Στρατηγικές Πληροφορημένης (Ευρετικής) Αναζήτησης

- Ευρετικές συναρτήσεις
- Αναζήτηση 'πρώτα στον καλύτερο'
- Αναζήτηση A*
- Ιδιότητες των ευρετικών συναρτήσεων

Ευρετικές Συναρτήσεις

Όλες οι στρατηγικές τυφλής αναζήτησης που μελετήσαμε μέχρι τώρα έχουν χρονική πολυπλοκότητα $O(b^d)$ ή κάτι παρόμοιο. Αυτό δεν είναι αποδεκτό σε πραγματικά προβλήματα!

Σε μεγάλους χώρους αναζήτησης, μπορούμε να βρούμε καλύτερους αλγόριθμους χρησιμοποιώντας **ειδική γνώση για το πρόβλημα (problem-specific knowledge)** για να επιταχύνουμε την αναζήτηση.

Αυτή η γνώση κωδικοποιείται με **ευρετικές συναρτήσεις (heuristic functions)** δηλαδή 'rules of thumb' για την επιλογή του επόμενου κόμβου που θα εξεταστεί από κάποιον αλγόριθμο.

Αναζήτηση Πρώτα στον Καλύτερο (Best-First Search)

Ένας αλγόριθμος τυφλής αναζήτησης θα μπορούσε να βελτιωθεί αν γνωρίζαμε **τον καλύτερο κόμβο (ή αυτόν που εκτιμούμε ότι είναι ο καλύτερος)** προς επέκταση.

function BestFirstSearch(*problem*,EvalFn)
returns a solution sequence
  QueuingFn ← a function that orders nodes in
  ascending order of EvalFn
  return TreeSearch(*problem*, *QueuingFn*)

Η συνάρτηση EvalFn ονομάζεται **συνάρτηση αξιολόγησης (evaluation function)**.

Παρατήρηση: Θα μπορούσαμε να χρησιμοποιήσουμε GraphSearch αντί του TreeSearch.

Συναρτήσεις Αξιολόγησης και Ευρετικές Συναρτήσεις

Υπάρχει μια ολόκληρη οικογένεια από αλγόριθμους 'πρώτα στον καλύτερο' με διαφορετικές συναρτήσεις αξιολόγησης.

Ένα σημαντικό στοιχείο πολλών από αυτούς τους αλγόριθμους είναι μια **ευρετική συνάρτηση** h τέτοια ώστε

> $h(n)$ = εκτιμώμενο κόστος της φθηνότερης διαδρομής από τον κόμβο n σε ένα κόμβο στόχου.

Η h μπορεί να είναι οποιαδήποτε συνάρτηση τέτοια ώστε $h(n) = 0$, αν ο n είναι κόμβος στόχου. Αλλά για να βρούμε μια καλή ευρετική συνάρτηση, χρειαζόμαστε **ειδική γνώση για το πρόβλημα μας.**

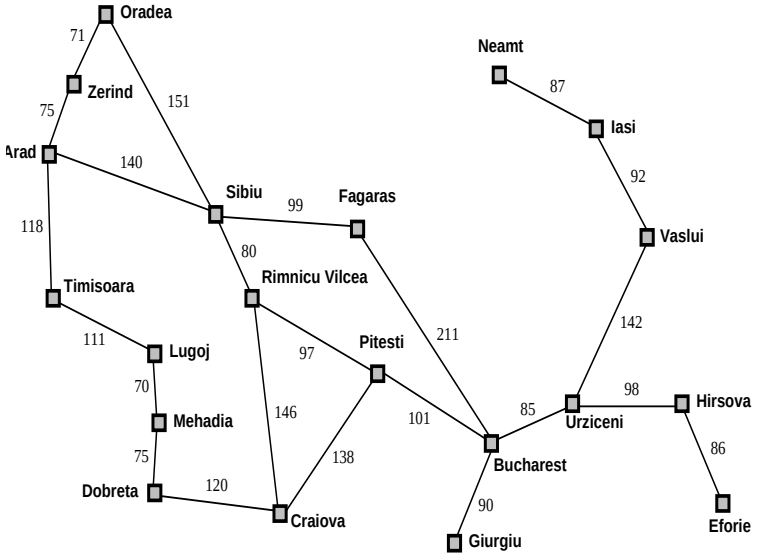
Άπληστη Αναζήτηση Πρώτα στον Καλύτερο

Η **άπληστη αναζήτηση πρώτα στον καλύτερο (greedy best-first search)** επεκτείνει τον κόμβο που είναι **ο πιο κοντινός στο στόχο σύμφωνα με κάποια εκτίμηση**, με το σκεπτικό ότι αυτός μπορεί να μας οδηγήσει σε εύρεση λύσης σύντομα. Συνεπώς, ο αλγόριθμος αυτός αξιολογεί τους κόμβους χρησιμοποιώντας απλώς μια ευρετική συνάρτηση $f(n) = h(n)$.

function GREEDYBESTFIRSTSEARCH($problem$)
returns a solution or failure
  return BESTFIRSTSEARCH($problem, h$)

Ο αλγόριθμος είναι **άπληστος (greedy)** επειδή σε κάθε βήμα προσπαθεί να βρεθεί όσο το δυνατόν πιο κοντά στο στόχο.

Παράδειγμα: Στο Δρόμο για το Βουκουρέστι



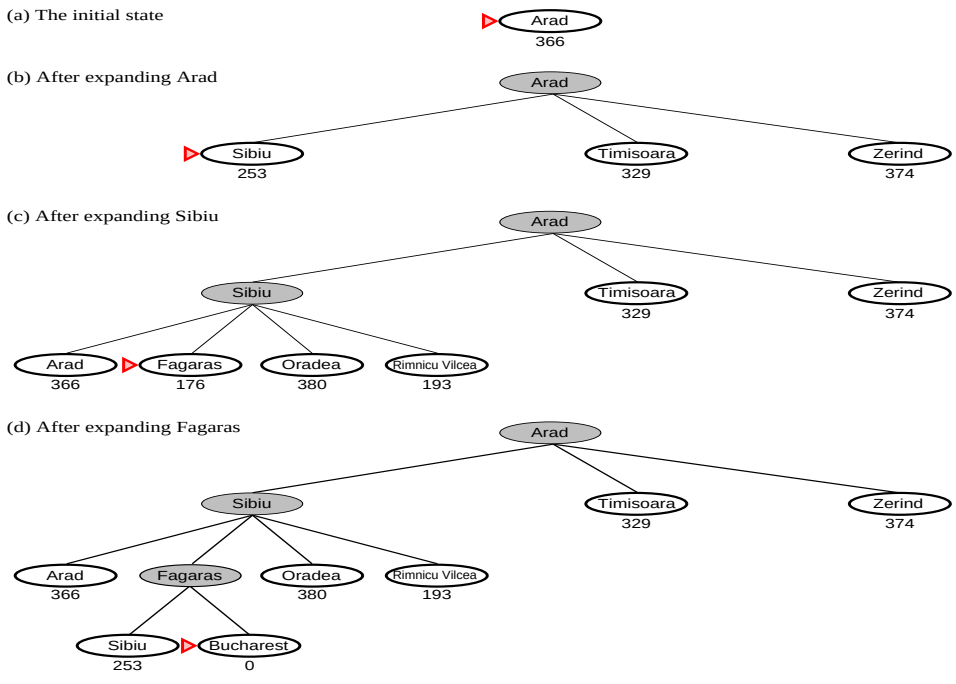
Παράδειγμα: Στο Δρόμο για το Βουκουρέστι

$h_{SLD}(n)$= **ευθύγραμμη απόσταση (straight-line distance)** ανάμεσα στον κόμβο n και τον κόμβο στόχου.

Οι αποστάσεις από το Βουκουρέστι δίδονται παρακάτω:

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Dobreta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Παράδειγμα: Στο Δρόμο για το Βουκουρέστι



Αποτίμηση

- **Πλήρης;** Όχι (έστω το πρόβλημα της μετάβασης από το Iasi στο Fagaras· η αναζήτηση θα παλινδρομεί ανάμεσα στο Iasi και στο Neamt).
- **Χρόνος:** $O(b^m)$ όπου m είναι το μέγιστο βάθος του δένδρου αναζήτησης.
- **Χώρος:** $O(b^m)$
- **Βέλτιστος;** Όχι (Η διαδρομή Arad-Sibiu-Rimnicu Vilcea-Pitesti-Bucharest είναι 418 χιλιόμετρα, ενώ η διαδρομή που περνάει από το Sibiu και το Fagaras 450 χιλιόμετρα.)

Μια καλή επιλογή της h μπορεί να μειώσει το χώρο και το χρόνο σημαντικά.

