Artificial Intelligence

Lecture 3

Supta Richard Philip

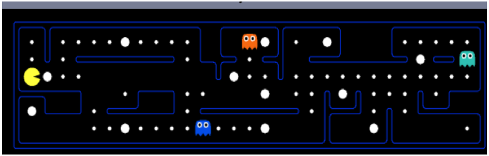
Department of CSE
City University

City University, April 2019

Table of Contents

Table of Contents

What is in State Space?

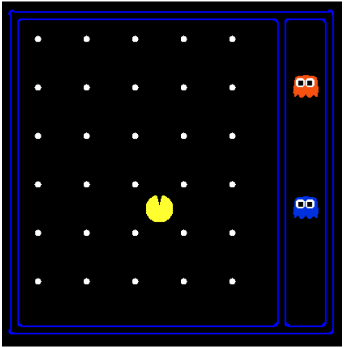


- A world state includes every details of the environment.
- A search state includes only details needed for planning.

Problem: Pathing
States: x,y locations
Actions: NSEW moves
Successor: update location
Goal: is (x,y) End?

Problem: Eat-all-dots
States: (x,y), dot booleans
Actions: NSEW moves
Successor: update location and dot boolean
Goal: dots all false?

State Space Sizes?

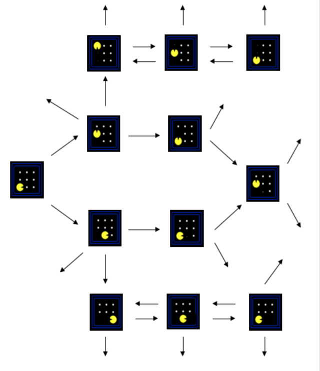


Pacman positions: 10 x 12 = 120
Pacman facing: up, down, left, right
Food Count: 30
Ghost positions: 12

World State: ?
$120^*(2^{30}) * (12^2) * 4$
States for Pathing: 120
States for eat-all-dots: $120 * (2^{30})$

State Space Graphs

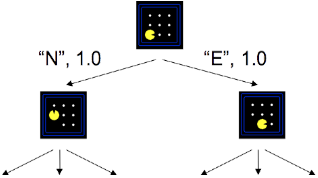
- Each node is a state
- The successor function is represented by arcs
- Edges may be labeled with costs
- We can rarely build this graph in memory (so we don't)



Search Trees

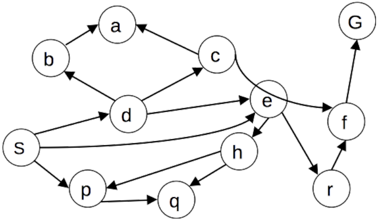
A search tree:

- Start state at the root node
- Children correspond to successors
- Nodes contain states, correspond to PLANS to those states
- Edges are labeled with actions and costs
- For most problems, we can never actually build the whole tree



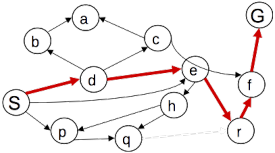
Example 1: Search Tree from state space graph

State Graph:

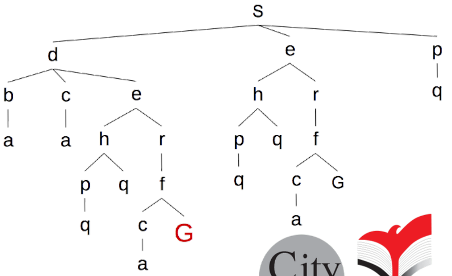


What is the search tree?

