Τεχνητή Νοημοσύνη 1 - Χειμερινό 2020-2021

Εργασια πρώτη

Ροβιθάκης Ιωάννης | sdi1800164

Πρόβλημα 1:

Βλέπε 1115201800164_project1_pacman.pdf

+ σχόλια στα αρχεία πηγαίου κώδικα

Πρόβλημα 2:

Με βάση τη θεωρία από τις διαφάνειες του μαθήματος (2blind1spp.pdf - διαφάνεια 46 - Αποτίμηση IDS), ο μεγαλύτερος αριθμός κόμβων που μπορούν να δημιουργηθούν από τον IDS (worst case scenario) θα είναι:

(d+1)b^0 + db^1 + $(d-1)b^2$ + ... + $2b^{d-1}$ + $1b^d$

Με βάση τον τρόπο που λειτουργεί ο IDS (Για κάθε νέο βάθος d, εφαρμόζεται από την αρχή BFS και οταν καλυφθούν όλοι οι διαθέσιμοι κόμβοι χωρις να βρεθεί το αποτέλεσμα, το βάθος μεγαλώνει και αρχίζει πάλι από την αρχή) και τον παραπάνω τυπο, για goal state σε βάθος g, θα έχουμε:

Πλήθος κόμβων:

Worst case:

(g+1)b^0 + gb^1 + $(g-1)b^2$ + ... + $2b^{g-1}$ + $1b^g$

(Το goal state είναι ο τελευταίος κόμβος του βάθους g που επισκέπτεται ο IDS)

Best case:

(g+1)b^0 + gb^1 + $(g-1)b^2$ + ... + $2b^{g-1}$ + (gb +1)

(Το goal state είναι ο πρώτος κόμβος του βάθους g που επισκέπτεται ο IDS)

Μέχρι βάθος g-1 ισχύουν τα ίδια, για βάθος g επισκέπτεται μόνο gb+1 κόμβους αφού σε κάθε επίπεδο, επεκτείνει b (branch factor) κόμβους οι οποίοι απλά μένουν μέσα στο σύνορο. Τέλος προσθέτουμε στον τελευταίο όρο +1 κόμβο που αντιστοιχεί στην ρίζα

Πρόβλημα 3:

(α) Η ευρετική συνάρτηση που δίνεται από την άσκηση είναι συνεπής και κατά συνέπεια είναι και παραδεκτή (Συμφωνα με την θεωρία διαφάνεια 24 heuristics1spp.pdf).

Απόδειξη:

Πρέπει νδο ισχύει για όλους τους κόμβους η σχέση:

h(n) =< C(n,a,n') + h(n'), για (n' απόγονο του n, και α την κίνηση για να μεταβούμε από τον n στον n'). Με λίγα λόγια, η προσέγγιση του κόστους κάθε νέας κατάστασης συν το κόστος για την μετάβαση στην κατάσταση αυτή, να είναι μεγαλύτερη/ίση από το ευρετικό κόστος της κατάστασης γονέα. Ο μόνος τρόπος που μπορώ να σκεφτώ στην περίπτωση της άσκησης αυτής είναι να δοκιμάσουμε όλους τους συνδυασμούς αφού είναι λιγοι. (Στην πράξη, είναι εύκολο να γράψει κάποιος ένα script που να κάνει τον ανάλογο έλεγχο στην ευρετική του). Πιο αναλυτικά:

h(mail)=26:

>Δεν έχει παιδιά οπότε δεν χρειάζεται να εξεταστεί

h(ts)=23:

23 <= c + h(mail) = 6 + 26 = 32 ισχύει

h(o103)=21:

21 <= c + h(ts) = 8 + 23 = 31 ισχύει

21 <= c + h(b3)= 4 + 17 = 21 ισχύει

21 <= c + h(o109) = 12 + 24 = 36 ισχύει

h(o109)=24:

24 <= c + h(o119) = 16 + 11 = 27 ισχύει

24 <= c + h(o111) = 4 + 27 = 31 ισχύει

h(o111)=27:

>Δεν έχει παιδιά οπότε δεν χρειάζεται να εξεταστεί

h(o119)=11:

11 <= c + h(o123) = 9 + 4 = 13 ισχύει

11 <= c + h(storage) = 7 + 12 = 19 ισχύει

h(o123)=4:

4 <= c + h(o125) = 4 + 6 = 10 ισχύει

4 <= c + h(r123) = 4 + 0 = 4 ισχύει

h(o125)=6:

>Δεν έχει παιδιά οπότε δεν χρειάζεται να εξεταστεί

h(r123)=0:

>Δεν έχει παιδιά οπότε δεν χρειάζεται να εξεταστεί

h(b1)=13:

13 <= c + h(c2) = 3 + 10 = 13 ισχύει

13 <= c + h(b2) = 6 + 15 = 21 ισχύει

h(b2)=15:

15 <= c + h(b4) = 3 + 18 = 21 ισχύει

h(b3)=17:

17 <= c + h(b1) = 4 + 13= 17 ισχύει

17 <= c + h(b4) = 7 + 18 = 25 ισχύει

h(b4)=18:

18 <= c + h(o109) = 7+ 24= 31 ισχύει

h(c1)=6:

6 <= c + h(c3) = 8+ 12= 20 ισχύει

h(c2)=10:

10 <= c + h(c1) = 4+ 6= 10 ισχύει

10 <= c + h(c3) = 6+ 10= 16 ισχύει

h(c3)=12:

>Δεν έχει παιδιά οπότε δεν χρειάζεται να εξεταστεί

h(storage)=12:

>Δεν έχει παιδιά οπότε δεν χρειάζεται να εξεταστεί

Συνεπώς η h(n) είναι συνεπής και άρα είναι και παραδεκτή

(β) Κατά την εκτέλεση τους, οι συγκεκριμένοι αλγόριθμοι αναζήτησης χρησιμοποιούν ως Σύνορο-Fringe κάποια δομή δεδομένων (Στοίβα, Ουρά , Ουρά προτεραιότητας κλπ) για να αποθηκεύουν τις καταστάσεις και να μπορούν να τις εξάγουν με την σωστή σειρά ώστε να επιτύχουν την επίλυση του προβλήματος.

Η σειρά με την οποία οι κόμβοι του προβλήματος μας εξάγονται από το Σύνορο που αντιστοιχεί σε κάθε ένα από τους παρακάτω αλγορίθμους είναι η εξής: (Δεν συμπεριέλαβα την λύση μου βηματικά-αναλυτικά καθώς δεν ζητήθηκε κάτι τετοιο. Αν θέλετε, έχω τα δέντρα σε χαρτί, στείλτε μου email να σας τα στείλω)

i) **BFS**: (Ουρά)

o103 - b3 - o109 - ts - b1 - b4 - mail - o111 - o119 - b2 - c2 - o123 - **r123**

(Έλεγχος goal state κατα την δημιουργία successors όπως και στις διαφάνειες “Αναζήτηση σε Γράφους”)

ii) **DFS**: (Στοίβα)

o103 - b3 - b1 - b2 - b4 - o109 - o111 - o119 - o123 - o125 - **r123**

iii) **IDS**: (DFS για κάθε depth μέχρι στόχο)

>depth = 0

o123

>depth = 1

o123 - b3 - o109 - ts

>depth = 2

o123 - b3 - b1 - b4 - o109 - o111 - o119 - ts - mail

>depth = 3

o123 - b3 - b1 - b2 - c2 - b4 - o109 - ts -mail

>depth = 4

o123 - b3 - b1 - b2 - b4 - c2 - c1 - c3 - o109 - o111 - o119 - o123 - o125 - **r123**

iv) **Greedy** (Best first search) με κριτήριο την h(n):

o103 - b3 – b1 – c2 – c1 – c3 – b2 - b4 – ts – o109 – o119 – o123 - **r123**

v) **A*** με ευρετική την h(n):

o123 - b3 - b1 - c2 - c1 - b2 - b4 - c3 - ts - o109 - o119 - mail - o123 - **r123**

(Είναι εντυπωσιακό πόσο γρήγορα κατευθύνεται ο A* προς την κατεύθυνση του στόχου, και αφού βρεθεί σε αδιέξοδο ελέγχει τις γύρω περιοχές μέχρι να βρει κάποια διέξοδο προς τη λύση)

Πρόβλημα 4:

(α) Ορισμός προβλήματος:

> **Αρχική κατάσταση:**

Το ρομπότ "τοποθετείται" στο node "mail"