Άποτίμηση

Το ελάττωμα της άπληστης αναζήτησης πρώτα στο καλύτερο είναι ότι δεν λαμβάνει υπόψη **το κόστος που έχουμε για να φτάσουμε σε ένα κόμβο** n που έχει ελάχιστο $h(n)$.

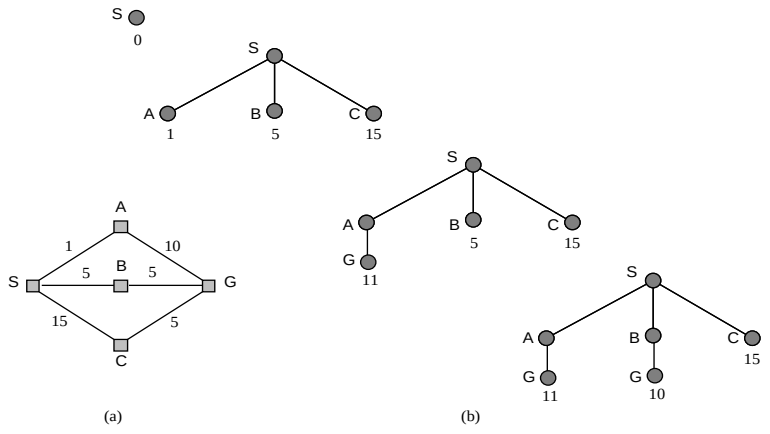
Ο αλγόριθμος που λαμβάνει υπόψη του αυτό το κόστος είναι ο **αλγόριθμος αναζήτησης ομοιόμορφου κόστους** που έχουμε παρουσιάσει.

Θα έχουμε ένα καλύτερο αλγόριθμο αν λάβουμε υπόψη αυτό το κόστος;

Αναζήτηση Ομοιόμορφου Κόστους (Υπενθύμιση)

Ας ξαναθυμηθούμε ότι η αναζήτηση ομοιόμορφου κόστους (UCS) είναι μια παραλλαγή του BFS η οποία επεκτείνει τον κόμβο του συνόρου που έχει το **χαμηλότερο κόστος** (υπολογισμένο με το κόστος μονοπατιού).

UCS - Παράδειγμα



Αποτίμηση της Αναζήτησης Ομοιόμορφου Κόστους

- **Πλήρης;** Ναι, υπό τις συνθήκες που δίνονται παρακάτω.
- **Χρόνος:** $O(b^{\lceil C^*/\epsilon \rceil})$ όπου b είναι ο **παράγοντας διακλάδωσης**, C^* είναι το κόστος της βέλτιστης λύσης και κάθε ενέργεια κοστίζει τουλάχιστον $\epsilon > 0$.
- **Χώρος:** το ίδιο με το χρόνο.
- **Βέλτιστος;** Ναι, υπό τις συνθήκες που δίνονται παρακάτω.

ΥΣ02 Τεχνητή Νοημοσύνη Μ.Κουμπαράκης

Αποτίμηση της Αναζήτηση Ομοιόμορφου Κόστους

Η **πληρότητα** και η **βέλτιστη συμπεριφορά** του UCS ισχύουν υπό τις ακόλουθες συνθήκες:

- Ο παράγοντας διακλάδωσης είναι πεπερασμένος.
- Το κόστος δε μειώνεται ποτέ καθώς προχωράμε σ΄ ένα μονοπάτι, δηλαδή $g(\textsc{Successor}(n)) \geq g(n)$ για κάθε κόμβο n. Αυτή η συνθήκη ισχύει π.χ., όταν κάθε ενέργεια κοστίζει τουλάχιστον $\epsilon > 0$.

Αν ισχύει η δεύτερη συνθήκη, ο UCS **επεκτείνει κόμβους κατά σειρά αυξανόμενου κόστους μονοπατιού. Έτσι, ο πρώτος κόμβος στόχος που επιλέγεται για επέκταση είναι η βέλτιστη λύση.**

Αναζήτηση A*

Άπληστη αναζήτηση πρώτα στο καλύτερο:

- Αναζητά ελαχιστοποιώντας το εκτιμώμενο κόστος $h(n)$ προς το στόχο
- Ούτε βέλτιστος, ούτε πλήρης

Αναζήτηση ομοιόμορφου κόστους:

- Αναζητά ελαχιστοποιώντας το κόστος $g(n)$ της διαδρομής μέχρι τον κόμβο n
- Βέλτιστος, πλήρης

Ο αλγόριθμος αναζήτησης A* συνδυάζει τους παραπάνω αλγόριθμους.

