Title: Solving Problems by Searching

AIMA: Chapter 3 (Sections 3.4 and 3.5)

Introduction to Artificial Intelligence

CSCE 476-876, Spring 2000

www.cse.unl.edu/~cse476

Berthe Y. Choueiry

choueiry@cse.unl.edu, (402)476-5444

Summary

- Problem-solving agents

  Formulate, Search, Execute

- Formulating problems
  - Problem types:
    Simple-state, Multiple-State, Contingency, Exploration
  - Problem components:
    State, Operators, Goal test, Path cost
    Solution: sequence of actions from initial state to goal state
  - Importance of modeling

Today $\longrightarrow$ Search

Outline

- Generate action sequences
  - tree, node, fringe, queue.
  - Data structure
- Search strategies:
  - Four evaluation criteria.. ?
  - Six strategies for uniformed search

Search: generate action sequences

partial solution: sequence yielding a (non goal) intermediate state

Search $\left\{ \begin{array}{l} \text{generate} \\ \text{maintain} \end{array} \right\}$ a set of sequences of partial solutions

Two aspects:

- how to generate sequences
- which data structures to keep track of them

Search generate action sequences

Basic idea:

offline, simulated exploration of state space

by generating successors of already-explored states

→ *expanding* states

Start from a state, test if it is a goal state

If it is, we are done

If it is not: *expand state*

Apply all operators applicable to current state to

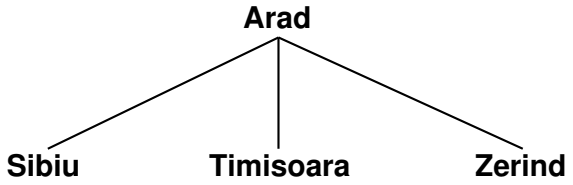
generate all possible sequences of future states

now we have set of partial solutions

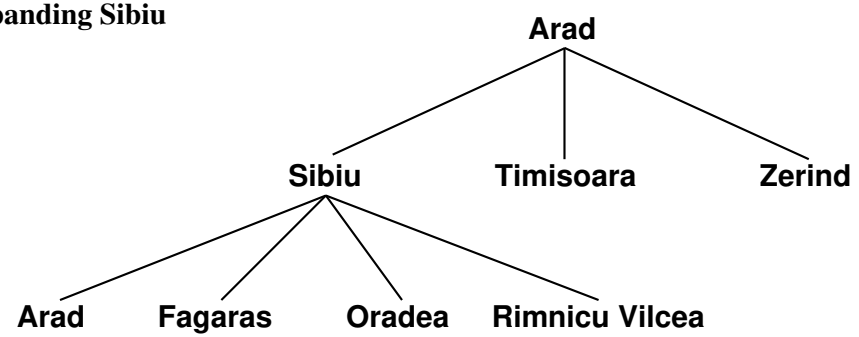
...

(a) The initial state

(b) After expanding Arad



(c) After expanding Sibiu



Search tree, nodes $\begin{cases} \text{root: initial state} \\ \text{leaves: states that can/should not be expanded} \end{cases}$

Data structure

for a search node:

- state in space state
- parent node
- operator used to generate node
- **depth**: # nodes from root (path length)
- path cost $g(x)$

A node x has a parent, children, depth, path cost $g(x)$

Nodes to be expanded:

- constitute a fringe (frontier)
- managed in a queue

Warning:

Do not confuse: State space and Search (tree) space

Holiday in Romania:
- What is a state?
- What is the state space?
- What is the size of state space?
- What is the size of search tree ?

A node has a parent, children, depth, path cost $g(x)$

A state has no parent, children, depth, etc..

```
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
```

Essence of search: which node to expand first?