> **Χώρος καταστάσεων:**

Στην πράξη, έχουμε να αναπαραστήσουμε τον χώρο εργασίας, το ρομπότ και τα πακέτα. Ένας τρόπος να αναπαρασταθούν πλήρως όλα τα παραπάνω είναι να διατηρούμε αποθηκευμένα για το ρομπότ: την τρέχουσα θέση του στον χώρο, και το αν κρατάει πακέτο ή όχι σε μια δεδομένη στιγμή. Για την αναπαράσταση του χώρου του γραφείου, δηλαδή των δυνατών τοποθεσιών του ρομπότ και των τοποθεσιών και προορισμών των πακέτων, έχουμε ένα γράφο αναπαριστά όλα τα δωμάτια και τους διαδρόμους, καθώς και τις συνδέσεις τους, μαζί με το κόστος μετάβασης απο ένα node σε ένα άλλο. Τέλος, στα nodes πρέπει να αποθηκεύεται αν το node περιέχει ή όχι κάποιο/α πακέτα, καθώς και τον προορισμό του/τους. Με όλα τα παραπάνω, καθώς και με τους κανόνες λογικής του προβλήματος + ότι παραδοχές επιλέξουμε να κάνουμε, μπορούμε να υπολογίσουμε με ακρίβεια το αποτέλεσμα κάθε δυνατής κίνησης/μετάβασης απο state σε state.

> **Συνάρτηση διαδόχων:**

Οι δυνατοί δίαδοχοι ενός state είναι όλα τα states τα οποία αντιστοιχούν σε nodes άμεσα συνδεδεμένα με το node τοποθεσία του state, στον γράφο αναπαράστασης του χώρου. Δηλαδή όλοι οι άμεσα γειτονικοί του κόμβοι.

> **Συνάρτηση κόστους:**

Το κόστος μετάβασης από ένα state σε άλλο, υπολογίζεται ως την απόσταση του ενός κόμβου από τον άλλον. Αυτό μπορεί να γίνει είτε με μοναδιαίο κόστος για κάθε μετακίνηση, είτε με πολλά διαφορετικά κόστη όπως στον γράφο-σχήμα που μας δίνεται για το πρόβλημα 3. (Η επιλογή αποτελεί σχεδιαστική επιλογη ανάλογα με το πρόβλημα, εδώ για λόγους απλότητας επιλέγω να κρατήσω το μοναδιάιο κόστος) (Απο υπόθεση, το ρομποτ καταναλώνει ενέργεια μόνο για μετακίνηση, το να σηκώνει/αφήνει πακέτα δεν έχει κόστος)

> **Συνάρτηση στόχου:**

Έλεγχος αν υπάρχουν nodes στον γράφο τα οποία περιέχουν πακέτα. Αν δεν υπάρχουν πλέον πακέτα διαθέσιμα για παράδοση, και το ρομποτ έχει επιστρέψει στο node που αντιστοιχεί στο mail.

(β) Διαβάζοντας το παραπάνω πρόβλημα, αμέσως έρχεται στο μυαλό το πρόβλημα των τεσσάρων γωνιών που αντιμετωπίσαμε στο Πρόβλημα 1. Μόνη διαφορά είναι ότι αντί για 4 κόμβους-γωνίες στόχου, έχουμε πολλαπλα πακέτα-στόχους. Συνεπώς, με μικρές τροποποιήσεις, μπορούμε να χρησιμοποιήσουμε στο πρόβλημα αυτό την ίδια ευρετική.

Στην πράξη, αρκεί να υπολογίζουμε το άθροισμα των συντομότερων Manhattan αποστάσεων μεταξύ των πακέτων που παραμένουν για παράδωση. Μοναδική τροποποίηση που πρέπει να γίνει έιναι να θεωρήσουμε την διαδρομή από τα δωμάτια προέλευσης των πακέτων, προς τα δωμάτια προορισμούς δεδομένη. Δηλαδή θεωρούμε ότι το αντίστοιχο του περάσματος από μια γωνία, είναι το πέρασμα από "αρχή" και "τέλος" της διαδρομής ενος πακέτου. Συνεπώς, η απόσταση δεν υπολογίζεται μεταξύ κόμβων που περιέχουν πακέτα, αλλά μεταξύ κόμβων προορισμών πακέτων και κόμβων που περιέχουν πακέτα. (Ξεκινώντας από το current/given node, υπολογισμός απόστασης από κοντινότερο πακένο, μετά από τον προορισμό του πακέτου, την απόσταστη μέχρι το επόμενο κοντινότερο πακέτο, και πάει λεγοντας.)

Η παραπάνω ευρετική είναι παραδεκτή καθώς υπολογίζει φυσικές αποστάσεις σε μία βέλτιστη διαδρομή, ενώ ταυτόχρονα δεν λαμβάνει καθόλου υπ όψην της τις αποστάσεις από ένα πακέτο στον προορισμό του, και κατα συνέπεια δεν μπορεί να υπερεκτιμήσει.

Πρόβλημα 5:
(α) BFS και αναζήτηση περιορισμένου βάθους
(β) IDS και αναζήτηση περιορισμένου βάθους
(γ) A* και αναζήτηση περιορισμένου βάθους

Στις περιπτώσεις (α) (β) και (γ), ο αλγόριθμος αμφίδρομης αναζήτησης είναι πλήρης υπό προϋποθέσεις καθως ο BFS, ο IDS και ο A* για να είναι πλήρεις, πρέπει ο παράγοντας διακλάδωσης του γράφου (branch factor) να είναι πεπερασμένος. Πέρα από αυτό, ο αλγόριθμος αναζήτησης περιορισμένου βάθους για να είναι πλήρης πρέπει το βάθος της λύσης να είναι μικρότερο από το βάθος στο οποίο μπορεί να ψάξει. Συνεπώς, αν το βάθος που ψάχνει είναι μικρό, στην πράξη περιορίζεται σε ένα μικρό υποκομμάτι του γράφου αφήνοντας περισσότερη "δουλειά" στον άλλο αλγόριθμο. Ως αποτέλεσμα, το "σημείο συνάντησης" δεν θα είναι πάντα το κέντρο με συνέπεια να αφήνει σε κάποιες περιπτώσεις πολλή περισσότερη δουλειά στον άλλο αλγόριθμο και να χάνεται σε μεγάλο βαθμό το πλεονέκτημα της αμφίδρομης αναζήτησης. Αν λοιπόν ο ένας αλγόριθμος καλύπτει μεγαλύτερη περιοχή από τον άλλο (σε περιπτώσεις που το βάθος της αναζήτησης είναι μικρό) η χωρική/χρονική πολυπλοκότητα παύει να είναι O(b^d/2) και συνεπώς οι μέθοδοι (α), (β) και (γ) δεν είναι βέλτιστες από πλευρά πολυπλοκότητας. Ακόμα η αναζήτηση περιορισμένου πλάτους, χρησιμοποιεί στην πράξη DFS, ο οποίος δεν επιστρέφει πάντα το βέλτιστο μονοπάτι. Συνεπώς, αφού το μονοπάτι που γυνάνε οι παραπάνω μέθοδοι δεν είναι εγγυημένα το βέλτιστο, οι παραπάνω μέθοδοι δεν είναι βέλτιστες.

(δ) A* και A*

Εφόσον στην περίπτωση αυτή έχουμε και στις δύο πλευρές A* που είναι πλήρης (πάλι εαν το branching factor είναι πεπερασμένο), προφανώς και αν υπάρχει λύση, αυτή θα βρεθεί περίπου στη "μέση". Αφού το φόρτο/βάθος αναζήτησης μοιράζεται σχεδόν ίσα, επιτυγχάνεται η βέλτιστη πολυπλόκότητα χώρου και χρόνου. Τέλος, αφού ο A* (σύμφωνα με τη θεωρία) επιστρέφει πάντα βέλτιστη λύση , ο συνδυασμός των δύο A* θα οδηγήσει σε βέλτιστη λύση στο αποτέλεσμα της αμφίδρομης αναζήτησης. Άρα η (δ) είναι βέλτιστη.

(Φυσικά για να λειτουργήσουν σωστά οι A*, πρέπει η ευρετική/ες συναρτηση/εις μου χρησιμοποιούν να είναι συνεπείς , άρα και παραδεκτές)

Ο έλεγχος ότι οι δύο συναρτήσεις θα συναντηθούν (εφόσον η εκφώνηση εγγυάται ότι υπάρχει μοναδική λύση) μπορεί να υλοποιηθέι βέλτιστα όταν ένας από τους δύο αλγορίθμους βγάλει από το σύνορο του έναν κόμβο, ο οποίος να έχει επεκταθεί ήδη από την άλλη συνάρτηση, δηλαδή να βρίσκεται στο explored set της άλλης. (Με βάση το βιβλίο, η συνάντηση εντοπίζεται όταν τα explored sets των δύο αλγορίθμων έχουν κάποιο κοινό στοιχείο). Αν λοιπόν υλοποιήσουμε τα explored με set, μπορούμε να κάνουμε τους ελέγχους σε O(1). Άρα με μονο δύο ελέγχους O(1) σε κάθε επανάληψη (ένας για κάθε συνάρτηση) μπορούμε να ελέγξουμε την συνάντηση τους, εξού και είναι αποδοτική η παραπάνω μέθοδος.