ΤΕΙ Δυτικής Μακεδονίας
ΤΜΗΜΑ ΔΙΟΙΚΗΣΗΣ ΕΠΙΧΕΙΡΗΣΕΩΝ
2015-2016

Τεχνητή Νοημοσύνη

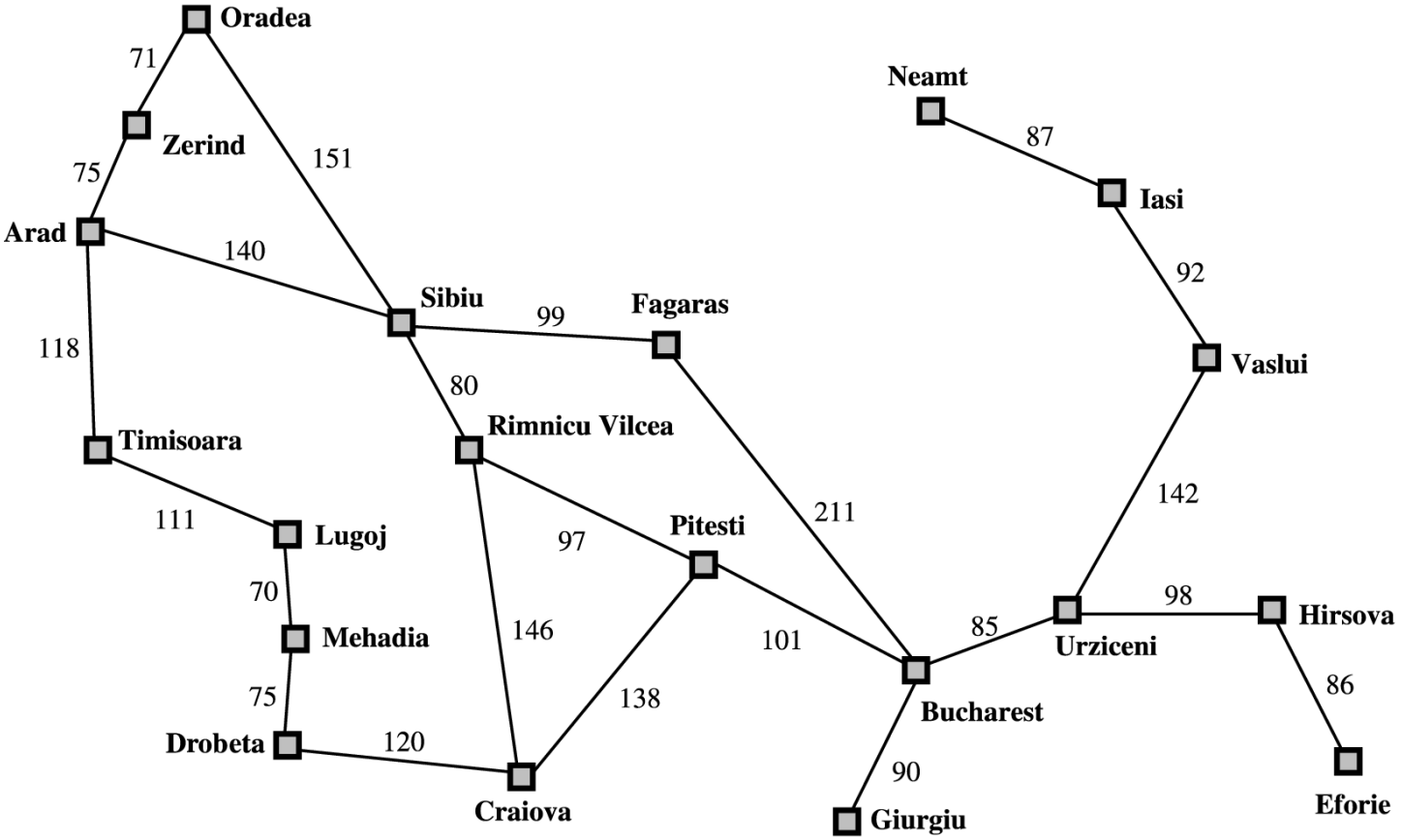
Επίλυση προβλημάτων με αναζήτηση

Διδάσκων: ***Τσίπουρας Μάρκος***
Εκπαιδευτικό Υλικό: ***Τσίπουρας Μάρκος***
http://ai.uom.gr/aima/

Πράκτορες επίλυσης προβλημάτων (1/2)

- Διατύπωση στόχου: Σύνολο καταστάσεων του κόσμου
- Διατύπωση προβλήματος
  - Επιλογή επιπέδου λεπτομέρειας (αφαίρεση)

Πράκτορες επίλυσης προβλημάτων (1/2)



Πράκτορες επίλυσης προβλημάτων (2/2)

- Ένας πράκτορας που έχει στη διάθεσή του πολλές άμεσες επιλογές άγνωστης αξίας μπορεί να αποφασίζει τι να κάνει εξετάζοντας πρώτα διάφορες δυνατές ακολουθίες ενεργειών που οδηγούν σε καταστάσεις γνωστής αξίας και μετά επιλέγοντας την καλύτερη ακολουθία.

- Διατύπωση – Αναζήτηση – Λύση – Εκτέλεση

Απλός πράκτορας επίλυσης προβλημάτων

- **function** Simple-Problem-Solving-Agent(*αντίληψη*) **returns** μια ενέργεια
  inputs: *αντίληψη*, μια αντίληψη
  static: *ακολουθία*, ακολουθία αντιλήψεων, αρχικά κενή
  κατάσταση, περιγραφή της τρέχουσας κατάστασης του κόσμου
  στόχος, ένας στόχος, αρχικά κενός
  πρόβλημα, μια διατύπωση προβλήματος

  κατάσταση ← Update-State(*κατάσταση*, *αντίληψη*)
  if *ακολουθία* είναι κενή **then do**
  στόχος ← Formulate-Goal(*κατάσταση*)
  πρόβλημα ← Formulate-Problem(*κατάσταση*, *στόχος*)
  ακολουθία ← Search(*πρόβλημα*)
  ενέργεια ← First(*ακολουθία*)
  ακολουθία ← Rest(*ακολουθία*)
  return *ενέργεια*

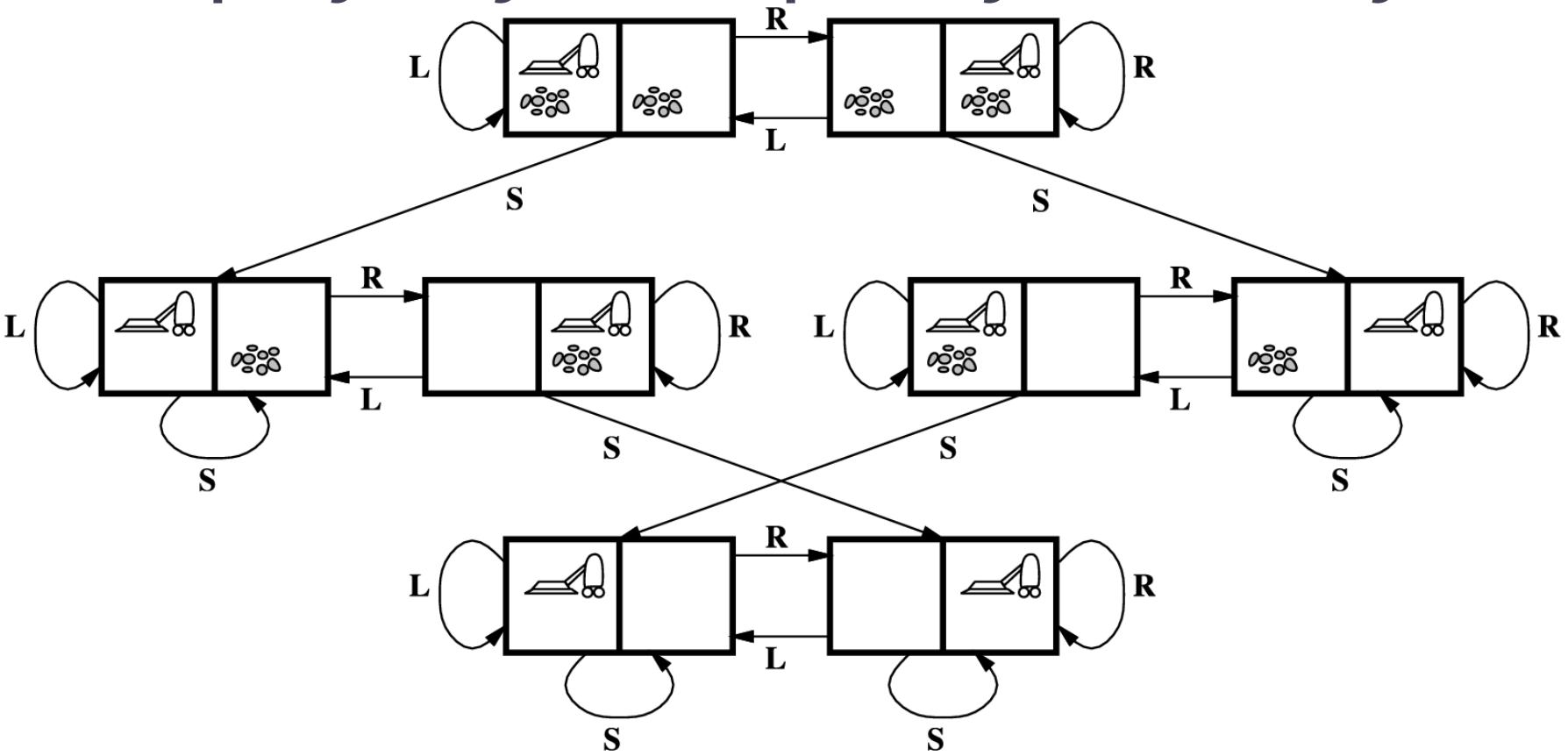
- Παραδοχές: Στατικό, παρατηρήσιμο, διακριτό, αιτιοκρατικό περιβάλλον
- Πράκτορας ανοικτού βρόχου

Προβλήματα και Λύσεις

- Συνιστώσες προβλήματος:
  - Αρχική κατάσταση
    - π.χ. *Εντός*(*Arad*)
  - Ενέργειες, συνάρτηση διαδόχων Successor-Fn(x)
    - π.χ. Successor-Fn(*Arad*) = { ⟨*Μετάβαση*(*Sibiu*), *Εντός*(*Sibiu*)⟩, ⟨*Μετάβαση*(*Timisoara*), *Εντός*(*Timisoara*)⟩, ⟨*Μετάβαση*(*Zerind*), *Εντός*(*Zerind*)⟩ }
    - Χώρος καταστάσεων, αναπαράσταση με γράφημα, διαδρομές
  - Έλεγχος στόχου
    - Ρητή απαρίθμηση καταστάσεων ή περιγραφή ιδιοτήτων
  - Κόστος διαδρομής
    - Κόστος βήματος, $c(x,a,y)$

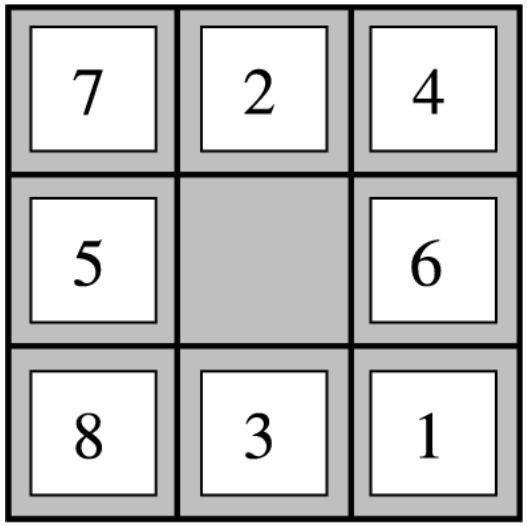
- Λύση: Διαδρομή από αρχική κατάσταση σε κατάσταση στόχου
  - Βέλτιστη λύση

Κόσμος της ηλεκτρικής σκούπας

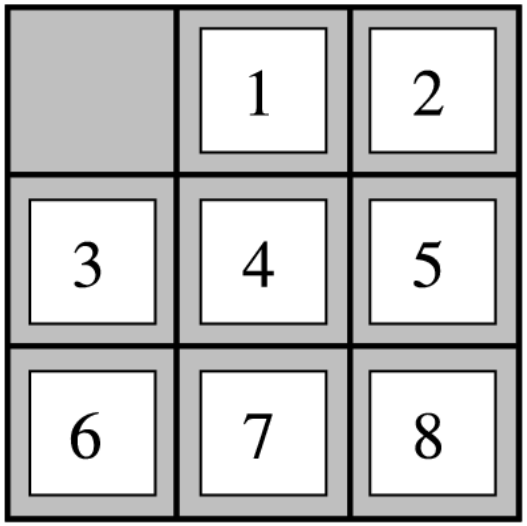


L=Αριστερά, R=Δεξιά, S=Αναρρόφηση

Το παζλ των 8 πλακιδίων



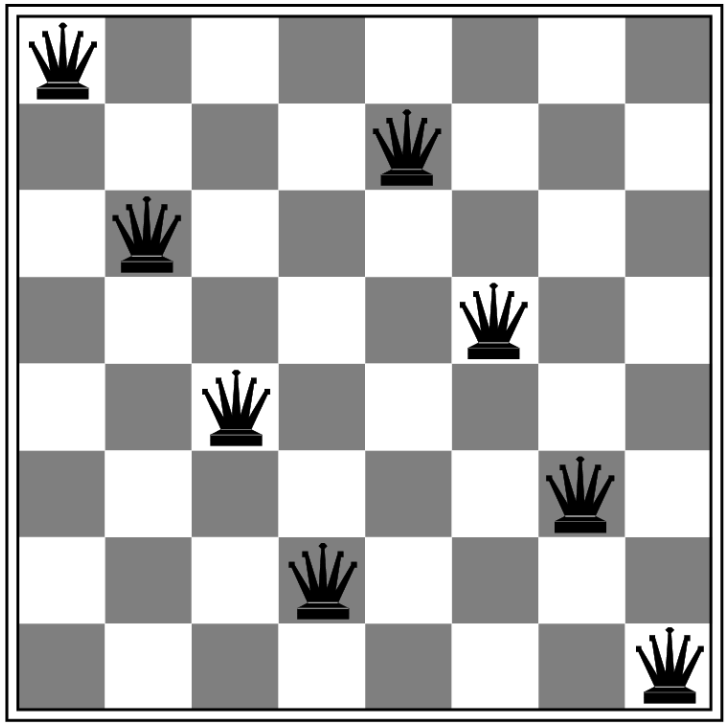
Αρχική κατάσταση



Κατάσταση στόχου

- Παζλ 15 πλακιδίων: Λύνεται εύκολα βέλτιστα
- Παζλ 24 πλακιδίων: Δεν λύνεται βέλτιστα (ακόμη)