$\longrightarrow$ search strategy

A strategy is defined by picking the *order of node expansion*

Types of Search

Uniformed : use only information available in problem definition

Heuristic : exploits some knowledge of the domain

Uninformed search strategies

1. Breadth-first search
2. Uniform-cost search
3. Depth-first search
4. Depth-limited search
5. Iterative deepening search
6. Bidirectional search

Search strategies

Criteria for evaluating search:

1. Completeness: does it always find a solution if one exists?
2. Time complexity: number of nodes generated/expanded
3. Space complexity: maximum number of nodes in memory
4. Optimality: does it always find a least-cost solution?

Time/space complexity 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 ∞)

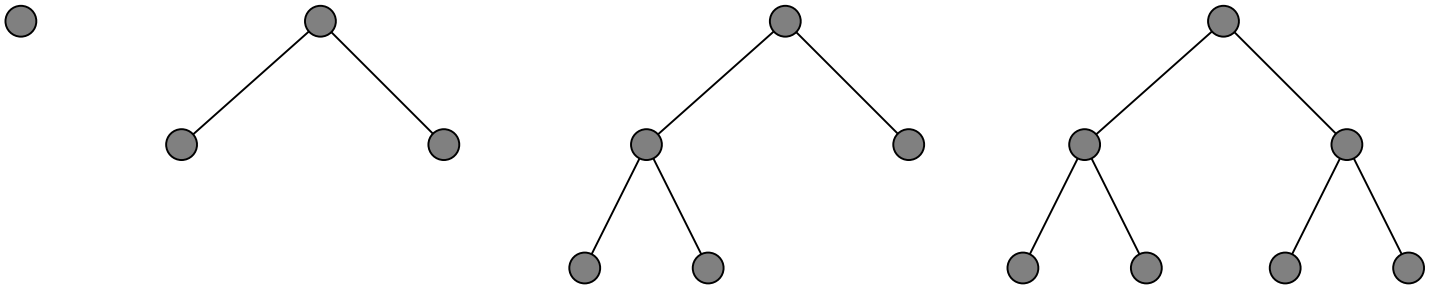
Breadth-first search (I)

→ Expand root node

→ Expand *all* children of root

→ Expand *each* child of root

→ Expand successors of each child of root, etc.



⟶ Expands nodes at depth d before nodes at depth $d+1$

⟶ Systematically considers all paths length 1, then length 2, etc.

⟶ Implement: put successors at end of queue

Breadth-first search (II)

$\longrightarrow$ One solution?

$\longrightarrow$ Many solutions? Finds shallowest goal first

1. Complete?
2. Optimal? (provided cost increases monotonically with depth)
3. Time? $O(b^d)$ $\begin{cases} \text{branching factor } b \\ \text{depth } d \end{cases}$
4. Space? same, $O(b^d)$, keeps every node in memory, big problem

Depth	Nodes	Time	Memory
0	1	1 millisecond	100 bytes
2	111	.1 seconds	11 kilobytes
4	11,111	11 seconds	1 megabyte
6	10^6	18 minutes	111 megabytes
8	10^8	31 hours	11 gigabytes
10	10^{10}	128 days	1 terabyte
12	10^{12}	35 years	111 terabytes
14	10^{14}	3500 years	11,111 terabytes