Αναζήτηση A*

Ο αλγόριθμος A* είναι ένας **αλγόριθμος αναζήτησης πρώτα στο καλύτερο** με συνάρτηση αξιολόγησης $f(n) = g(n) + h(n)$.

Στην περίπτωση αυτή $f(n)$ είναι το εκτιμώμενο κόστος της φθηνότερης λύσης **μέσω του κόμβου** n.

function A*SEARCH(*problem*) **returns** a solution or failure
  return BESTFIRSTSEARCH(*problem*, $g + h$)

Πηγαίνοντας στο Βουκουρέστι με τον A*

Δείτε το σχήμα στο αρχείο `astar-progress.ps` ή στο βιβλίο AIMA.

Αποτίμηση της Αναζήτησης A*

Ας υποθέσουμε ότι η αναζήτηση A* χρησιμοποιεί τη συνάρτηση TreeSearch ως κύρια υπορουτίνα της και επιπλέον:

- Η συνάρτηση h επιλέγεται έτσι ώστε **ποτέ να μην υπερεκτιμά το κόστος που έχουμε για να φτάσουμε στο στόχο.** Μια τέτοια συνάρτηση h ονομάζεται **παραδεκτή (admissible).**

  Αν η h είναι παραδεκτή, τότε η $f(n)$ ποτέ δεν υπερεκτιμά το κόστος να φτάσουμε στο στόχο μέσω του n.
- Ο παράγοντας διακλάδωσης b είναι πεπερασμένος.
- Κάθε ενέργεια κοστίζει τουλάχιστον $\delta > 0$.

Αποτίμηση της Αναζήτηση A*

- **Πλήρης;** Ναι.
- **Χρόνος: Εκθετικός**, εκτός αν το σφάλμα της ευρετικής συνάρτησης h δεν αυξάνεται ταχύτερα από το λογάριθμο του πραγματικού κόστους διαδρομής.

  Για τις περισσότερες ευρετικές συναρτήσεις που χρησιμοποιούνται στην πράξη, αυτό το σφάλμα είναι τουλάχιστον ανάλογο του κόστους διαδρομής.

  Αλλά ακόμα κι' αν ο A* χρειάζεται εκθετικό χρόνο, προσφέρει μια **τεράστια βελτίωση** συγκρινόμενος με τους αλγόριθμους τυφλής αναζήτησης που μελετήσαμε.

Αποτίμηση της Αναζήτησης A*

- **Χώρος:** $O(b^m)$ όπου m είναι το μέγιστο βάθος του δένδρου αναζήτησης. Αυτό είναι το κύριο ελάττωμα του A*.

  Έχουν επινοηθεί αρκετοί αλγόριθμοι για να αντιμετωπιστεί αυτό το πρόβλημα: IDA*, RBFS, MA*, SMA*. Δείτε το βιβλίο AIMA για περισσότερες λεπτομέρειες.
- **Βέλτιστος;** Ναι.

Βέλτιστη Συμπεριφορά και Πληρότητα του A*

Πρόταση. Ο αλγόριθμος A* είναι βέλτιστος.

Απόδειξη: Ας υποθέσουμε ότι το κόστος της βέλτιστης λύσης είναι C^* και ένας μη βέλτιστος κόμβος στόχου G_2 εμφανίζεται στο σύνορο. Επειδή ο G_2 είναι μη βέλτιστος και $h(G_2) = 0$, έχουμε:

$$f(G_2) = g(G_2) + h(G_2) = g(G_2) > C^*$$

Έστω τώρα ένας κόμβος n του συνόρου που βρίσκεται πάνω σε μια βέλτιστη διαδρομή λύσης. Επειδή η h δεν υπερεκτιμά το κόστος να φτάσουμε σε λύση, έχουμε:

$$f(n) = g(n) + h(n) \leq C^*$$

Άρα $f(n) \leq C^* < f(G_2)$. Έτσι, ο G_2 **δεν θα επιλεγεί για επέκταση** (θα προτιμηθεί ο n), και ο A* επιστρέφει μια βέλτιστη λύση!

ΥΣ02 Τεχνητή Νοημοσύνη Μ.Κουμπαράκης

Βέλτιστη Συμπεριφορά και Πληρότητα του A*

Η απόδειξη της βέλτιστης συμπεριφοράς **δεν ισχύει** όταν ο A* χρησιμοποιεί τον αλγόριθμο GraphSearch ως κύρια υπορουτίνα.

