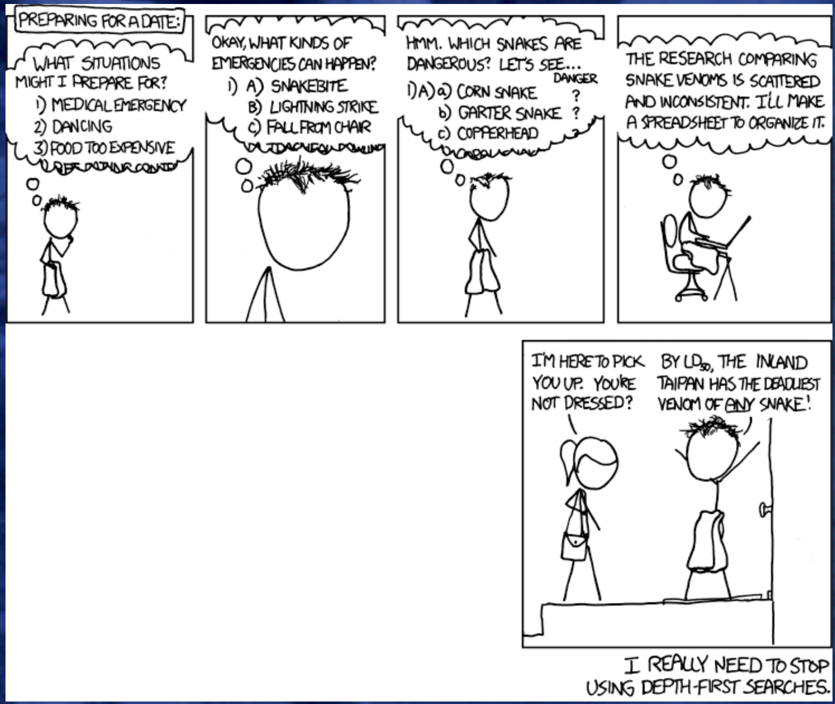
Uninformed Search Algorithms



CITS3001 Algorithms, Agents and Artificial Intelligence

2021, Semester 2

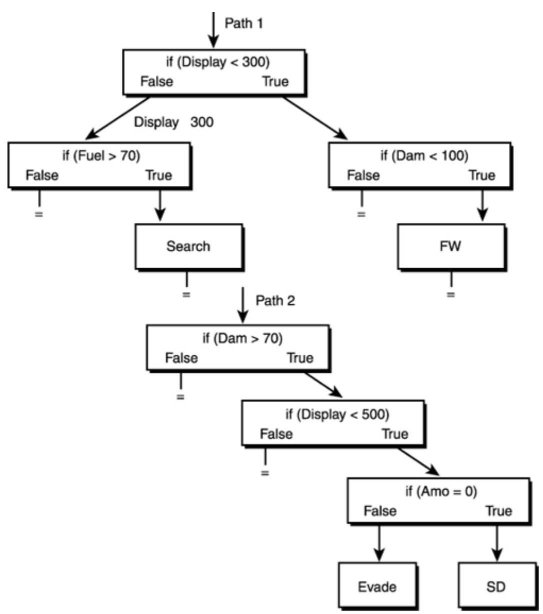
Tim French

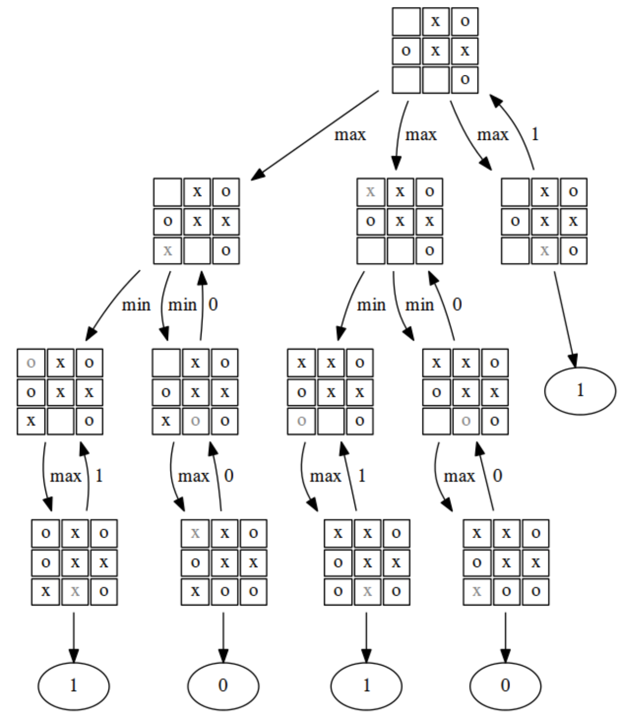
Department of Computer Science and Software Engineering