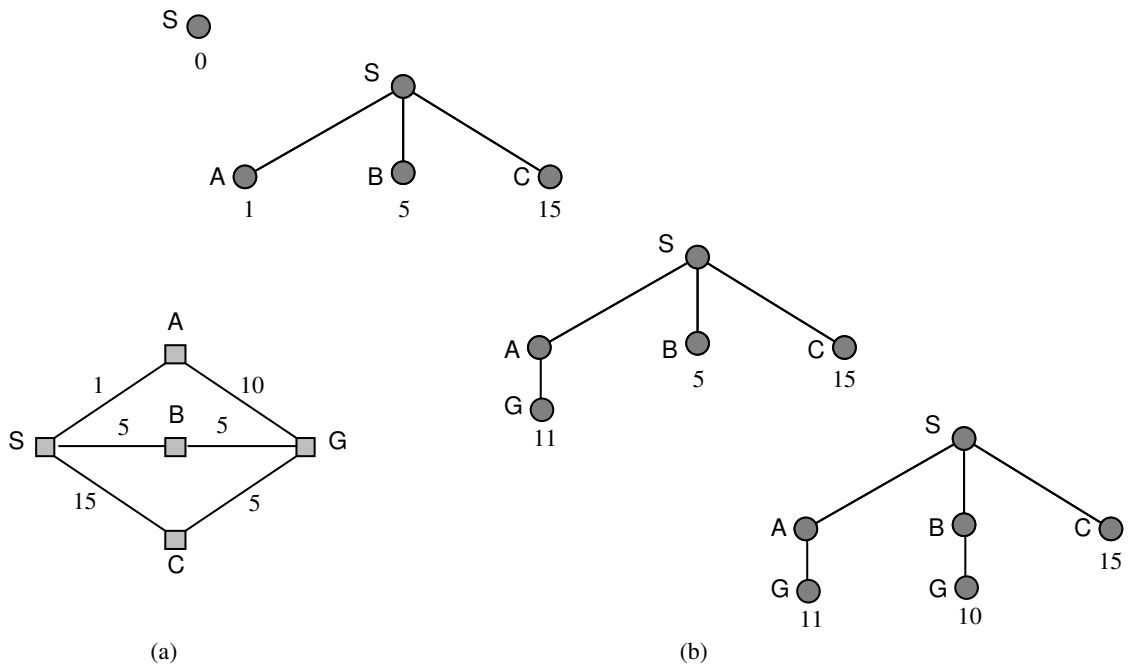
Uniform-cost search (I)

$\longrightarrow$ Breadth-first does not consider path cost $g(x)$

$\longrightarrow$ Uniform-cost expands first lowest-cost node on the fringe

$\longrightarrow$ Implement: sort queue in decreasing cost order

When $g(x) = \text{Depth}(x) \longrightarrow$ Breadth-first $\equiv$ Uniform-cost



Uniform-cost search (II)

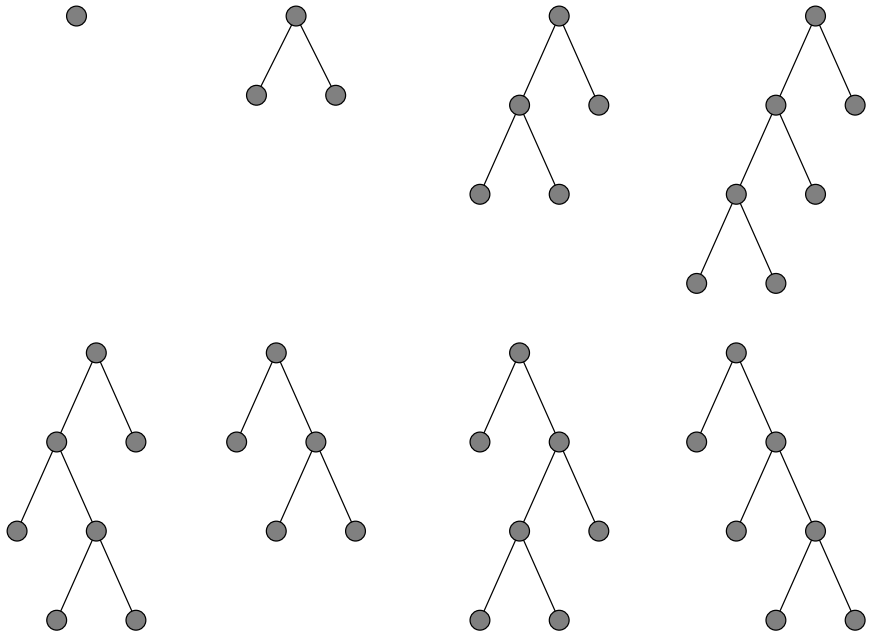
1. Complete?
   Yes, if cost $\geq \epsilon$
2. Optimal?
   If the cost is a monotonically increasing function
   When cost is added up along path, an operator's cost .....?
3. Time?
   # of nodes with $g \leq$ cost of optimal solution
   worst-case $O(b^d)$
4. Space?
   # of nodes with $g \leq$ cost of optimal solution
   worst-case $O(b^d)$

