Problem Solving Using Search

Introduction to Artificial Intelligence

G. Lakemeyer

Winter Term 2018/19

Problem Solving Using Search

Here we consider goal-oriented agents.

Given an initial (world) state, the agent wants to reach a certain goal using appropriate actions, where actions transform one state into another.

A goal is a set of world states, which the agent finds desirable (wants to reach one of them).

Example: Driving from City A to City B.

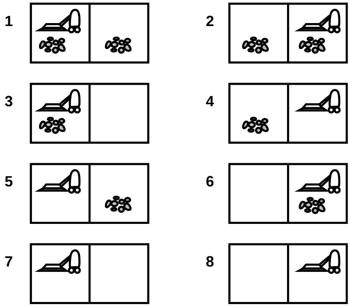
Problem formulation

- How to represent world states?
- Which actions should be considered?

Search: Finding an action sequence which transforms an initial state into a goal state.

Problem Classification

The vaccuum-world states:



- Single-state problem
  complete world knowledge,
  complete knowledge about the actions
- Multiple-state problem
  incomplete world knowledge,
  complete knowledge about the actions
- Contingency problem
  incomplete knowledge about actions,
  needs to gather information at run-time
- Exploration problem
  World states and effect of actions are both unkown. Very difficult!

Some Terminology (1)

Initial state:	World state which the agent believes to be in initially.
State space:	Set of all possible states.
Operator:	Description of which state is reached by an action from a given state.
Successor function S:	$S(x)$ returns the set of states reachable by any action from state x.
Goal test:	Tests whether a state is a goal state.

Some Terminology (2)

Path:	Sequence of actions.
Path cost:	Cost function over paths, usually the sum of the costs of the actions of the path, often denoted as g.
Solution:	Path from the initial state to a state that satisfies the goal test.
Search cost:	Time and memory needed to find a solution.
Total cost:	Sum of the path cost (of the solution) and search cost.

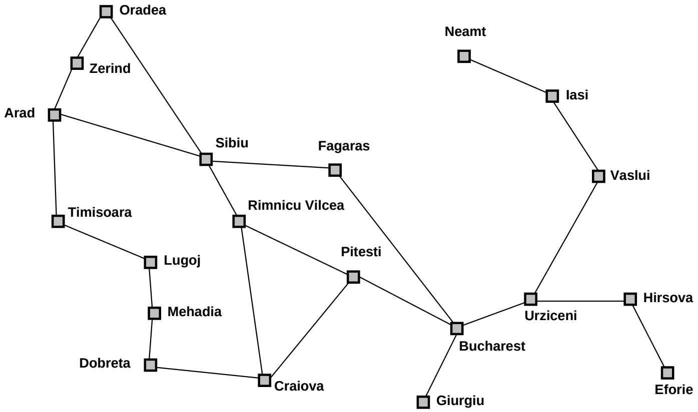
Note: For multiple-state problems, replace state by state set, operators apply to all elements of the set, and paths connect sets of states.

Choosing a State Space

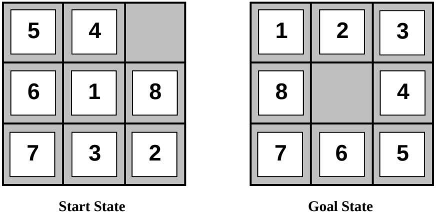
It is an art to choose the right state space. Want to leave out unnecessary detail (Abstraction)!

Example: Drive from A to B.

Only represent the most important places.

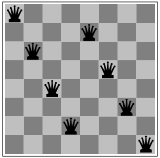


8-Puzzle



- States: Describe the location of each tile including the blank.
- Operators: Blank moves left, right, up, or down.
- Goal test: Does the state match picture on the right?
- Path cost: Each step costs 1 unit.
- Search cost: Problem is NP-complete in the size of the puzzle.

8-Queens Problem



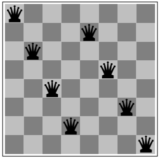
- Goal test: 8 queens on the board, all safe.
- Path cost: 0 (only the solution counts).