Το πρόβλημα των 8 βασιλισσών



- Δύο εναλλακτικές διατυπώσεις:
  - Αυξητική (παραλλαγές)
  - Πλήρεις καταστάσεις

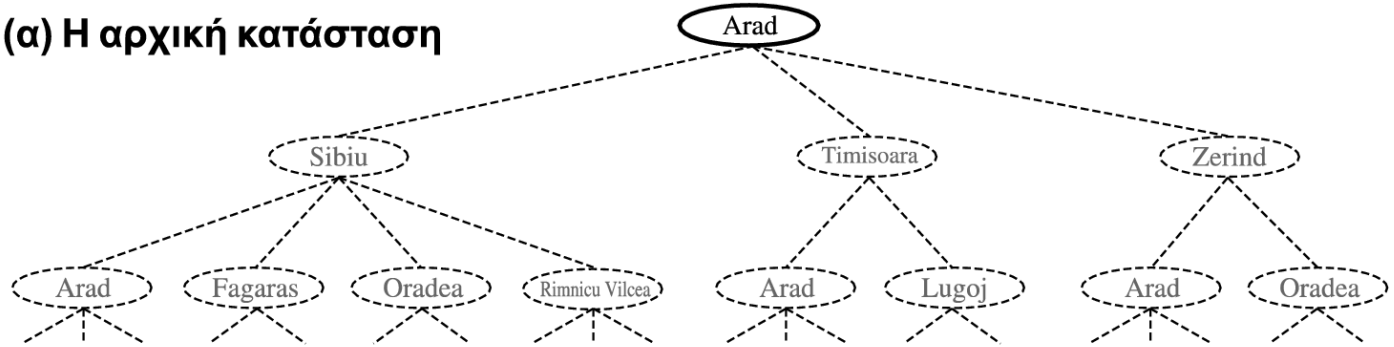
Προβλήματα του πραγματικού κόσμου

- Εύρεση δρομολογίου
- Προβλήματα περιήγησης
  - Πλανόδιος πωλητής
- Διάταξη κυκλωμάτων VLSI
  - Διάταξη κελιών, διάταξη καναλιών
- Πλοήγηση ρομπότ
- Αυτόματη ακολουθία συναρμολόγησης
  - Σχεδίαση πρωτεϊνών
- Αναζήτηση στο διαδίκτυο

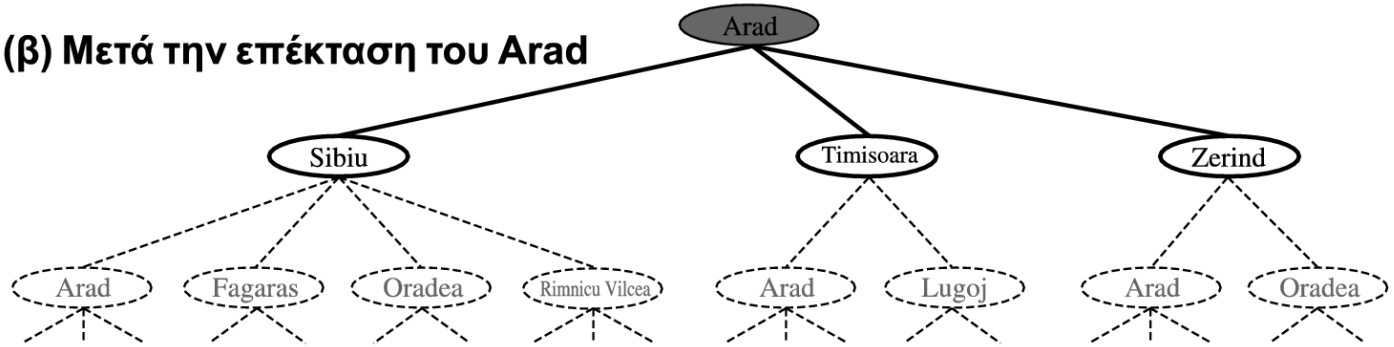
Αναζήτηση λύσεων

Δένδρο αναζήτησης (1/3)

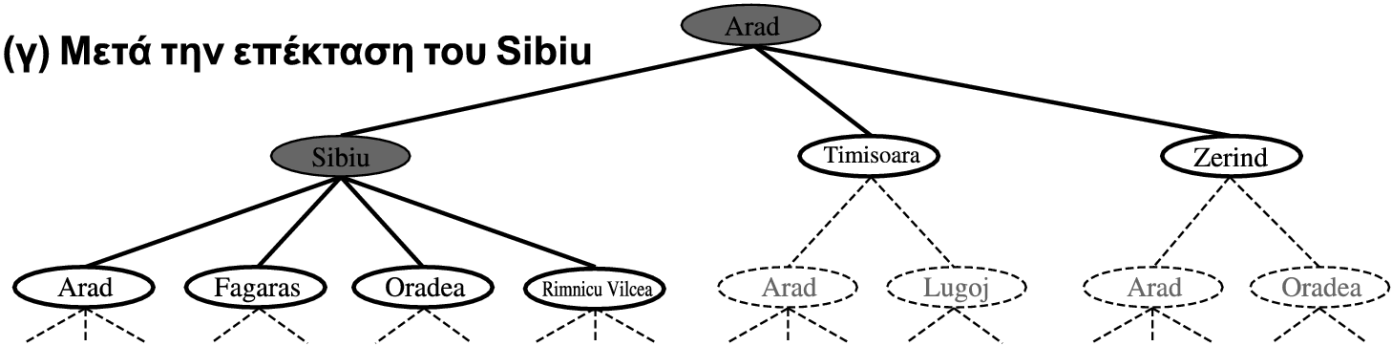
(α) Η αρχική κατάσταση



(β) Μετά την επέκταση του Arad

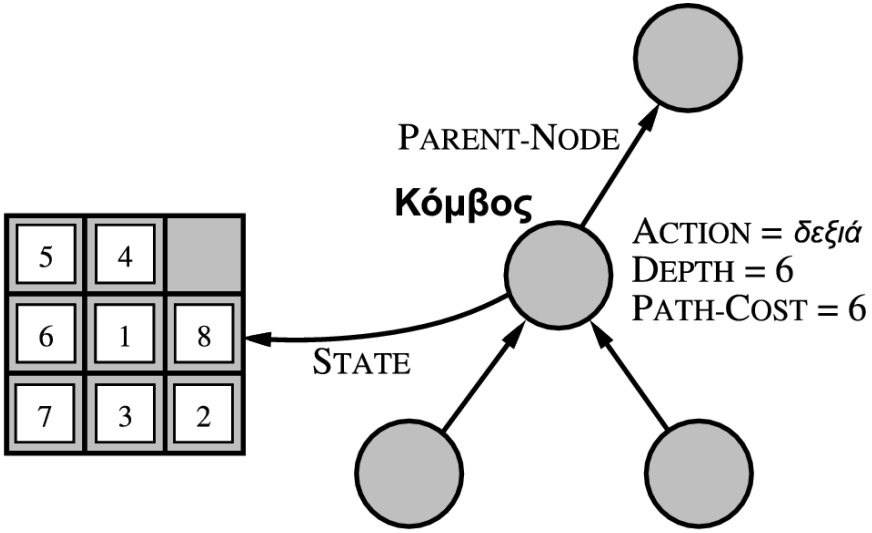


(γ) Μετά την επέκταση του Sibiu



Δένδρο αναζήτησης (2/3)

- Κόμβος αναζήτησης
  - ⟨State, Parent-Node, Action, Path-Cost, Depth⟩



- Επέκταση τρέχουσας κατάστασης, παραγωγή νέων καταστάσεων
- Στρατηγική αναζήτησης

Δένδρο αναζήτησης (3/3)

- Κόμβος φύλλο (leaf node)
- Σύνορο (fringe)
  - Υλοποίηση με ουρά

- Βασικές λειτουργίες στο σύνορο:
  - void Make-Queue(*στοιχείο*, …).
  - bool Empty?(*ουρά*).
  - node First(*ουρά*)
  - node Remove-First(*ουρά*)
  - void Insert(*στοιχείο*, *ουρά*)
  - void Insert-All(*στοιχεία*, *ουρά*)

Άτυπος αλγόριθμος αναζήτησης

- **function** Tree-Search(*πρόβλημα*, *στρατηγική*) **returns** μια λύση ή αποτυχία

  αρχικοποίηση του δένδρου αναζήτησης με χρήση
  της αρχικής κατάστασης του *προβλήματος*

  loop do

  if δεν υπάρχουν υποψήφιοι για επέκταση **then return** αποτυχία

  επιλογή ενός κόμβου-φύλλου για να επεκταθεί, σύμφωνα με τη *στρατηγική*

  if ο κόμβος περιέχει μια κατάσταση στόχου **then**
  return την αντίστοιχη λύση

  else ο κόμβος επεκτείνεται και οι κόμβοι που προκύπτουν
  προστίθενται στο δένδρο αναζήτησης

Τυπικός αλγόριθμος αναζήτησης (1/2)

- **function** Tree-Search(*πρόβλημα*, *σύνορο*) **returns** μια λύση ή αποτυχία

σύνορο ← Insert(Make-Node(Initial-State[*πρόβλημα*]), *σύνορο*)
loop do
if Empty?(*σύνορο*) **then return** αποτυχία
κόμβος ← Remove-First(*σύνορο*)
if Goal-Test[*πρόβλημα*] που εφαρμόστηκε στην State[*κόμβος*] επιτύχει
then return Solution(*κόμβος*)
σύνορο ← Insert-All(Expand(*κόμβος*, *πρόβλημα*), *σύνορο*)

Τυπικός αλγόριθμος αναζήτησης (2/2)

- **function** Expand(*κόμβος*, *πρόβλημα*) **returns** ένα σύνολο κόμβων

διάδοχοι ← κενό σύνολο
for each ⟨*ενέργεια*, *αποτέλεσμα*⟩ **in** Successor-Fn[*πρόβλημα*](State[*κόμβος*]) **do**
s ← νέος κόμβος Node
State[*s*] ← *αποτέλεσμα*
Parent-Node[*s*] ← *κόμβος*
Action[*s*] ← *ενέργεια*
Path-Cost[*s*] ← Path-Cost[*κόμβος*] + Step-Cost(*κόμβος*, *ενέργεια*, *s*)
Depth[*s*] ← Depth[*κόμβος*] + 1
προσθήκη του *s* στο σύνολο *διάδοχοι*
return *διάδοχοι*

Μέτρηση απόδοσης αλγορίθμων αναζήτησης

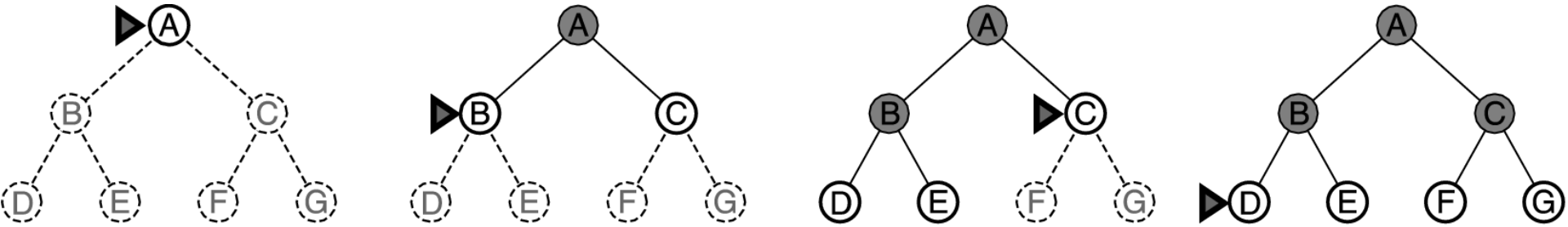
- Πληρότητα (Completeness)
- Βέλτιστη συμπεριφορά (Optimality)
- Χρονική πολυπλοκότητα (Time complexity)
- Χωρική πολυπλοκότητα (Space complexity)

- Μέγεθος προβλήματος:
  - Παράγοντας διακλάδωσης, *b* (branching factor)
  - Βάθος *d* (depth) του πιο ρηχού κόμβου στόχου
  - Μέγιστο μήκος, m, οποιασδήποτε διαδρομής του χώρου καταστάσεων.

- Κόστος αναζήτησης (χρονικό)
- Ολικό κόστος = Κόστος αναζήτησης + κόστος εκτέλεσης πλάνου

Στρατηγικές απληροφόρητης αναζήτησης

Αναζήτηση πρώτα κατά πλάτος
(Breadth-first search)



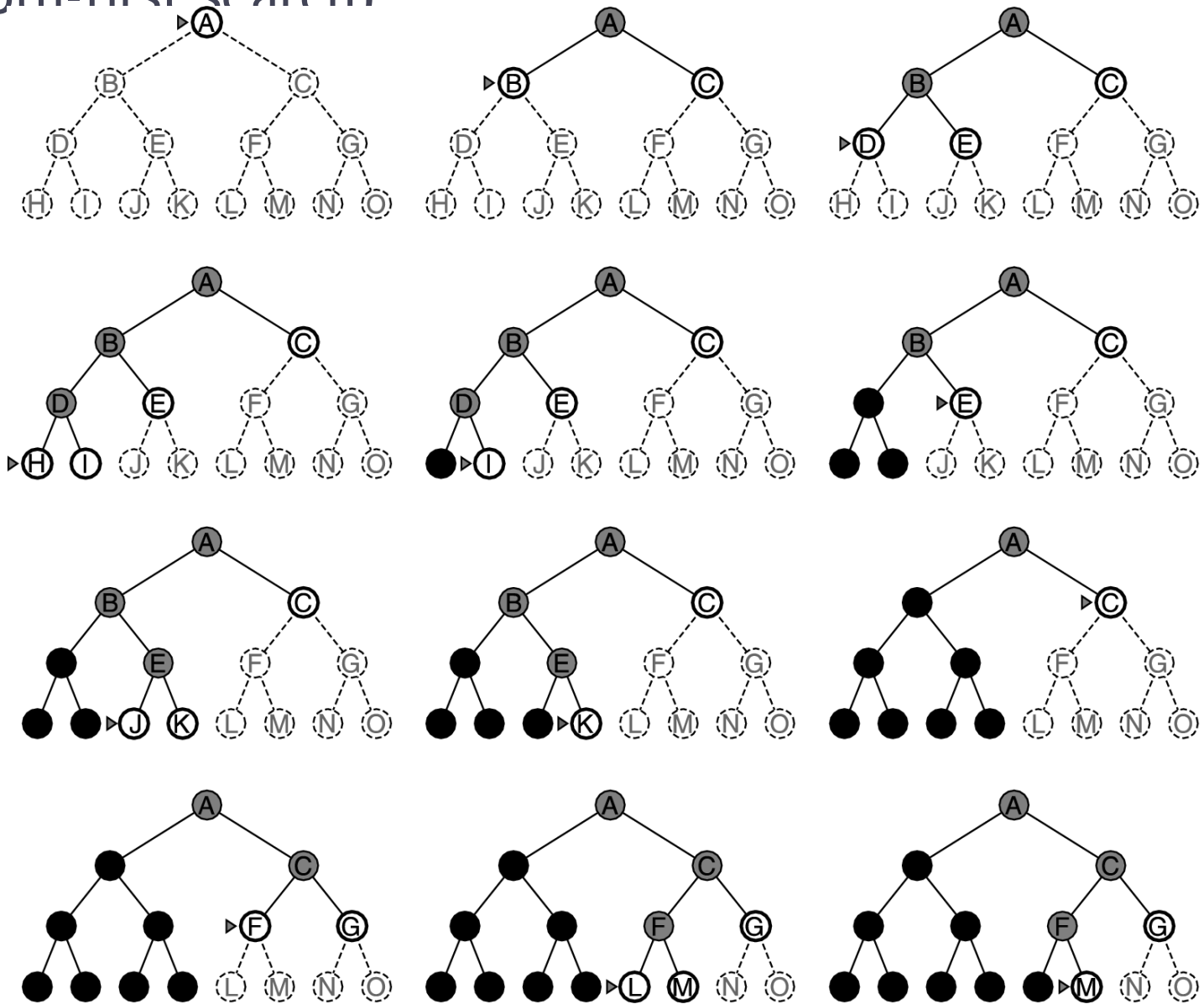
- Πλήρης, βέλτιστος (υπό προϋποθέσεις)
- Χωρική, χρονική πολυπλοκότητα $O(b^{d+1})$
  - π.χ. για b=10, 10.000 κόμβοι/sec, 1000 bytes/κόμβο

Βάθος	Κόμβοι	Χρόνος	Μνήμη
2	1100	0.11 δευτερόλεπτα	1 megabyte
4	111.100	11 δευτερόλεπτα	106 megabytes
6	10^7	19 λεπτά	10 gigabytes
8	10^9	31 ώρες	1 terabyte
10	10^{11}	129 ημέρες	101 terabytes
12	10^{13}	35 χρόνια	10 petabytes
14	10^{15}	3.523 χρόνια	1 exabyte

Αναζήτηση ομοιόμορφου κόστους (Uniform-cost search)

- Παραλλαγή της αναζήτησης πρώτα κατά πλάτος:
  - Επεκτείνει πάντα τον κόμβο με το μικρότερο κόστος διαδρομής
- Προϋπόθεση πληρότητας: Κόστος βήματος > ο
- Εγγυάται ότι θα βρει τη βέλτιστη λύση:
  - Προϋπόθεση: Ο έλεγχος στόχου γίνεται κατά την έξοδο ενός κόμβου από το σύνορο.

Αναζήτηση πρώτα κατά βάθος (1/2)
(Depth-first search)



Αναζήτηση πρώτα κατά βάθος (2/2)
(Depth-first search)

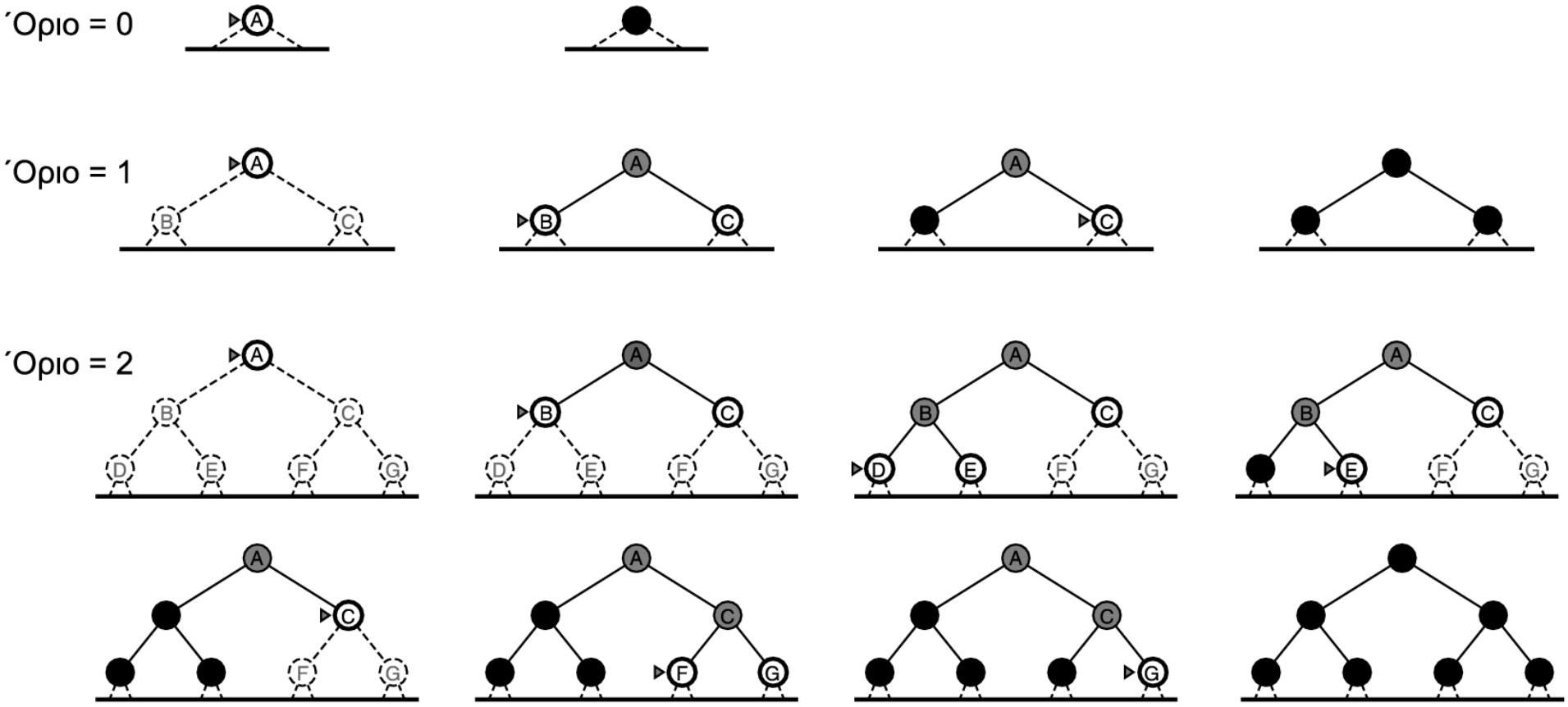
- Ελάχιστες απαιτήσεις σε χώρο: $b \cdot m+1$
- Δεν είναι βέλτιστη
- Δεν είναι πλήρης
- Πολυπλοκότητα χρόνου: $O(b^m)$
  - Προσοχή: $m \geq d$