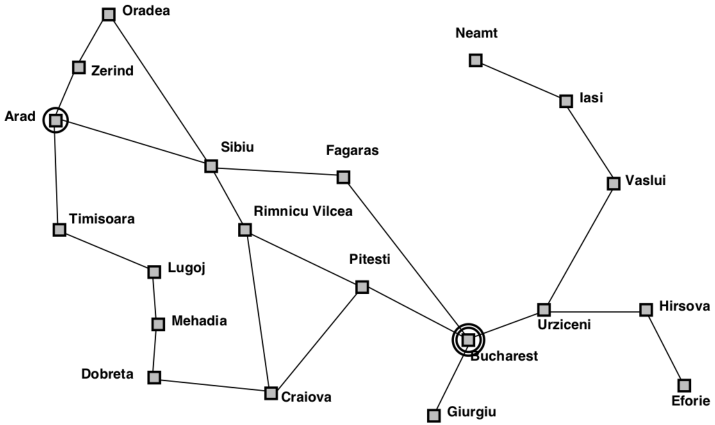
Path:



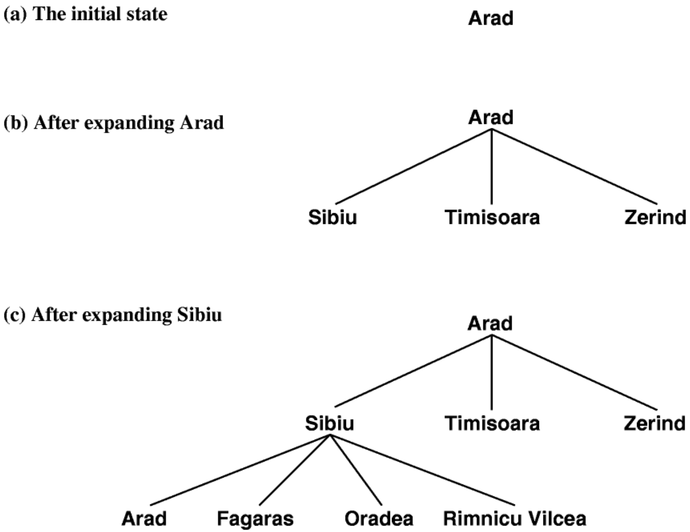
Search Tree:



Example 2: Search Tree from State Space Graph



Example 2: Search Tree from State Space Graph



General Search Algorithm

```
function GENERAL-SEARCH(problem, strategy) returns a solution, or failure
   initialize the search tree using the initial state of problem
   loop do
      if there are no candidates for expansion then return failure
      choose a leaf node for expansion according to strategy
      if the node contains a goal state then return the corresponding solution
      else expand the node and add the resulting nodes to the search tree
   end
```

```
function GENERAL-SEARCH(problem, QUEUING-FN) returns a solution, or failure

   nodes ← MAKE-QUEUE(MAKE-NODE(INITIAL-STATE[problem]))
   loop do
      if nodes is empty then return failure
      node ← REMOVE-FRONT(nodes)
      if GOAL-TEST[problem] applied to STATE(node) succeeds then return node
      nodes ← QUEUING-FN(nodes, EXPAND(node, OPERATORS[problem]))
   end
```

Search strategies

A strategy is defined by picking the *order of node expansion.* Strategies are evaluated along the following dimensions:

- **completeness:** is the strategy guaranteed to find a solution when there is one?
- **time complexity:** how long does it take to find a solution?
- **space complexity:** how much memory is required to perform the search?
- **optimality:** does the search strategy find the highest quality solution when there are multiple solutions?

Time and space complexity are measured in terms of
b—maximum branching factor of the search tree
d—depth of the least-cost solution
m—maximum depth of the state space (may be ∞)

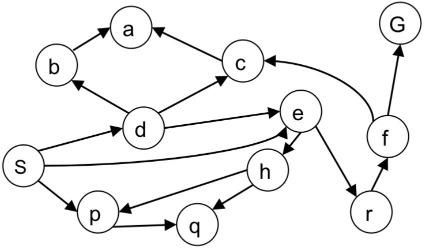
Uninformed Search

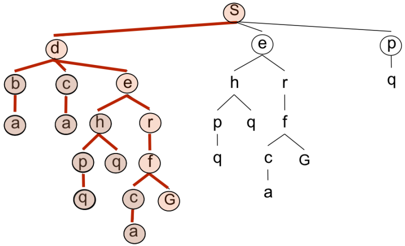
Uninformed strategies use only the information available in the problem definition.

