.26 / 20
probBLOCKS-8-0.pddl	2.47 / 1.064	6.78 / 9
probBLOCKS-9-0.pddl	-	-

Seconds/Steps	A*	Best
probBLOCKS-4-0.pddl	0.02 secs / 3 steps	0.04 secs / 3 steps
probBLOCKS-10-0.pddl	0.04 / 17	0.04 / 17
probBLOCKS-15-0.pddl	0.09 / 23	0.11 / 23
probBLOCKS-20-0.pddl	0.26 / 36	0.25 / 36
probBLOCKS-30-0.pddl	1.27 / 52	1.28 / 52
probBLOCKS-40-0.pddl	4.26 / 72	4.31 / 72
probBLOCKS-50-0.pddl	11.20 / 92	11.25 / 92
probBLOCKS-55-0.pddl	17.34 / 104	17.20 / 104
probBLOCKS-60-0.pddl	25.15 / 113	25 / 113
probBLOCKS-60-1.pddl	24.74 / 113	25.27 / 113

Συμπεράσματα

Οι αλγόριθμοι τυφλής αναζήτησης όσο μεγαλώνει το μέγεθος του προβλήματος τόσο πιο αργοί γίνονται διότι παράγουν όλο και περισσότερα παιδία από όλους τους κόμβους ακόμη και αν ένας κόμβος δεν είναι αποτελεσματικός. Ο breadth σε σχέση με τον depth μπορεί να αργήσει σε κάποιο πρόβλημα αλλά θα βρει καλύτερη λύση. Ο depth ίσως κερδίζει λίγο σε απόδοση μιας και δεν τον νοιάζει τόσο η βέλτιστη λύση.

Οι αλγόριθμοι πληροφορημένης αναζήτησης βασίζονται στην ευρετική συνάρτηση, η οποία πρέπει να είναι αποδοτική και λογική ώστε να οδηγεί σε βέλτιστες λύσεις. Παρατήρησα ότι ο Best-First Search έβρισκε λύση γρηγορότερα από τον A*, επειδή εξετάζει μόνο την ευρετική συνάρτηση h(n). Ωστόσο, με τη βελτιστοποίηση της ευρετικής συνάρτησης που χρησιμοποίησα, τόσο ο A* όσο και ο Best-First είχαν καλύτερη απόδοση. Εφόσον μοιραζόντουσαν κοινή συνάρτηση είχαν και ίδιες αποδόσεις στην ταχύτητα και στον αριθμό βημάτων που παρουσιάζουν σαν λύση.