Τεχνητή Νοημοσύνη

Εργαστηριακές Ασκήσεις 2021

Γιαννακόπουλος Δημήτριος ΑΜ:4336
Μυριτζής Ευστράτιος ΑΜ:4444
Παναγιώτου Σωτήριος ΑΜ:4456

Αρχικά, οι ασκήσεις μας έχουν υλοποιηθεί σε Java.

Άσκηση 1(Μέρος 1ο)

Σχετικά με το 1ο μέρος της άσκησης 1, εφαρμόσαμε την UCS και επιπλέον προσθέσαμε κάποιους εκσφαλματωτές οι οποίοι ελέγχουν αν το input που μας δίνει ο χρήστης είναι έγκυρο (πχ αν ο χρήστης δίνει μια λίστα που η sorted μορφή της δεν είναι αύξουσα κατά 1 και δεν αρχίζει με 1, ή αν ο χρήστης δίνει μια λίστα Ν θέσεων στην οποία δεν έχουν τοποθετηθεί φυσικοί αριθμοί από 1 έως Ν και 1 φορά ο καθένας).Έπειτα ακολουθεί η υλοποίηση της UCS στο αρχείο ask1.java .

Άσκηση 1(Μέρος 2ο)

Στο 2ο μέρος της άσκησης, αμέσως μετά την UCS, εκτελείται η A*, ζητάμε από τον χρήστη να μας δώσει μια **έγκυρη** λίστα, κι έπειτα εκτελούμε την A* τυπώνοντας αρχικά την εκτίμηση της βέλτιστης τιμής που υπολογίζουμε με την ευρετική μας. Στην συνέχεια τυπώνουμε το κόστος της διαδρομής (root-goal), τα expansions που κάνει ο αλγόριθμός μας και τέλος την διαδρομή που ακολουθεί η A*.

Πώς υλοποιήσαμε την h(n) και γιατί είναι αποδεκτή

Πώς έγινε η υλοποίηση

Η A* μας είναι ένα αντίγραφο της UCS, στο οποίο έχουμε προσθέσει την ευρετική μας συνάρτηση h(n).
Βρίσκουμε τον αριθμό 1 στην εισαγόμενη λίστα και διατρέχουμε τον πίνακα από το 1 μέχρι την αρχή και μετά διατρέχουμε την υπόλοιπη από το 1 μέχρι το τέλος. Αυτό γίνεται για να δούμε αν ο πίνακας μας είναι κοντά στην sorted μορφή του ή όχι. Στην μεταβλητή point αποθηκεύουμε το ποσό των sorted αριθμών (2,3 = 1 point, 1,2,3 = 2 point) στην λίστα που ελέγχουμε . Την μεταβλητή penalty την χρησιμοποιούμε αργότερα για να προσθέσουμε ένα μικρό ποσό στο h (penalty * 0.05) επειδή παρατηρήσαμε ότι για δύο καταστάσεις με τα ίδια points, πάντα εκείνη με το μεγαλύτερο sorted κομμάτι πριν το 1 θα είναι εκείνη που θα είναι πιο κοντά στην λύση. Μετά εκτελούμε ανάποδα το ίδιο πράγμα, και άλλη μία φορά διατρέχουμε τον πίνακα από την θέση p[0] μέχρι την θέση p[max], για να ψάξουμε για sorted ζεύγη. Τέλος, άμα η δεύτερη ή η τρίτη αναζήτηση του πίνακα φέρνουν μικρότερο h από την αρχική(από το 1 μέχρι την αρχή και μετά μέχρι το τέλος), τότε αντικαθιστούμε το h με το h εκείνων των αναζητήσεων. Λόγω του penalty, το τελικό h που υπολογίζουμε έχει συνήθως δεκαδικό κομμάτι.

Γιατί η ευρετική μας συνάρτηση είναι αποδεκτή;

Πλήθος αριθμών λίστας(και λίστα)	3 (2,3,1)	4 (4,1,3,2)	5 (3,2,4,5,1)	6 (2,1,4,6,5,3)
g(goal node)	2	3	3	4
h(root)	0.57	1.35	2.2	3.23

Πλήθος αριθμών λίστας(και λίστα)	7 (4,2,3,5,7,6,1)	7 (5,4,6,2,3,1,7)	8 (6,1,2,5,7,3,4,8)	9 (6,7,3,8,1,5,4,2,9)
g(goal node)	5	5	6	7
h(root)	3.986	4.036	5.15	5.03

{g(goal node) -> βάθος του goal node
h(root) -> τιμή που μας δίνει η ευρετική ξεκινώντας από το root}

Παρατηρώντας τις τιμές που μας δίνει το g(goal node) και η h(root), διαπιστώνουμε πως για κάθε περίπτωση h(root) <= g(goal node). Έτσι βγάζουμε το συμπέρασμα πως η ευρετική μας συνάρτηση είναι **αποδεκτή**.
(οι τιμές παραπάνω πηγάζουν από τον υπολογισμό της διαδρομής(απόστασης) από το root μέχρι το goal node)

Παρατηρούμε επίσης ότι η A* είναι πολύ πιο γρήγορη από την UCS, καθώς έχει πολύ λιγότερα expansions (για την τελευταία λίστα με τα 9 ψηφία η UCS είχε 88755 expansions έναντι 459 της A*).

Άμα δεν τρέξουμε την UCS, βλέπουμε ότι η A* μπορεί να τρέξει λίστες μεγάλου μεγέθους σε μικρό χρονικό διάστημα, όπως φαίνεται και παρακάτω: