TECHNICAL UNIVERSITY OF CRETE

Τεχνητή Νοημοσύνη
ΠΛΗ 311

$1^{η}$ *Προγραμματιστική Άσκηση*

Καλαϊτζάκης Παναγίωτης

Προπτυχιακός Φοιτητής - 2018030188

pkalaitzakis@isc.tuc.gr

Λαμπρινάκης Γεώργιος

Προπτυχιακός Φοιτητής - 2018030017

glamprinakis@isc.tuc.gr

Τμημα Ηλεκτρολογων Μηχανικων,
Μηχανικων Υπολογιστων

30 Απριλίου 2022

Εισαγωγή

Το πρόβλημα που παρουσιάζεται στην παρούσα άσκηση είναι σχεδόν πανομοιότυπο με τα κλασσικά προβλήματα εύρεσης ελάχιστου μονοπατιού σε άκυκλους γράφους με βάρη. Σε τέτοιου είδους προβλήματα συνήθως υπάρχει ως είσοδος ένας προκαθορισμένος γράφος που αποτελείται από κόμβους που συνδέονται μεταξύ τους με ακμές διαφορετικού βάρους (distance metric) και με σαφώς ορισμένους τους κόμβους αρχής και στόχου.

Η φύση της παρούσας άσκησης ωστόσο, εκ κατασκευής, διαφοροποιεί ορισμένες έννοιες σε σχέση με τα προαναφερθέντα προβλήματα. Συγκεκριμένα, υπάρχει ένα δισδιάστατο επίπεδο (x,y) πάνω στο οποίο βρίσκεται η περιοχή κίνησης - lanelet με τα όριά της (γκρί περιοχή), ο αρχικός κόμβος (node_initial) ως ακριβές σημείο εκκίνησης (x_0, y_0), μια κίτρινη περιοχή στόχος (goal_area) με ορισμένο κέντρο (x_g, y_g) και εμβαδόν, καθώς και ορισμένα εμπόδια (κόκκινες περιοχές) που επίσης έχουν προκαθορισμένα τοπογραφικά χαρακτηριστικά όπως ο στόχος. Οι κόμβοι γείτονες δεν προϋπάρχουν στην μνήμη ως πραγματικοί κόμβοι, αλλά παράγονται μέσω ενός μοντελοποιημένου μηχανισμού κίνησης στο οποίο υπακούει το όχημα. Κατ' ουσίαν, χωρίς να μπούμε σε πολλές λεπτομέρειες, ο μηχανισμός αυτός υπολογίζει και παράγει βάσει της τωρινής ταχύτητας και του προσανατολισμού του οχήματος, τις πιθανές θέσεις γειτονικών κόμβων σε σχέση με τον τρέχοντα.

Στην παρούσα εργασία ασχολούμαστε κυρίως με την υλοποίηση δύο ευρέως διαδεδομένων αλγορίθμων, τον A^* και τον IDA^*, που μπορούν να εφαρμοστούν ώστε να δωθεί βέλτιστη λύση στο πρόβλημα. Οι δύο αυτοί αλγόριθμοι αποτελούν μέλη μιας ευρύτερης οικογένειας αλγορίθμων αναζήτησης που ονομάζονται αλγόριθμοι πληροφορημένης αναζήτησης. Ονομάζονται έτσι καθώς ανά πάσα στιγμή, δηλαδή από οποιαδήποτε κατάσταση-κόμβο, διαθέτουν πληροφορίες που αφορούν την ελάχιστη απόσταση της τωρινής κατάστασης έως κάποιο επιθυμητό κόμβο-στόχο. Φυσικά οι πληροφορίες αυτές αποτελούν μονάχα μια πρόβλεψη για την απόσταση και λαμβάνονται από μια συνάρτηση που ονομάζεται heuristic function. Όσο πιο "κοντά" στις πραγματικές ελάχιστες αποστάσεις είναι οι προβλέψεις της heuristic, τόσο βελτιώνεται η επίδοση των αλγορίθμων. Ο προσδιορισμός της καταλληλότερης ευρετικής συνάρτησης είναι πολύ σημαντικός παράγοντας και εξετάζεται ενδελεχώς στην παρούσα αναφορά.

Επιλογή Heuristic Function

Η καταλληλότερη επιλογή για την ευρετική συνάρτηση και κατά επέκταση τον τρόπο που υπολογίζονται οι εκτιμήσεις των αποστάσεων, είναι άκρως καθοριστική για την επίδοση των αλγορίθμων. Μια ευρετική συνάρτηση είναι:

- Παραδεκτή, όταν δεν υπερεκτιμά το πραγματικό κόστος εύρεσης λύσης. Δηλαδή, μια ευρετική συνάρτηση δίνει πάντα ένα (θετικό) κάτω φράγμα στο πραγματικό κόστος.
- Συνεπής αν, για κάθε ενέργεια με κόστος c, η εκτέλεση αυτής της ενέργειας έχει σαν αποτέλεσμα την μείωση της ευρετικής συνάρτησης κατά το πολύ c.

Σε περίπτωση που η ευρετική συνάρτηση είναι παραδεκτή, οι αλγόριθμοι A^* και IDA^* επιστρέφουν βέλτιστο μονοπάτι αν αυτό υπάρχει. Επομένως, βασική μας προτεραιότητα είναι να μοντελοποιήσουμε την ελάχιστη απόσταση του τρέχοντα κόμβου έως την περιοχή του στόχου. Οι δύο συνηθέστεροι τύποι που μοντελοποιούν την αποσταση δύο σημείων είναι οι εξής:

- Η ευκλείδεια μετρική, είναι η συνάρτηση $d : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ που αντιστοιχεί σε δύο διανύσματα **x, y**, του n-διάστατου διανυσματικού χώρου $\mathbb{R}^n$ στον αριθμό

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{(y_1 - x_1)^2 + (y_2 - x_2)^2 + \cdots + (y_n - x_n)^2} = \sqrt{\sum_{i=1}^{n} (y_i - x_i)^2}.$$

  Κοινώς η ευκλείδεια απόσταση είναι μια ευθεία γραμμή που ενώνει τα δύο σημεία. Δεδομένου οτι η δική μας υλοποίηση δεν λαμβάνει υπόψη τις θέσεις των εμποδίων, πράγματι η ευκλείδεια απόσταση μεταξύ των συντεταγμένων του τρέχοντος κόμβου μέχρι το κέντρο της περιοχής στόχου είναι παραδεκτή ευρετική συνάρτηση.

- Η 1-νορμική απόσταση είναι η συνάρτηση $d : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ που αντιστοιχεί σε δύο διανύσματα **x, y**, του n-διάστατου διανυσματικού χώρου $\mathbb{R}^n$ στον αριθμό:

$$d_1 = \sum_{i=1}^{n} |x_i - y_i|$$

  Η απόσταση αυτή ονομάζεται και μανχάτταν, επειδή είναι η απόσταση που διανύει ένα αυτοκίνητο σε μια πόλη που ορίζεται από οικοδομικά τετράγωνα (εάν δεν υπάρχουν μονόδρομοι).

Ο παράγοντας που μας ενδιαφέρει σε σχέση με την επιλογή ευρετικής, πέραν του αν αυτή είναι παραδεκτή ή όχι, είναι ο αριθμός κόμβων που επεκτείνονται έως ότου τερματιστεί ο εκάστοτε αλγόριθμος πληροφορημένης αναζήτησης. Το γεγονός οτι η ευκλείδεια απόσταση είναι σχεδόν πάντα παραδεκτή μπορεί να καθιστά βέβαιο οτι το μονοπάτι που επιστρέφεται είναι βέλτιστο, όμως δεν σημαίνει οτι και η ευρετική των αλγορίθμων είναι η καλύτερη δυνατή. Αυτό οφείλεται κυρίως στο γεγονός οτι η υλοποίηση της ευρετικής μας, δεν λαμβάνει υπόψη της την χωροταξία των εμποδίων του σεναρίου που επιλύεται, με αποτέλεσμα η απόσταση πολλών κόμβων να υποτιμάται κατά πολύ. Από την άλλη μεριά η χρήση της μανχάτταν ως μετρική απόσταση μπορεί να δώσει περισσότερο insight στον πράκτορα σε τέτοιες περιπτώσεις, οπότε να συμβάλει στην μείωση των κόμβων που επεκτείνονται. Σε περιπτώσεις όμως όπου δεν υπάρχουν εμπόδια στο μονοπάτι, εκ φύσεως η μανχάτταν αποτυγχάνει πλήρως να μοντελοποιήσει την ελάχιστη απόσταση και καταλήγει να είναι μη παραδεκτή. Αν το φανταστούμε γεωμετρικά, είναι απολύτως λογικό αφού το άθροισμα δύο κάθετων πλευρών ενός ορθογώνιου τριγώνου (μανχαττάν) είναι πάντα μεγαλύτερο από την υποτείνουσά του (ευκλείδεια).

Δημιουργία Αλγορίθμων

Αλγόριθμος A^*

Ο αλγόριθμος A^*, είναι ένας άπληστος αλγόριθμος. Χαρακτηρίζεται έτσι διότι από κάθε κατάσταση στην οποία βρίσκεται η αναζήτηση, προσπαθεί να εκτελέσει την κατ' εκτίμηση πιο συμφέρουσα κίνηση προς κάποιον γειτονικό κόμβο. Πιο συγκεκριμένα ορίζουμε:

- g_cost: Πραγματικό κόστος που έχει δαπανηθεί από τον αρχικό κόμβο, έως τον τρέχοντα.
- h_cost: Εκτίμηση κόστους που πρέπει να δαπανηθεί από τον τρέχοντα κόμβο, έως την περιοχή του στόχου.
- f_cost: g_cost + h_cost

Ο συγκεκριμένος αλγόριθμος διαθέτει δυο λίστες στις οποίες διατηρούνται οι κόμβοι που συναντάμε στην πορεία προς τον στόχο. Η μια από αυτές (OPEN list), λειτουργεί ως ουρά προτεραιότητας και περιλαμβάνει τους κόμβους προς επέκταση (κίτρινο χρώμα - fringe). Ο καθορισμός της προτεραιότητας (θέσης στην λίστα) του εκάστοτε κόμβου εξαρτάται κυρίως από την τιμή f_cost του, ενώ στην περίπτωση που υπάρχουν ισότιμα f_cost values, η προτεραιότητα καθορίζεται βάσει της σειράς εισαγωγής του κόμβου στην λίστα. Η δεύτερη λίστα (CLOSED list) περιλαμβάνει τους πλήρως εξερευνημένους κόμβους (γκρί χρώμα). Ως πλήρως εξερευνημένοι κόμβοι νοούνται οι κόμβοι για τους οποίους έχουν παραχθεί όλοι οι κόμβοι παιδία τους και έχουν μάλιστα τοποθετηθεί στην σωστή θέση του fringe.

Ο ψευδοκώδικας που ακολουθήσαμε για την δημιουργία του αλγορίθμου φαίνεται παρακάτω:

```
1  Put node_start in the OPEN list with f(node_start) = h(node_start) (initialization)
2  while the OPEN list is not empty {
3    Take from the open list the node node_current with the lowest
4         f(node_current) = g(node_current) + h(node_current)
5    if node_current is node_goal we have found the solution; break
6    Generate each state node_successor that come after node_current
7    for each node_successor of node_current {
8      Set successor_current_cost = g(node_current) + w(node_current, node_successor)
9      if node_successor is in the OPEN list {
10       if g(node_successor) ≤ successor_current_cost continue (to line 20)
11     } else if node_successor is in the CLOSED list {
12       if g(node_successor) ≤ successor_current_cost continue (to line 20)
13       Move node_successor from the CLOSED list to the OPEN list
14     } else {
15       Add node_successor to the OPEN list
16       Set h(node_successor) to be the heuristic distance to node_goal
17    }
18     Set g(node_successor) = successor_current_cost
19     Set the parent of node_successor to node_current
20   }
21   Add node_current to the CLOSED list
22 }
23 if(node_current != node_goal) exit with error (the OPEN list is empty)
```

Η λογική του αλγόριθμου είναι απλή: Κάθε φορά που πρέπει να επεκταθεί κάποιος κόμβος, επιλέγεται αυτός με το ελάχιστο f_cost. Ο κόμβος αυτός αφαιρείται από την ουρά προτεραιότητας και εξετάζεται αν αποτελεί κόμβος-στόχος. Αν όχι, παράγονται ένα-ένα οι κόμβοι-παιδιά του, από τα οποία για όσα δεν ανιχνεύεται σύγκρουση με κάποιο εμπόδιο, υπολογίζονται τα αντίστοιχα g_cost, f_cost τους και εξετάζεται αν πρέπει να τοποθετηθούν στο fringe (κίτρινο χρώμα). Το evaluation αυτό είναι απαραίτητο στην περίπτωση που η ευρετική είναι παραδεκτή (admissible) αλλά όχι συνεπής (consistent), ώστε να εξασφαλιστεί ότι το μονοπάτι που επιστρέφει ο αλγόριθμος είναι βέλτιστο. Στην ουσία αυτό που ελέγχεται κάθε φορά είναι:

- αν το παιδί υπάρχει ήδη στο fringe με μικρότερο κόστος από αυτό που υπολογίσαμε για το παιδί, τότε το παιδί αγνοείται πλήρως και δεν τοποθετείται σε καμία λίστα.

- αν το παιδί υπάρχει στους εξερευνημένους κόμβους:

  – με μικρότερο κόστος από αυτό που υπολογίσαμε για το παιδί, τότε το παιδί αγνοείται πλήρως και δεν τοποθετείται σε καμία λίστα.

  – με μεγαλύτερο κόστος από αυτό που υπολογίσαμε για το παιδί, τότε το παιδί μεταφέρεται από το group των πλήρως εξερευνημένων, στο fringe.

Φυσικά εάν δεν ισχύει τίποτα από τα παραπάνω, τότε το παιδί προστίθεται στους κόμβους προς επέκταση φυσιολογικά και αναμένει την σειρά του ώστε να εξαχθεί. Μαζί με τον κόμβο παιδί (CostNode) διατηρείται στην μνήμη και το MotionPrimitive που οδήγησε σε αυτό το **ίδιο** παιδί, για λόγους προγραμματιστικής υλοποίησης του goal testing με την pre-built συνάρτηση goal_reached(), στο ακριβές σημείο που ορίζει ο ψευδοκώδικας του αλγορίθμου. Αφού παραχθούν όλα τα παιδιά του τρέχοντος κόμβου, εισάγεται στην λίστα με τους πλήρως εξερευνημένους κόμβους και η διαδικασία επαναλαμβάνεται, είτε έως ότου ο κόμβος που εξερευνάται αυτήν την στιγμή (κόκκινος) είναι κόμβος-στόχος (βρεθεί λύση), ή μέχρι να εξαντληθεί πλήρως το fringe (δεν βρεθεί λύση).

Αλγόριθμος IDA^*

Ο αλγόριθμος IDA^* (Iterative Deepening A^*) είναι κατά βάση πανομοιότυπος με τον αλγόριθμο απληροφόρητης αναζήτησης IDFS (Iterative Deepening Depth First Search), αλλά δανείζεται την ιδέα της ευρετικής (heuristic function) από τον A^*, γεγονός που τον καθιστά και αυτόν αλγόριθμο πληροφορημένης αναζήτησης όπως ο A^*. Πιο συγκεκριμένα, ο αλγόριθμος αυτός δεν διαθέτει fringe, αλλά λειτουργεί αναδρομικά εκτελόντας αναζητήσεις κατά βάθος οι οποίες τερματίζονται σε τιμές cut-off distance bounds, οι οποίες καθορίζονται σε κάθε αναδρομική κλήση του κώδικα. Μάλιστα οι τιμές αυτές παράγονται από την ίδια την ευρετική συνάρτηση που χρησιμοποποιεί ο αλγόριθμος A^*. Το πλεονέκτημά του σε σχέση με τον A^*, έγκειται στην χαμηλή χρήση μνήμης, ακριβώς διότι δεν υπάρχει fringe και αξιοποιείται η τεχνική της αναδρομής, όμως από άποψη χρόνου εκτέλεσης ο αλγόριθμος καθυστερεί ακριβώς για τους ίδιους λόγους (εκτελεί τις ίδιες πράξεις ξανά και ξανά).

Η υλοποίηση του κώδικά μας βασίστηκε στον παρακάτω ψευδοκώδικα:

```
path              current search path (acts like a stack)
node              current node (last node in current path)
g                 the cost to reach current node
f                 estimated cost of the cheapest path (root..node..goal)
h(node)           estimated cost of the cheapest path (node..goal)
cost(node, succ)  step cost function
is_goal(node)     goal test
successors(node)  node expanding function, expand nodes ordered by g + h(node)
ida_star(root)    return either NOT_FOUND or a pair with the best path and its cost

procedure ida_star(root)
    bound := h(root)
    path := [root]
    loop
        t := search(path, 0, bound)
        if t = FOUND then return (path, bound)
        if t = ∞ then return NOT_FOUND
        bound := t
    end loop
end procedure

function search(path, g, bound)
    node := path.last
    f := g + h(node)
    if f > bound then return f
    if is_goal(node) then return FOUND
    min := ∞
    for succ in successors(node) do
        if succ not in path then
            path.push(succ)
            t := search(path, g + cost(node, succ), bound)
            if t = FOUND then return FOUND
            if t < min then min := t
            path.pop()
        end if
    end for
    return min
end function
```

Σχολιασμός & Ανάλυση Αποτελεσμάτων

Αποτελέσματα A^*				
Scenario	Visited Nodes	Estimated Cost	Heuristic Cost (initial node)	Heuristic Type
1	115	31.5	4.5	h = 0
1	38	31.5	34.5	Euclidean
1	21	35.32	34.56	Manhattan
2	478	31.5	4.5	h = 0
2	172	54.0	54.5	Euclidean
2	100	54.0	54.56	Manhattan
3	251	67.5	4.5	h = 0
3	165	67.5	59.5	Euclidean
3	114	67.5	59.76	Manhattan

Αποτελέσματα IDA^*				
Scenario	Visited Nodes	Estimated Cost	Heuristic Cost (initial node)	Heuristic Type
1	39	31.5	34.5	Euclidean
1	52	36	39.56	Manhattan
2	67	54.0	54.5	Euclidean
2	104	54.0	54.56	Manhattan
3	-	-	-	Euclidean
3	-	-	-	Manhattan

Οι πίνακες των αποτελεσμάτων επιβεβαιώνουν τα εξής σημεία της θεωρίας:

- Η ευκλείδεια απόσταση ως ευρετική είναι παραδεκτή συνάρτηση, οπότε ο αλγόριθμος A^* επιστρέφει βέλτιστο μονοπάτι. Στα προβλήματα 2 και 3 βέβαια όπου απαιτείται να στρίψει το όχημα για να αποφύγει τα εμπόδια και να φτάσει στον στόχο του, η ευκλείδεια απόσταση υποτιμά κατά πολύ την ελάχιστη απόσταση με αποτέλεσμα να εξερευνόνται πολλοί κόμβοι που αργότερα θα οδηγήσουν σε dead-end. Μάλιστα για τον ίδιο ακριβώς λόγο ο IDA^*, στο σενάριο 3 ο αλγόριθμος αδυνατεί να βρεί λύση, διότι η ύπαρξη του δεύτερου εμποδίου καθιστά τις εκτιμήσεις του τελείως μη ρεαλιστικές και καταλήγει να παγιδεύεται σε ατέρμονο loop (ξαναρχίζει από την αρχή).

- Η απόσταση μανχάτταν βοηθάει από άποψη αριθμού κόμβων που επεκτείνονται σε περιπτώσεις που υπάρχουν εμπόδια, αλλά σε άλλες περιπτώσεις το μονοπάτι που επιστρέφεται δεν είναι βέλτιστο.

- Εάν θέσουμε την τιμή της ευρετικής ίση με μηδέν, ο αλγόριθμος A_* συμπεριφέρεται ακριβώς όπως ο αλγόριθμος UCS (Uniform Cost Search), ο οποίος εξασφαλισμένα επιστρέφει βέλτιστο μονοπάτι.

Αναφορικά με την κλιμάκωση της ευρετικής συνάρτησης κατά έναν θετικό ακέραιο w, αυτή δεν έχει ιδιαίτερο νόημα όσον αφορά τα δεδομένα σενάρια. Βελτίωση παρατηρείται μόνο στο 3^o σενάριο στην περίπτωση του IDA^*, όπου η κλιμάκωση κατά 1.2 μπορεί να επιφέρει τα επιθυμητά αποτελέσματα. Περισσότερο νόημα θα είχε να μοντελοποιούνταν πιο σύνθετα η ευρετική συνάρτηση και η μοντελοποίηση αυτή να περιελάμβανε με έναν ιδιαίτερα εφάνταστο τρόπο και την χωροταξική οργάνωση των εμποδίων.