Αν η βέλτιστη διαδρομή περιέχει μια κατάσταση s και η s παράγεται πρώτα ως μέρος μιας μη βέλτιστης διαδρομής, τότε ο GraphSearch θα απορίψει την s την δεύτερη φορά που θα τη δεί (και άρα **θα απορίψει και την βέλτιστη διαδρομή**).

Για να εξασφαλίσουμε τη βέλτιστη συμπεριφορά του A*, έχουμε δύο επιλογές:

- Να επεκτείνουμε τον αλγόριθμο GraphSearch έτσι ώστε να απορρίπτει **την πιο δαπανηρή** από δύο διαδρομές που βρίσκει προς τον ίδιο κόμβο.
- Να απαιτήσουμε μια επιπλέον ιδιότητα **συνέπειας (consistency)** ή **μονοτονικότητας (monotonicity)** από την ευρετική συνάρτηση h.

Συνεπείς Ευρετικές Συναρτήσεις

Ορισμός. Μια ευρετική συνάρτηση h ονομάζεται **συνεπής (consistent)** αν για όλους τους κόμβους n, n' που είναι τέτοιοι ώστε ο n' είναι απόγονος του n που παράγεται από μια ενέργεια a, έχουμε $h(n) \leq c(n, a, n') + h(n')$.

Η παραπάνω ανίσωση μας θυμίζει την **τριγωνική ανισότητα:** η κάθε πλευρά ενός τριγώνου είναι μικρότερη από το άθροισμα των δύο άλλων πλευρών.

ΥΣ02 Τεχνητή Νοημοσύνη Μ.Κουμπαράκης

Συνεπείς Ευρετικές Συναρτήσεις

Πρόταση. Κάθε συνεπής ευρετική συνάρτηση είναι παραδεκτή.
Απόδειξη;

Η συνέπεια είναι μια πιο αυστηρή συνθήκη από την παραδεκτότητα. Όμως οι περισσότερες παραδεκτές ευρετικές συναρτήσεις που μπορεί κάποιος να σκεφτεί είναι και συνεπείς (π.χ., η h_{SLD})!

Βέλτιστη Συμπεριφορά και Πληρότητα του A*

Πρόταση. Άν η ευρετική συνάρτηση h είναι συνεπής, τότε οι τιμές της f για τους κόμβους που έχουν επεκταθεί από τον A^* κατά μήκος κάθε διαδρομής δεν μειώνονται.

Απόδειξη: Έστω ότι ο n είναι ένας κόμβος και ο n' είναι ένας απόγονός του. Τότε

$$g(n') = g(n) + c(n, a, n')$$

για κάποια ενέργεια a, και έχουμε:

$f(n') = g(n')+h(n') = g(n)+c(n, a, n')+h(n') \geq g(n)+h(n) = f(n).$

ΥΣ02 Τεχνητή Νοημοσύνη Μ.Κουμπαράκης

Βέλτιστη Συμπεριφορά και Πληρότητα του A*

Σύμφωνα με την παραπάνω πρόταση, αν η h είναι συνεπής τότε οι κόμβοι n που επεκτείνονται από τον A* σχηματίζουν μια **μη φθίνουσα ώς προς το $f(n)$ ακολουθία.**

Άρα ο πρώτος κόμβος στόχου G που θα επιλεγεί για επέκταση από τον A* χρησιμοποιώντας GraphSearch είναι **βέλτιστος** γιατί κάθε επόμενος κόμβος G' θα έχει $f(G') \geq f(G)$.

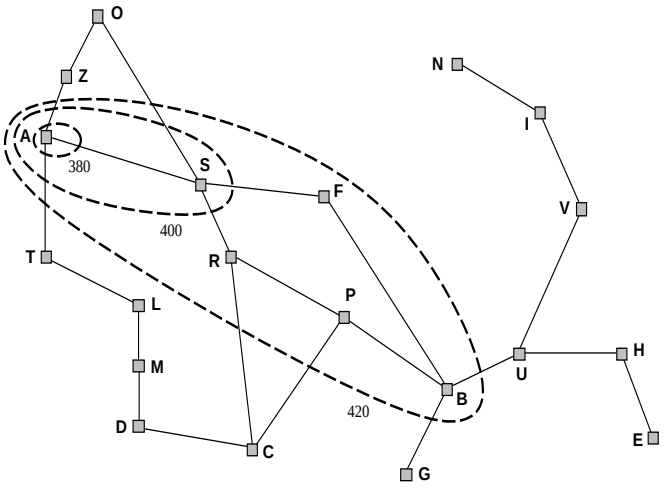
Πρόταση. Αν η ευρετική συνάρτηση h είναι συνεπής τότε ο A* χρησιμοποιώντας GraphSearch είναι βέλτιστος.