Αναζήτηση περιορισμένου βάθους (Depth-limited search)

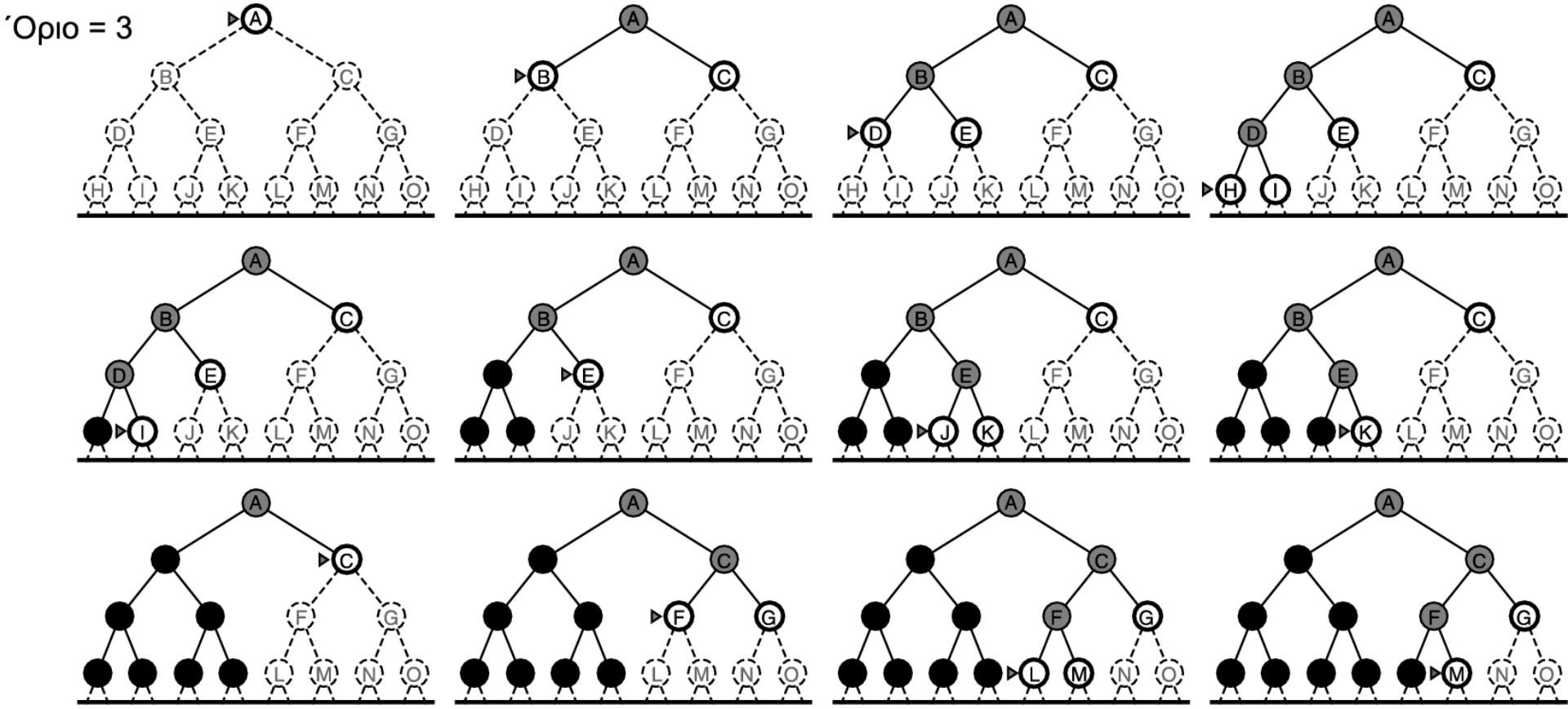
- Τίθεται όριο βάθους L<m.
- Λύνεται το πρόβλημα των άπειρων διαδρομών
- Απώλεια πληρότητας αν L<d.
- Όχι βέλτιστες λύσεις αν L>d
- Χρονική πολυπλοκότητα: $O(b^L)$
- Χωρική πολυπλοκότητα: $O(bL)$

- Επιλογή L με γνώση του προβλήματος (π.χ. διάμετρος του χώρου καταστάσεων)

Επαναληπτική εκβάθυνση (1/3)



Επαναληπτική εκβάθυνση (2/3)

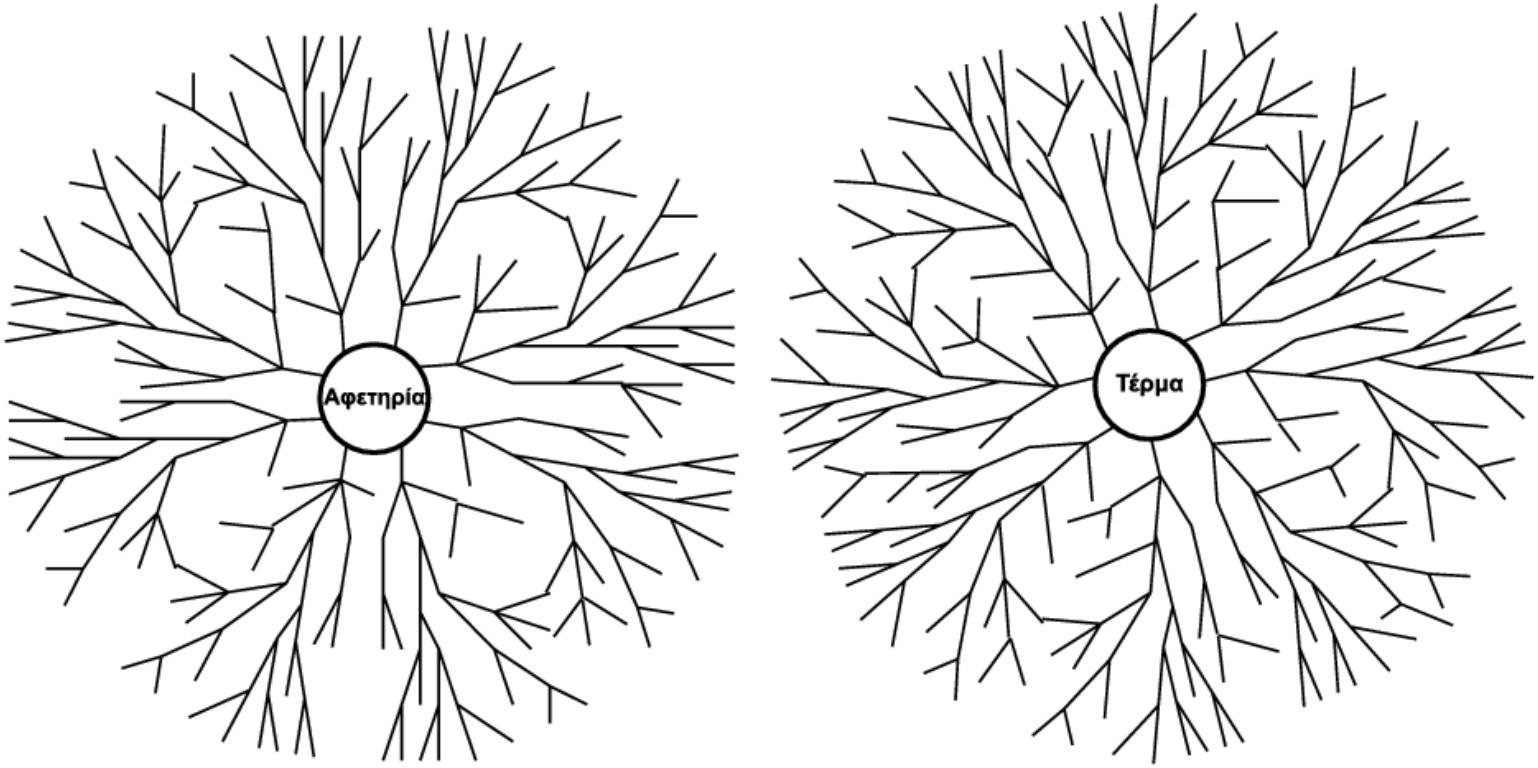


Επαναληπτική εκβάθυνση (3/3)

(Iterative-deepening search)

- Χωρική πολυπλοκότητα: $O(bd)$
- Χρονική πολυπλοκότητα:
  - $N(\text{IDS}) = (d)b + (d-1)b^2 + \ldots + (1)b^d = O(b^d)$!!!!
  - Μικρότερη από τη χωρική πολυπλοκότητα της αναζήτησης πρώτα κατά πλάτος.
- Γενικά, η επαναληπτική εκβάθυνση είναι η προτιμότερη μέθοδος απληροφόρητης αναζήτησης όταν ο χώρος αναζήτησης είναι μεγάλος και το βάθος της λύσης δεν είναι γνωστό.
- Αναζήτηση επαναληπτικής επιμήκυνσης (με κόστη αντί για βήματα)

Αμφίδρομη αναζήτηση (1/2)



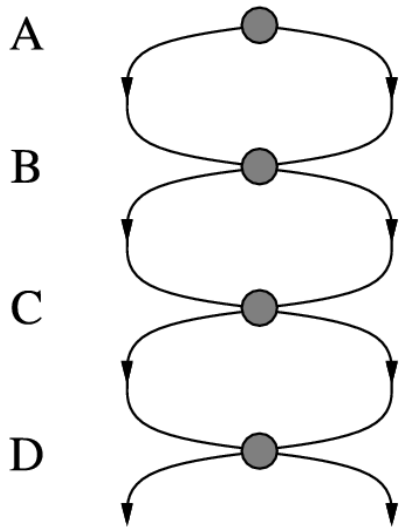
- $b^{d/2}+b^{d/2} << b^{d}$

Αμφίδρομη αναζήτηση (2/2)

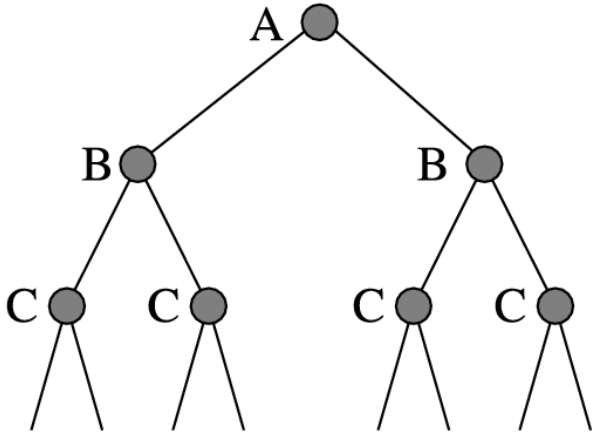
- Χρονική πολυπλοκότητα: $O(b^{d/2})$
- Χωρική πολυπλοκότητα: $O(b^{d/2})$ (μειονέκτημα)

- Προβλήματα:
  - Εύρεση προκατόχων καταστάσεων
  - Έλλειψη καλά καθορισμένων στόχων (π.χ. σκάκι)

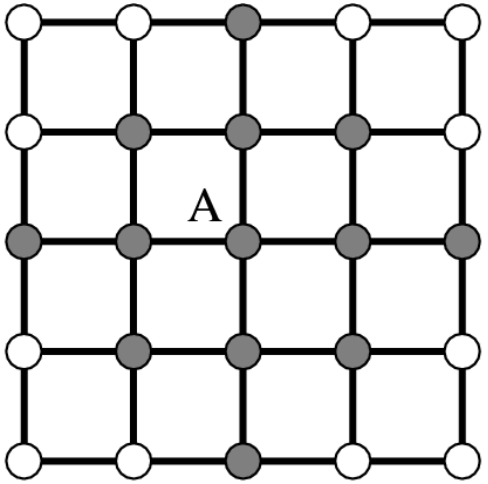
Αποφυγή επαναλαμβανόμενων καταστάσεων (1/2)



(α)



(β)



(γ)

Αποφυγή επαναλαμβανόμενων καταστάσεων (2/2)

- Οι αλγόριθμοι που ξεχνούν την ιστορία τους είναι καταδικασμένοι
  να την επαναλαμβάνουν !

- Έλεγχος μόνο τρέχουσας διαδρομής
  - π.χ. αλγόριθμος πρώτα κατά βάθος

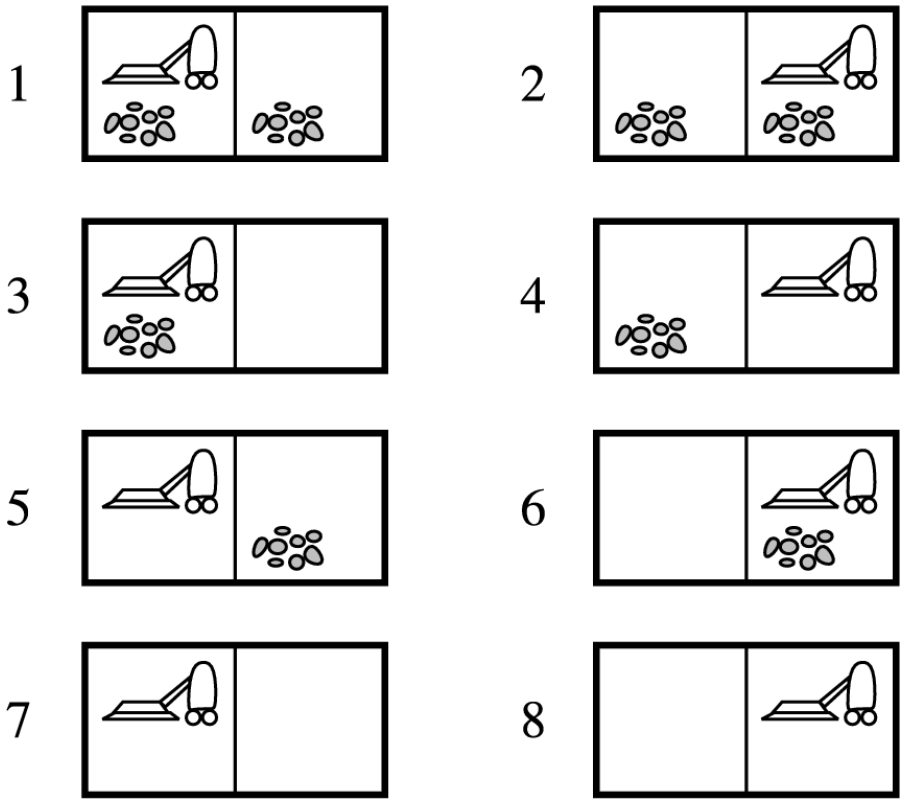
- Έλεγχος όλων των καταστάσεων
  - Κλειστή λίστα
    - (σύνορο = ανοικτή λίστα)