Representation 1

- States: any placement of 0–8 queens.
- Operators: place a queen on the board.
- Problem: Too many states (64^8)!

8-Queens Problem

- Goal test: 8 queens on the board, all safe.
- Path cost: 0 (only the solution counts).



Representation 1

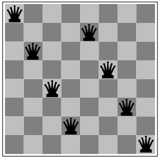
- States: any placement of 0–8 queens.
- Operators: place a queen on the board.
- Problem: Too many states (64^8)!

Representation 2

- States: 0–8 on board, all safe.
- Operators: place a queen starting from left.
- Problem: few state (2057), but sometimes deadend (see figure).

8-Queens Problem

- Goal test: 8 queens on the board, all safe.
- Path cost: 0 (only the solution counts).



Representation 1

- States: any placement of 0–8 queens.
- Operators: place a queen on the board.
- Problem: Too many states (64^8)!

Representation 2

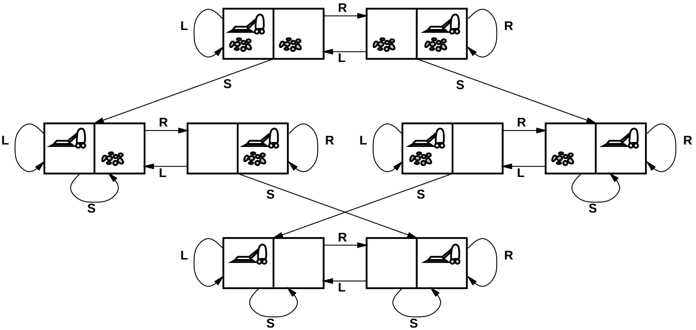
- States: 0–8 on board, all safe.
- Operators: place a queen starting from left.
- Problem: few state (2057), but sometimes deadend (see figure).

Representation 3

- States: 8 queens on board, one per column.
- Operators: move a queen under attack in the same column.

Vacuum World 1

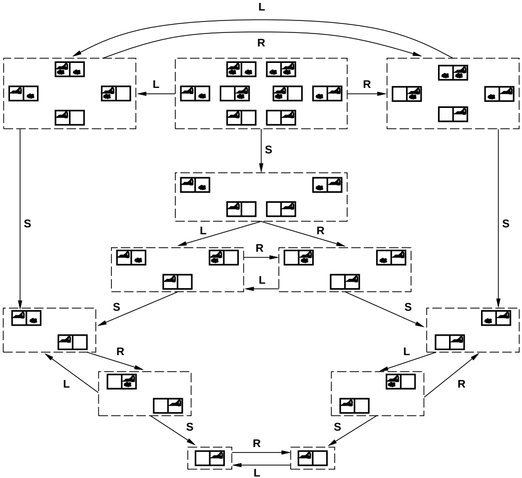
Complete World Knowledge (single-state problem):



- States: All combinations of robot location and presence of dust (8).
- Operators: left (L), right (R), suck (S).
- Goal test: no dust in any room.
- Path cost: 1 unit per action.

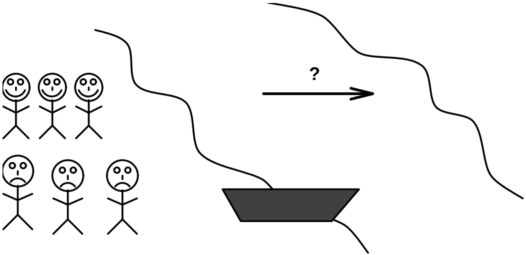
Vacuum World 2

Incomplete Knowledge (multiple-state problem).



- State set: All subsets of the 8 states.
- Goal test: No dust anywhere in the elements of the state set.

Missionaries and cannibals (MaC)



3 missionaries and 3 cannibals want to cross a river. There is a row boat holding at most 2 people. Should the cannibals ever outnumber the missionaries, the missionaries will be eaten. How do all six make it across safely.