The University of Western Australia

Introduction

- We will formalise the definition of a problem for an agent to solve, conceptualising
  - The environment
  - The goal to achieve
  - The actions available
  - *etc.*
- We will describe the fundamental search algorithms available to agents





Problem Solving and Search

- We have seen that most intelligent agent models have
  - Some knowledge of the state of the world
  - A notion of how *actions* or *operations* change the state of the world
  - A notion of the *cost* of each possible action
  - One or more *goals*, or states of the world, that they would like to bring about
- Finding a sequence of actions that changes the world from its current state to a desired goal state is a *search problem*
  - Or a basic *planning problem*
- Usually we want to find the cheapest sequence
  - The cost of a sequence of actions (or a *path*) is just the sum of their individual costs

- ***Search algorithms are the cornerstone of AI***
- We will examine
  - How all of the above concepts are formalised
  - The most common search strategies

A Running Example

- Our running example (taken from AIMA) is a simplified road map of part of Romania
- The *state* of the world is our current location
- The only *operator* available is to drive from one city to a connected city
- The *cost* of an action is the distance between the cities
- The *start state* is where we start from (Arad)
- The *goal state* is where we want to get to (Bucharest)
- A *solution* is a sequence of cities that we drive through
- In general the state of the world is described abstractly, focussing only on the features relevant to the problem

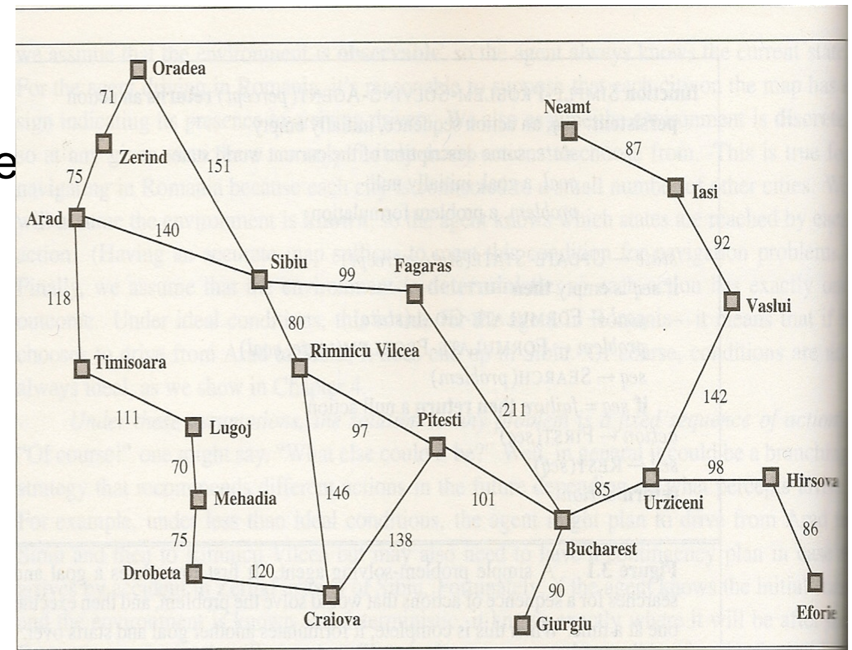


Figure 3.2 A simplified road map of part of Romania.

A second example: the 8-puzzle

- Slide the tiles in the puzzle until the goal state is reached
- The state is the current layout of the tiles
- The only operator is to slide a tile into the blank square
  - Or to slide the blank square…
- The cost of each action is 1
- The start state is the puzzle’s initial configuration
- The goal state is as shown
- A solution is a sequence of tile-moves

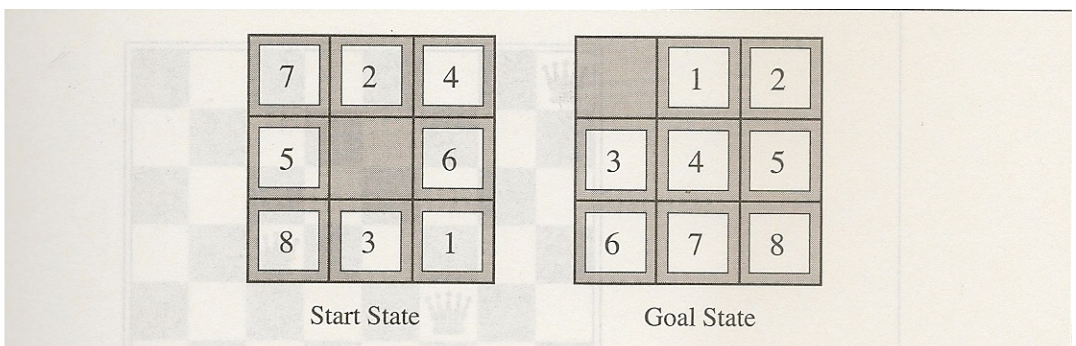
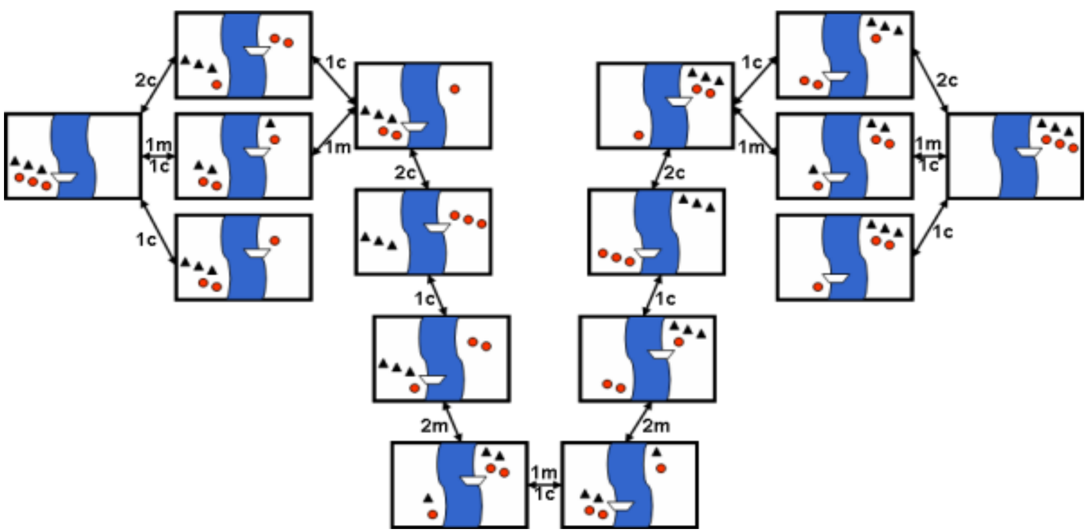
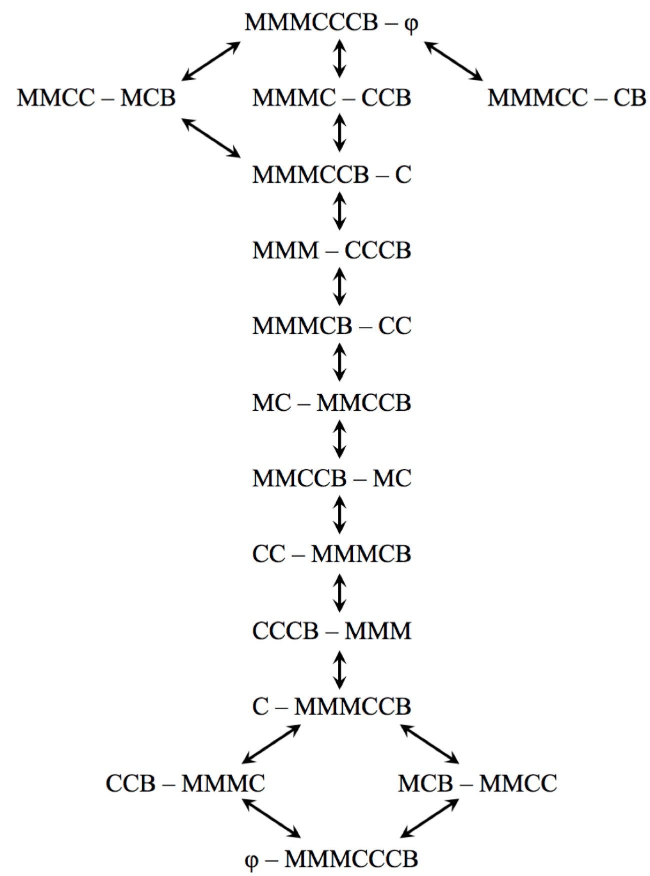


Figure 3.4 A typical instance of the 8-puzzle.