Depth-first search (I)

$\longrightarrow$ Expands nodes at deepest level in tree

$\longrightarrow$ When dead-end, goes back to shallower levels

$\longrightarrow$ Implement: put successors at end of queue (or, recursive calls)



$\longrightarrow$ Little memory: path and unexpanded nodes

For b: branching factor, m: maximum depth, space .....?

Depth-first search (II)

Time complexity:

We may need to expand all paths, $O(b^m)$

When there are many solutions, DFS may be quicker than BFS

When m is big, ∞ (deep, loops), .. troubles

$\longrightarrow$ Major drawback of DFS: going deep where there is no solution..

Properties:

1. Complete?
2. Optimal?
3. Time? Woow.
4. Space?

Depth-limited search (I)

$\longrightarrow$ DFS is going too deep, put a threshold on depth!
For instance, 20 cities on map for Romania, any node deeper than 19 is cycling. Don't expand deeper!

$\longrightarrow$ Implement: nodes at depth l have no successor

Properties:

1. Complete?
2. Optimal?
3. Time? (given l depth limit)
4. Space? (given l depth limit)

Problem: how to choose l?

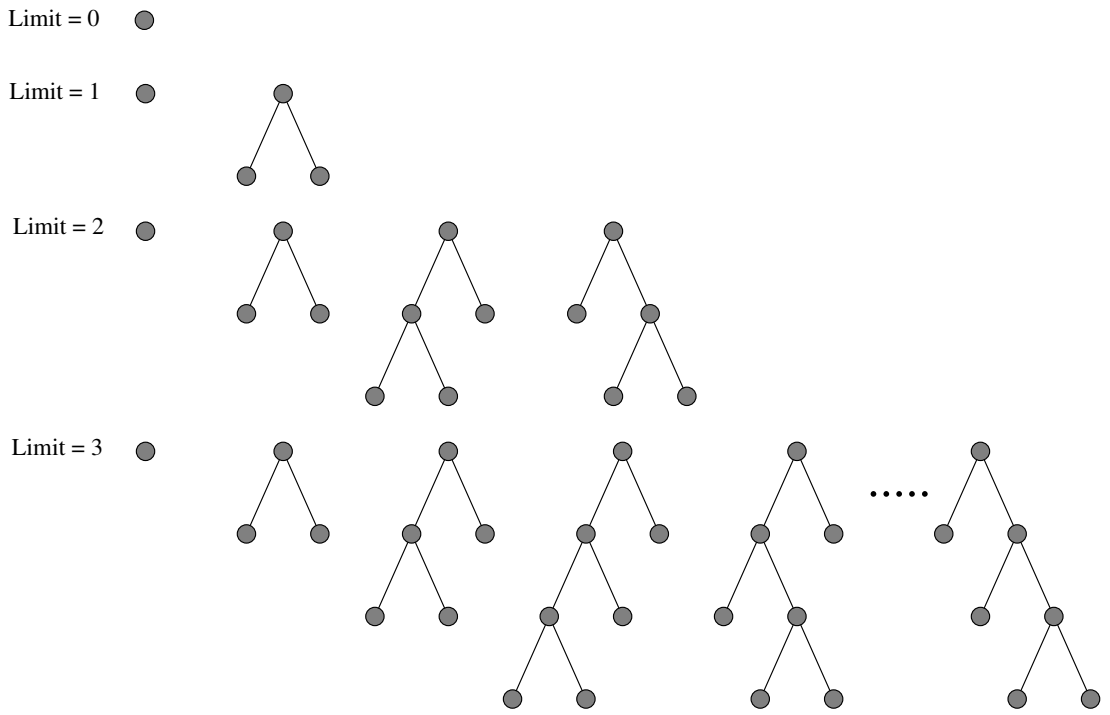
Iterative-deepening search (I)

→ DLS with depth = 0

→ DLS with depth = 1

→ DLS with depth = 2

→ DLS with depth = 3...