Βέλτιστη Συμπεριφορά και Πληρότητα του A*

Με βάση τις προηγούμενες παρατηρήσεις μπορούμε να παραστήσουμε γραφικά τη συμπεριφορά του A* σχεδιάζοντας **ισοϋψείς καμπύλες (contours)** στο χώρο καταστάσεων όπως σχεδιάζουμε ισοϋψείς καμπύλες σ΄ ένα τοπογραφικό χάρτη. Στην περίπτωση μας το 'υψόμετρο' της καμπύλης δίνεται από την συνάρτηση f.

Για $h = 0$ (δηλαδή για τον UCS) οι ισοϋψείς θα είναι κυκλικές γύρω από την αρχική κατάσταση. Με πιο ακριβείς ευρετικούς μηχανισμούς οι ισοϋψείς θα **εστιάζονται πιο στενά γύρω από την βέλτιστη διαδρομή.**

Η Συμπεριφορά του A*



Βέλτιστη Συμπεριφορά και Πληρότητα του A*

Ο αλγόριθμος A* είναι **πλήρης (complete):** καθώς προσθέτουμε ισοϋψείς καμπύλες που αντιστοιχούν σε αυξανόμενη f, θα πρέπει αναπόφευκτα να φθάσουμε σε μια καμπύλη όπου η τιμή f είναι ίση με το κόστος της διαδρομής προς μια κατάσταση στόχου.

Αν C^* είναι το κόστος της βέλτιστης λύσης, ο A* λειτουργεί ως εξής:

- Επεκτείνει **όλους τους κόμβους με** $f(n) < C^*$**.**
- Μπορεί έπειτα να επεκτείνει μερικούς από τους κόμβους ακριβώς πάνω στην 'ισοϋψή καμπύλη στόχου', για την οποία ισχύει $f(n) = C^*$, προτού επιλέξει ένα κόμβο στόχου.

Βέλτιστη συμπεριφορά και Πληρότητα του A*

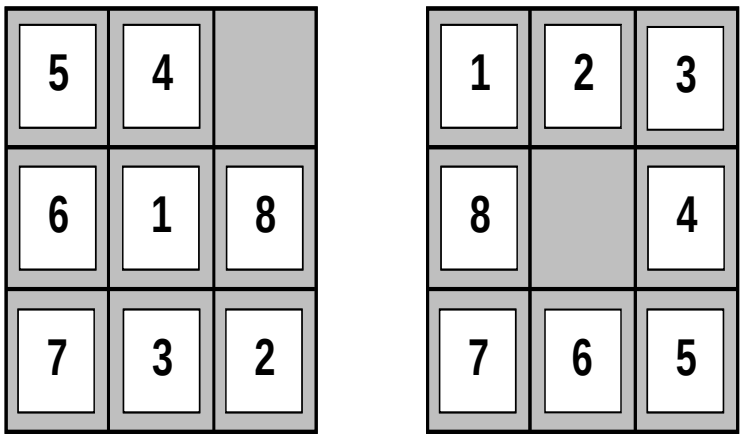
Ο A* **δεν επεκτείνει κανένα κόμβο με** $f(n) > C^*$ όπου C^* είναι το κόστος της βέλτιστης λύσης.

Αυτό ονομάζεται **κλάδεμα (pruning)**: η απαλοιφή καταστάσεων χωρίς να χρειαστεί να εξεταστούν.

Σημαντικό: Δεν υπάρχει άλλος βέλτιστος αλγόριθμος αυτού του τύπου (δηλαδή, αλγόριθμος που να επεκτείνει διαδρομές αναζήτησης ξεκινώντας από τη ρίζα) που να είναι εγγυημένο ότι επεκτείνει λιγότερους κόμβους από τον A*.

Ευρετικές Συναρτήσεις

Ποια είναι μια καλή ευρετική συνάρτηση για το πρόβλημα των 8 πλακιδίων;



Το Πρόβλημα των 8 Πλακιδίων (8-puzzle)

Τυπικός ορισμός:

- **Καταστάσεις**: μια κατάσταση καθορίζεται από τη θέση κάθε πλακιδίου και του κενού.
- **Ενέργειες**: Το κενό κινείται αριστερά, δεξιά, πάνω ή κάτω.
- Η **αρχική κατάσταση** και η **κατάσταση στόχου** δίνονται.
- **Κόστος μονοπατιού**: το μήκος του μονοπατιού.