- Πρόβλημα: Ποια από τις δύο καταστάσεις κρατάμε;

Αναζήτηση με μερική πληροφόρηση

Κατηγορίες προβλημάτων

1. Προβλήματα χωρίς αισθητήρες (sensorless problems) ή σύμμορφα προβλήματα (conformant problems)
2. Προβλήματα ενδεχομένων (contingency problems)
3. Προβλήματα εξερεύνησης (exploration problems)

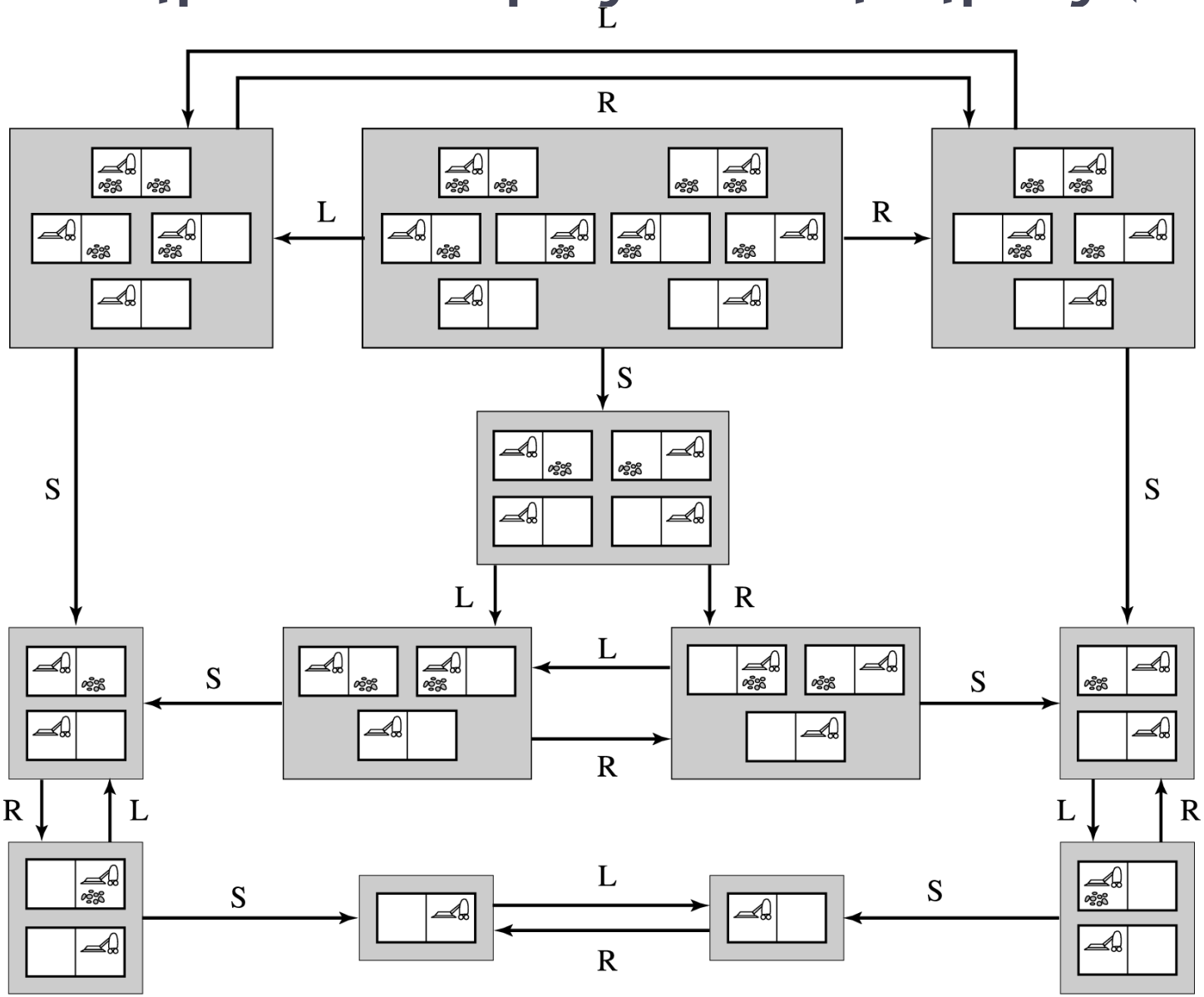


Προβλήματα χωρίς αισθητήρες (1/2)

- Κατάσταση πεποίθησης
  - Αρχική ➔ {1,2,3,4,5,6,7,8}
- Εξαναγκασμός
  - [Δεξιά] ➔ {2,4,6,8}
  - [Δεξιά, Αναρρόφηση] ➔ {4,8}
  - [Δεξιά, Αναρρόφηση, Αριστερά] ➔ {3,7}
  - [Δεξιά, Αναρρόφηση, Αριστερά, Αναρρόφηση] ➔ {7}
- Χώρος καταστάσεων πεποίθησης: Δυναμοσύνολο του χώρου καταστάσεων.
  - Δεν είναι πάντα όλες προσπελάσιμες.
- Η προσέγγιση καλύπτει και μη αιτιοκρατικές ενέργειες.

- Δεν έχουν πάντα λύση αυτά τα προβλήματα.

Προβλήματα χωρίς αισθητήρες (2/2)



Προβλήματα ενδεχομένων

- Όταν έχουμε:
  - Μη πλήρως παρατηρήσιμο περιβάλλον, ή/και
  - Στοχαστικές ενέργειες
- Ενέργειες αίσθησης χρησιμοποιούνται για συλλογή πληροφοριών κατά την εκτέλεση της λύσης.
- Π.χ. Έστω ο κόσμος της σκούπας, όπου:
  - ο πράκτορας έχει τοπικό αισθητήρα σκόνης
  - η ενέργεια Αναρρόφηση δεν πετυχαίνει πάντα.
- Ο πράκτορας ξεκινά με την αντίληψη [Αριστερό, Σκονισμένο].
- Λύση:
  - [*Αναρρόφηση*, *Δεξιά*, **if** [*Δεξιό*, *Σκονισμένο*] **then** *Αναρρόφηση*]
- …και νέα αναζήτηση μέχρι να επιτευχθεί ο στόχος…
- Δεν είναι απαραίτητο από να έχουμε πλήρη λύση εξαρχής.
  - Διαπλοκή (interleaving) αναζήτησης και εκτέλεσης.