⟶ Combines benefits of DFS and BFS

Iterative-deepening search (II)

$\longrightarrow$ combines benefits of DFS and BFS

Properties:

1. Complete? like BFS
2. Optimal? like BFS (if step cost = 1)
3. Space? like DFS
4. Time? same

$\longrightarrow$ Some nodes are expanded several times, wasteful?

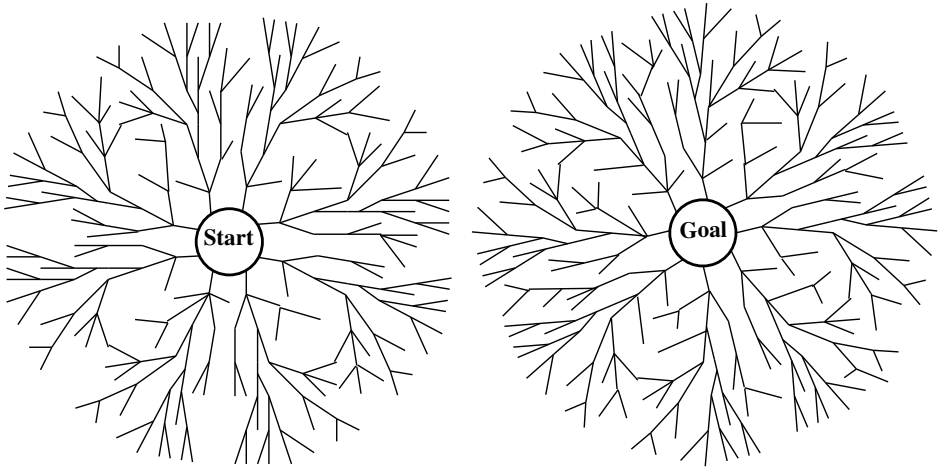
expansions in BFS: $1 + b + b^2 + b^3 + \ldots + b^d = O(b^d)$

in IDS: $(d+1).1 + (d).b + (d-1).b^2 + \ldots + 1.b^d = O(b^d)$

$\longrightarrow$ IDS is preferred when search space is large and depth unknown

Bidirectional search (I)

→ Given initial state and the goal state, start search from both ends and meet in the middle



→ Assume same b branching factor, $\exists$ solution at depth d, time: $O(2b^{d/2}) = O(b^{d/2})$

$b = 10, d = 6$, DFS= 1,111,111 nodes, BDS=2,222 nodes!

Bidirectional search (II)

In practice :—(

- Need to define predecessor operators to search backwards
  If operator are invertible, no problem
- What if $\exists$ many goals (set state)?
  do as for multiple-state search
- need to check the 2 fringes to see how they match
  need to check whether any node in one space appears in the other space (use hashing)
  need to keep all nodes in a half in memory $O(b^{d/2})$
- What kind of search in each half space?

Summary

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Time	b^d	b^d	b^m	b^l	b^d	$b^{d/2}$
Space	b^d	b^d	bm	bl	bd	$b^{d/2}$
Optimal?	Yes	Yes	No	No	Yes	Yes
Complete?	Yes	Yes	No	Yes, if $l \geq d$	Yes	Yes

b branching factor

d solution depth

m maximum depth of tree

l depth limit

List of terms for glossary:

Due date: Mon, Jan 31, 2000

- Completeness
- Complexity (time/space)
- Diameter (of a graph). *Look in graph theory.*
- Expand (a search node)
- Fringe
- Optimality
- Search:
  - Breadth-first
  - Heuristic
  - Node
  - Space
  - Strategy
  - Tree
  - Uniformed (blind)