Ευρετικές Συναρτήσεις

- h_1 = ο αριθμός των πλακιδίων σε λανθασμένες θέσεις
- h_2 = το άθροισμα των αριθμών που δίνουν τις απαιτούμενες οριζόντιες ή κατακόρυφες κινήσεις κάθε πλακιδίου για να πάει από τη θέση του στην θέση που έχει στην κατάσταση στόχου (η **απόσταση Manhattan**).

Οι παραπάνω ευρετικές συναρτήσεις είναι παραδεκτές (admissible). Ποιά είναι η καλύτερη;

Ευρετικές Συναρτήσεις

Ένας τρόπος χαρακτηρισμού της ποιότητας μιας ευρετικής συνάρτησης είναι η εύρεση του **δραστικού παράγοντα διακλάδωσής της (effective branching factor)** b^*.

Αν το πλήθος των κόμβων που επεκτείνονται από τον αλγόριθμο A* για ένα συγκεκριμένο πρόβλημα είναι N, και το βάθος της λύσης είναι d τότε

$$N = 1 + b^* + (b^*)^2 + \cdots + (b^*)^d.$$

Συνήθως, το b^* είναι σταθερό για ένα μεγάλο αριθμό στιγμιοτύπων του προβλήματος. Μια καλά ορισμένη ευρετική μπορεί να έχει τιμή b^* κοντά στο 1.

Σύγκριση A* και IDS

	Search Cost			Effective Branching Factor		
d	IDS	A*(h_1)	A*(h_2)	IDS	A*(h_1)	A*(h_2)
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	364404	227	73	2.78	1.42	1.24
14	3473941	539	113	2.83	1.44	1.23
16	–	1301	211	–	1.45	1.25
18	–	3056	363	–	1.46	1.26
20	–	7276	676	–	1.47	1.27
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

Ευρετικές Συναρτήσεις

Ορισμός. Έστω ευρετικές συναρτήσεις h_1 και h_2. Άν $h_2(n) \geq h_1(n)$ για όλους τους κόμβους n, τότε θα λέμε ότι η h_2 **κυριαρχεί (dominates)** στην h_1 (ή ότι η h_2 είναι πιο **πληροφορημένη (informed)**) από την h_1.

Παράδειγμα: Στο πρόβλημα των 8 πλακιδίων η h_2 κυριαρχεί στην h_1.

Ευρετικές Συναρτήσεις

Θεώρημα: Άν η h_2 κυριαρχεί στην h_1, τότε ο A* που χρησιμοποιεί την h_2 θα επεκτείνει μικρότερο ή ίσο αριθμό κόμβων από τον A* που χρησιμοποιεί την h_1.

Απόδειξη: Είπαμε παραπάνω ότι ο A* επεκτείνει όλους τους κόμβους n για τους οποίους ισχύει $f(n) < C^*$ όπου C^* είναι το κόστος της βέλτιστης λύσης.

Ισοδύναμα, ο A* επεκτείνει όλους τους κόμβους n για τους οποίους ισχύει ότι $h(n) < C^* - g(n)$.

Έστω λοιπόν ένας κόμβος m που επεκτείνεται από τον A* που χρησιμοποιεί την h_2. Τότε $h_2(m) < C^* - g(m)$.

Έχουμε όμως ότι $h_1(m) \leq h_2(m)$ άρα $h_1(m) < C^* - g(m)$. Συνεπώς, ο m επεκτείνεται επίσης από τον A* που χρησιμοποιεί την h_1.

Ευρετικές Συναρτήσεις: Δίδαγμα

Είναι πάντα καλύτερο να χρησιμοποιούμε μια παραδεκτή ευρετική συνάρτηση με **υψηλότερες τιμές**.

Αν έχουμε τις παραδεκτές ευρετικές συναρτήσεις $h_1, \ldots, h_n$ τέτοιες ώστε καμιά να μην κυριαρχεί έναντι των άλλων, τότε μπορούμε να διαλέξουμε $h = max(h_1, \ldots, h_n)$.

Τελική παρατήρηση: Το κόστος του υπολογισμού της ευρετικής συνάρτησης για κάθε κόμβο θα πρέπει να ληφθεί επίσης υπ' όψιν στην επιλογή μας.