3rd example: Missionaries and Cannibals

- Start state: 3 missionaries, 3 cannibals, and one boat that holds up to 2 people, all on one side of a river
- Goal state: everybody on the other side of the river
- Current state: the positions of the people and the boat
  - A state is *legal* only if no one gets eaten
  - i.e. cannibals never outnumber missionaries
- Operator: 1 or 2 people cross the river in the boat
- Cost: 1





A Generalised Search Algorithm

- The fundamental idea is
  - At any given moment we are in some state *s*
  - *s* will usually offer several possible actions
  - Choose one action to explore first
  - Keep *s* and the other actions to explore later, in case the first one doesn't deliver
- Action-selection is determined by a *search strategy*
- We picture a *search tree* of states, expanding outwards from the initial state of the problem

```
GeneralSearch (problem, strategy):
initialise the tree with the initial state of problem
while no solution found
   if there are no possible expansions left
   then return failure
   else use strategy to choose a leaf node x to expand
   if x is a goal state
       then return success
   else expand x
       add the new nodes to the tree
```

Exploring Romania

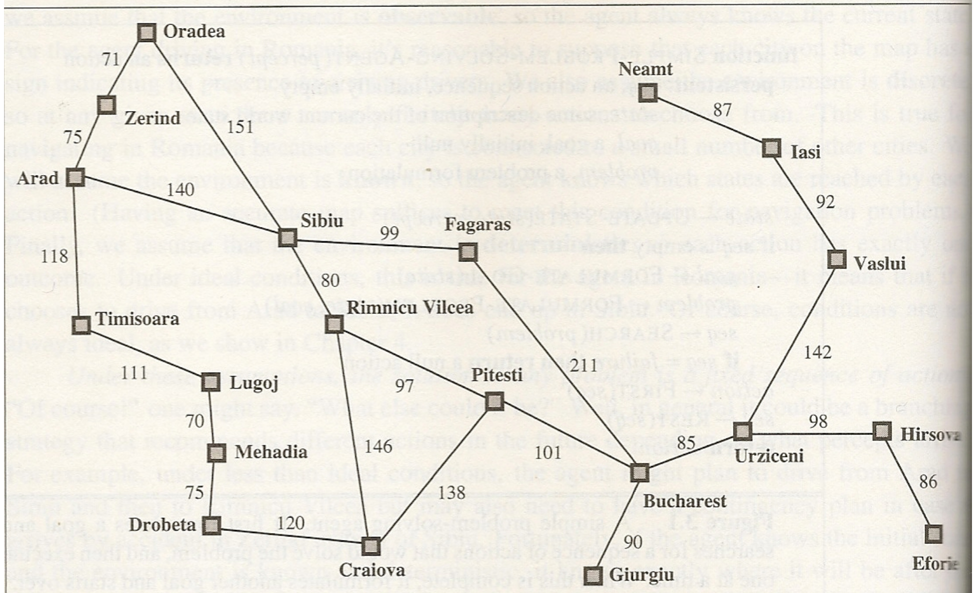


Figure 3.2 A simplified road map of part of Romania.

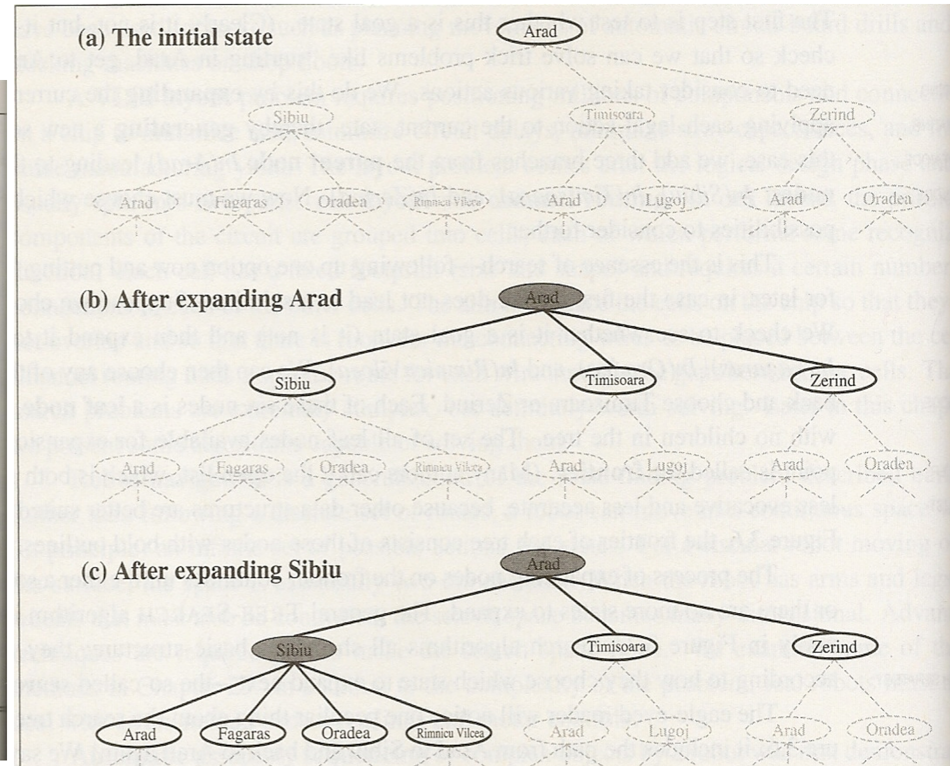


Figure 3.6 Partial search trees for finding a route from Arad to Bucharest. Nodes that have been expanded are shaded; nodes that have been generated but not yet expanded are outlined in bold; nodes that have not yet been generated are shown in faint dashed lines.

Alternative Formulation

THE UNIVERSITY OF WESTERN AUSTRALIA

- Given
  - A set of possible states S
  - A start state s_0
  - A goal function $g(s) \rightarrow \{true, false\}$
  - A terminal condition $t(s) \rightarrow \{true, false\}$

- The data structure used to store U can impose
  an order in which the nodes will be visited
  - e.g. a priority queue where nodes are stored in the order in which they should be visited

```
U = {s0}              -- unvisited nodes
V = {}                -- visited nodes
while U ≠ {}
  s = select a node from U
  if s ∈ V            -- occurs check
    then discard s
  else if g(s)
    then return success
  else if t(s) -- cut-off check
    then discard s
  else
    V = V + {s}
    U = U – {s} + successors(s)
```

Comparing Search Strategies

- The performance of search strategies is generally compared in four ways
- *Completeness*: is the strategy guaranteed to find a solution, assuming that there is one?
- *Optimality*: is the strategy guaranteed to find the optimal solution?
- *Time complexity*: how long does the strategy take to find a solution?
- *Space complexity*: how much memory is needed to conduct the search?
- The complexities are often expressed in terms of
  - *b*: maximum *branching factor* of the tree
  - *m*: maximum *depth* of the search space
  - *d*: depth of the *least-cost* solution

Uninformed Search Strategies

- Breadth-first search
  - Expand the shallowest node next
- Uniform-cost search
  - Expand the lowest-cost node next
- Depth-first search
  - Expand the deepest node next
- Depth-limited search
  - Depth-first, but with a cut-off depth
- Iterative deepening depth-first search
  - Repeated depth-limited, with increasing cut-offs
- Bidirectional search
  - Search from both ends concurrently
- In the next lecture, we will look at *informed search strategies*, that use additional information

Breadth-first Search

- Expand the shallowest node next
  - Expand all nodes at one level before moving down
- Complete: yes, if *b* is finite
- Optimal: yes, if all step-costs are equal
- Time: O($1 + b + b^2 + \ldots + b^d$) = O($b^d$)
- Space: O(b^d), because all of the nodes at one level must be stored simultaneously
- Space is the big problem
- You might wait thirteen days to solve a problem
  - But who has a petabyte of memory!?
- Welcome to AI!

Depth	Nodes	Time	Memory
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	10^6	1.1 seconds	1 gigabyte
8	10^8	2 minutes	103 gigabytes
10	10^{10}	3 hours	10 terabytes
12	10^{12}	13 days	1 petabyte
14	10^{14}	3.5 years	99 petabytes
16	10^{16}	350 years	10 exabytes

Figure 3.13 Time and memory requirements for breadth-first search. The numbers shown assume branching factor $b = 10$; 1 million nodes/second; 1000 bytes/node.

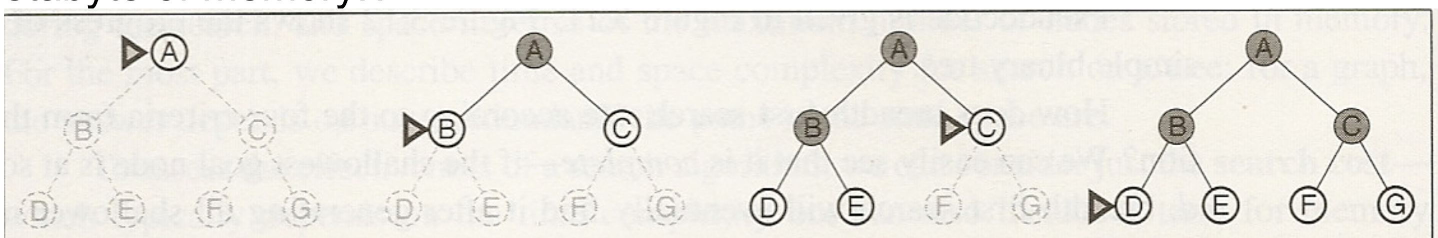


Figure 3.12 Breadth-first search on a simple binary tree. At each stage, the node to be expanded next is indicated by a marker.

Uniform cost saerch

- Expand the lowest-cost node next
  - Basically a version of breadth-first that allows for varying step-costs
- Complete: yes, if all step-costs ≥ 0
- Optimal: as above
- Time: O(*n*), where *n* is the number of nodes with cost less than the optimum
- Space: as above

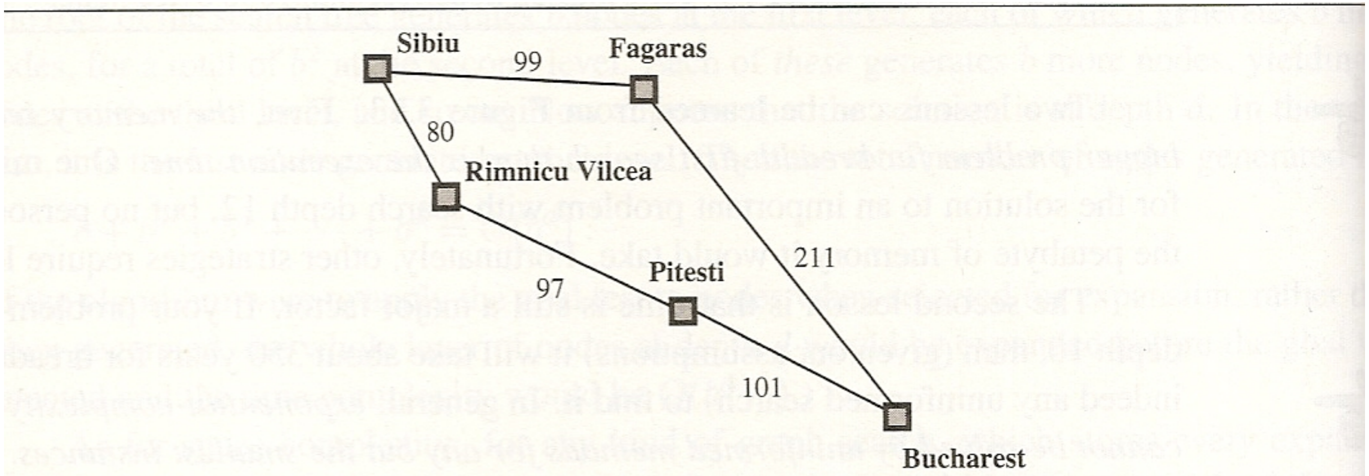


Figure 3.15 Part of the Romania state space, selected to illustrate uniform-cost search.

Depth-first search

- Expand the deepest node next
  - Follow one path until you can go no further, then backtrack to the last choice point and try an alternative

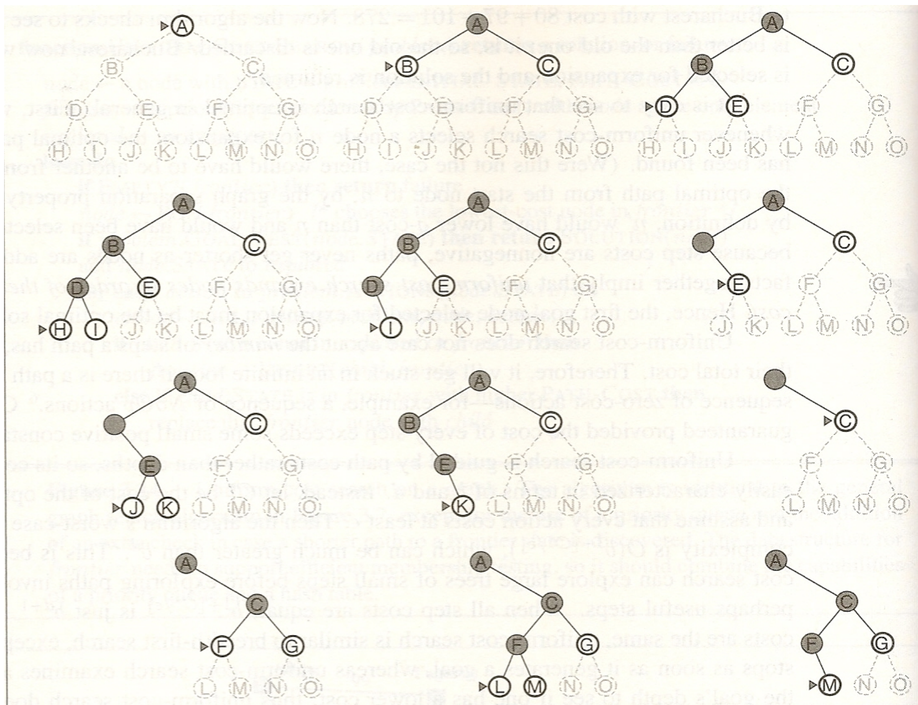


Figure 3.16 Depth-first search on a binary tree. The unexplored region is shown in light gray. Explored nodes with no descendants in the frontier are removed from memory. Nodes at depth 3 have no successors and M is the only goal node.

Usually needs an occurs-check (as per Page 10) to prevent looping

- Complete: no, fails in infinite-depth spaces
- Optimal: no, could hit any solution first
- Time: $O(b^m)$, follows paths "all the way down"
- Space: $O(bm)$, because it only needs to store the current path plus untried alternatives

Space is a ***huge*** advantage
The other metrics can be big disadvantages

Depth-limited search

- Depth-first, but with a cut-off depth
  - Terminate paths at depth *l*
  - cf. *t(s)* on Page 10
- Sometimes used to apply depth-first search to infinite (or effectively infinite) spaces
  - Find best solution with limited resources
  - e.g. game-playing (Lecture 8)
- Works well if we have a good way to choose *l*
  - e.g. the Romanian map has diameter 9

Iterative deepening depth-first search

- Repeated depth-limited, with increasing cut-offs
- Probe deeper and deeper, iteratively increasing *l*

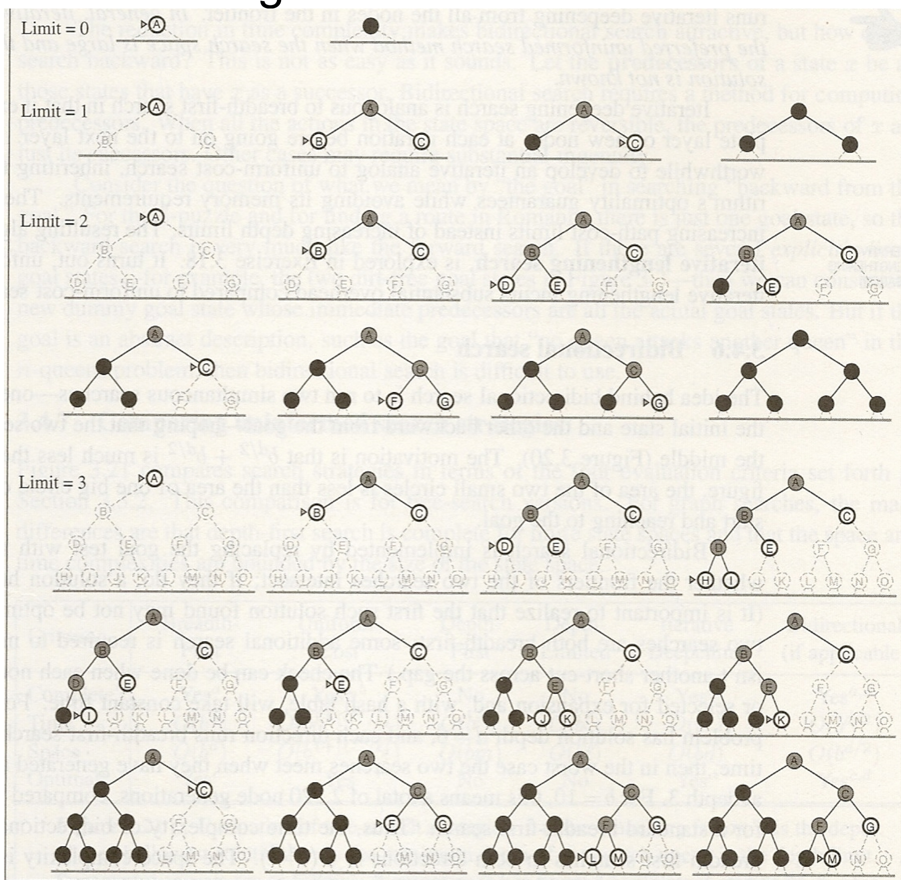


Figure 3.19 Four iterations of iterative deepening search on a binary tree.

- Complete: yes
- Optimal: yes, for constant step-costs
  - And easily adapted to varying step-costs
- Time: $O((d+1)b^0 + db^1 + (d-1)b^2 + \ldots + b^d) = O(b^d)$
- Space: $O(bd)$

The multiplying factors in the time complexity come from the repeated expansion of nodes near the root

But this is not normally a big problem

For typical values of *b*, the last layer of the tree dominates the space requirements

And it's worth it for the space complexity!

Iterative deepening allows a system to adapt to resource limits

In this context it acts an *anytime algorithm*

Find an initial (hopefully usable) solution, then try to find a better one

A common optimisation is start off *l* bigger than 0

Bi-directional search

THE UNIVERSITY OF WESTERN AUSTRALIA

- Search from both ends concurrently
- Usually expands many fewer nodes than unidirectional
  - $2b^{d/2} << b^d$
- But raises many other difficulties
  - There may be many goal states to start from
  - Formalising backward steps may be difficult
  - The “backwards branching factor” may be much bigger than *b*
  - The cost of checking when the two sides meet may be high
- e.g. chess
  - There are many, many checkmate positions
  - Was the last move a capture?

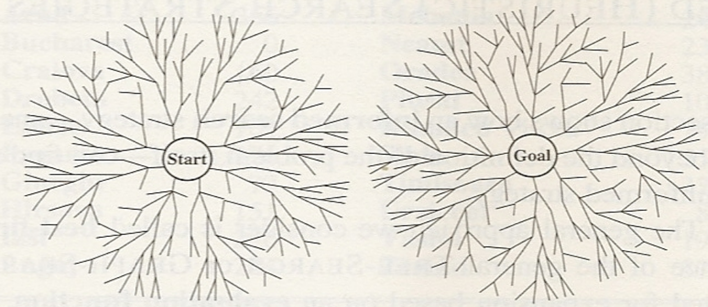


Figure 3.20 A schematic view of a bidirectional search that is about to succeed when a branch from the start node meets a branch from the goal node.

Summary

- Iterative deepening offers
  - the completeness and optimality of breadth-first
  - the space advantage of depth-first

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes^a	Yes^{a,b}	No	No	Yes^a	Yes^{a,d}
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes^c	Yes	No	No	Yes^c	Yes^{c,d}

Figure 3.21 Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: ^{a} complete if b is finite; ^{b} complete if step costs $\geq \epsilon$ for positive ϵ; ^{c} optimal if step costs are all identical; ^{d} if both directions use breadth-first search.