- Depth-first search
- Depth-limited search
- Iterative deepening search
- Breadth-first search
- Uniform-cost search

Depth-first search

Strategy: expand deepest node first
Implementation: Fringe is a LIFO queue (a stack)
Expansion order: (d,b,a,c,a,e,h,p,q,q,r,f,c,a,G)





Properties of Depth-first search

Complete: No. fails in infinite-depth spaces, spaces with loops
Modify to avoid repeated states along path
$\Rightarrow$ complete in finite spaces
Optimal: No
Time: $O(b^m)$: terrible if m is much larger than d
but if solutions are dense, may be much faster than breadth-first
Space: $O(bm)$, i.e., linear space!

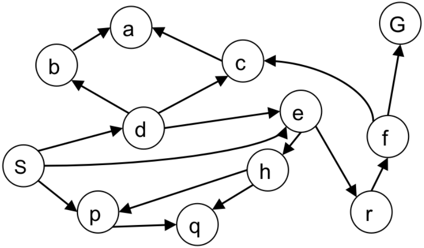
Depth-limited search

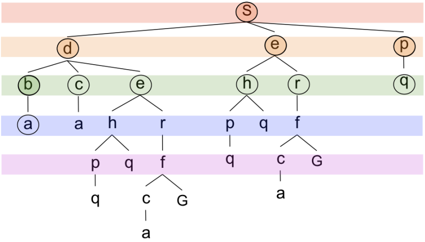
depth-first search with depth limit l
Implementation: Nodes at depth l have no successors

Iterative deepening search

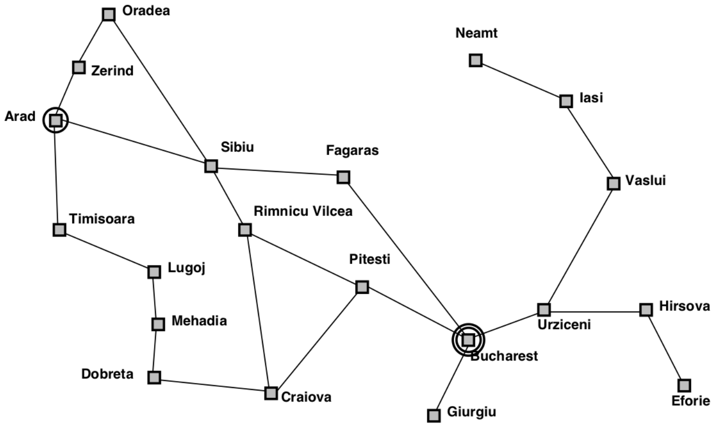
Breadth-first search

Strategy: expand shallowest node first
Implementation: Fringe is a FIFO queue
Expansion order: (S,d,e,p,b,c,e,h,r,q,a,a,h,r,p,q,f,p,q,f,q,c,G)



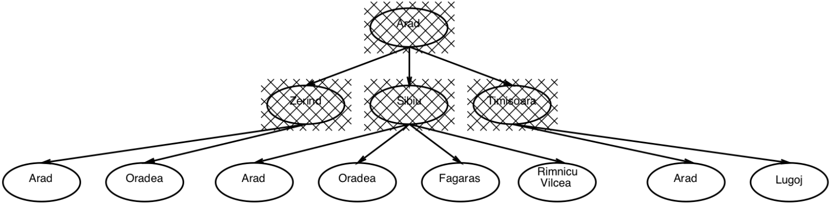


Breadth-first search



Properties of Breadth-first search

Complete: Guaranteed to find a solution if one exists?
Yes (if b is finite)
Optimal: Guaranteed to find the least cost path?
Yes (if cost = 1 per step); not optimal in general
Time: $1 + b + b^2 + b^3 + \ldots + b^d = O(b^d)$, i.e., exponential in d
Space: $O(b^d)$ (keeps every node in memory)
Space is the big problem; can easily generate nodes at 1MB/sec so 24hrs = 86GB.



Uniform cost search

Table of Contents

References

Stuart Russell and Peter Norvig. 2009. Artificial Intelligence: A Modern Approach (3rd ed.). Prentice Hall Press, Upper Saddle River, NJ, USA.

http://www.massey.ac.nz/ mjjohnso/notes/59302/all.html