Ευρετικές Συναρτήσεις: Πώς τις Επινοούμε;

Φυσικά μπορούμε να μελετήσουμε το δοσμένο πρόβλημα προσεκτικά και να χρησιμοποιήσουμε τη φαντασία μας! Είναι όμως δυνατό να παραχθούν ευρετικές συναρτήσεις **αυτόματα**;

Υπάρχουν διάφορες σχετικές μέθοδοι στην βιβλιογραφία.

Χαλαρωμένες Εκδοχές Ενός Προβλήματος

Είναι δυνατό να βρούμε ευρετικές συναρτήσεις θεωρώντας **χαλαρωμένες εκδοχές (relaxed versions)** του δοθέντος προβλήματος.

Παρατήρηση: Το κόστος μιας βέλτιστης λύσης για ένα χαλαρωμένο πρόβλημα είναι ένας παραδεκτός ευρετικός μηχανισμός για το αρχικό πρόβλημα. **Γιατί;**

Τα χαλαρωμένα προβλήματα μπορούν κάποιες φορές να παραχθούν αυτόματα και έτσι οι ευρετικοί μηχανισμοί μπορούν να ανακαλυφθούν **αυτόματα**!

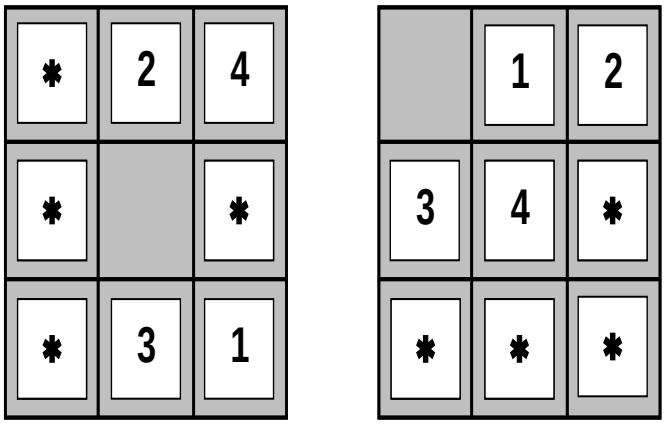
Το κόστος εύρεσης της βέλτιστης λύσης για το χαλαρωμένο πρόβλημα πρέπει να ληφθεί υπόψη.

Υποπροβλήματα και Βάσεις Προτύπων

Μπορούμε να βρούμε παραδεκτές ευρετικές συναρτήσεις για ένα πρόβλημα χρησιμοποιώντας το κόστος λύσης ενός **υποπροβλήματος**.

Το κόστος της βέλτιστης λύσης ενός υποπροβλήματος είναι πάντα κάτω φράγμα του κόστους του πλήρους προβλήματος.

Παράδειγμα



Υποπροβλήματα και Βάσεις Προτύπων

Οι **βάσεις προτύπων (pattern databases)** στηρίζονται στην ιδέα να αποθηκεύονται τα ακριβή κόστη λύσης για κάθε δυνατό στιγμιότυπο υποπροβλήματος (στο παράδειγμα μας, για κάθε δυνατή διάταξη των τεσσάρων πλακιδίων και του κενού).

Για να βρούμε μια παραδεκτή ευρετική συνάρτηση για μια πλήρη κατάσταση που συναντάμε **ψάχνουμε αυτή τη βάση προτύπων για το αντίστοιχο υποπρόβλημα.**

Η ιδέα εδώ είναι ότι το μεγάλο κόστος της κατασκευής της βάσης προτύπων θα ξεπληρώνεται τμηματικά κατά τη λύση πολλών επόμενων στιγμιότυπων του προβλήματος.

Μπορούμε να έχουμε διάφορες βάσεις προτύπων για διαφορετικά υποπροβλήματα και να συνδυάζουμε τις τιμές που μας δίνουν με χρήση της μέγιστης τιμής.

Εκμάθηση Ευρετικών Μηχανισμών

Ένας πράκτορας μπορεί επίσης να **μαθαίνει ευρετικές συναρτήσεις από την εμπειρία του** π.χ., λύνοντας πολλά προβλήματα των 8 πλακιδίων.

Εδώ μπορούν να χρησιμοποιηθούν διάφορες τεχνικές μηχανικής μάθησης. Για περισσότερες λεπτομέρειες δείτε το βιβλίο.

Μελέτη

Κεφάλαιο 4 του βιβλίου ΑΙΜΑ (Ενότητες 4.1 και 4.2).