Non-Intended MaC Solutions

1. All six walk across the bridge nearby.
2. They take a helicopter.
3. There are three more missionaries on the other side.
4. ...

Contradicts the underlying assumption that the description of the puzzle contains all relevant information.

Sabotaging the Correct MaC Solution

1. There is a hole in the boat.
2. The missionaries are too weak to row.
3. One missionary goes overboard and drowns.
4. ...

Contradicts the assumptions of what happens normally.

Formalizing the MaC-Problem

- States: Triple (x,y,z) with $0 \leq x, y \leq 3$ and $0 \leq z \leq 1$, where x, y, and z denote how many missionaries, cannibals, and boats are currently on the left river bank.
- Initial state: (3,3,1)
- Operators: Either one missionary, one cannibal, two missionaries, two cannibals, or one of each cross the river in a boat. (i.e. 5 operators)

  Note: not every state is reachable [e.g. (0,0,1)].
- Goal state: (0,0,0)
- Path cost: 1 unit per river crossing.

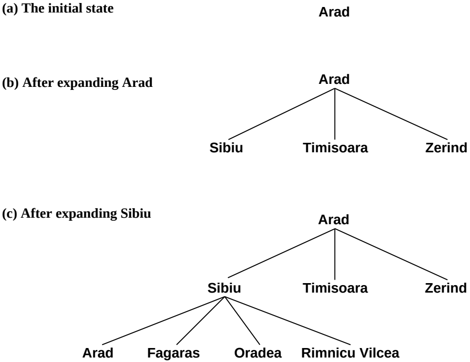
Note: Solution due to Saul Amarel (1968).

Search in General

Starting from the initial state we always expand one state, i.e. we create all successor states of that state.

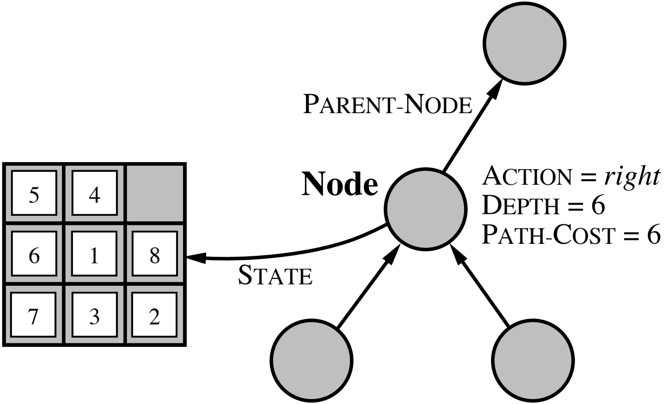
Search tree = the tree induced by expansion.

Nodes contain world states; edges = actions.



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

Search Nodes



Evaluating Search Strategies

- Completeness: Does it always find a solution if one exists?
- Time Complexity: How long does it take to find a solution in the worst case?
- Space Complexity: How much memory is used in the worst case?
- Optimality: Does it always find the best solution?

Uninformed vs. Informed Search

Uninformed (blind) search:

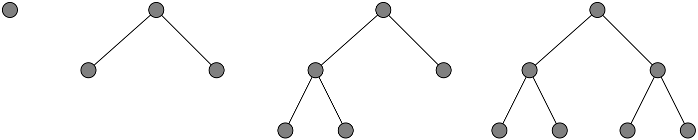
- No information available about the length or the cost of a solution.
- The main methods are:
  breadth-first search, uniform cost search, depth-first search,
  depth-limited search, iterative deepening, bidirectional search.

Informed (heuristic) search:

- Here the user has information about the length or the cost of a solution, which often helps in speeding up the search.
- We will look at:
  Greedy search, A^* and some variants, hill-climbing, and
  simulated annealing.

Breadth-First Search

Expand the nodes in the same order in which they are generated.



- Always finds the shallowest solution (completeness).
- Solutions are optimal, provided all actions have identical non-negative cost.

Cost of Breadth-First Search

The Cost is very high!. Let b be the max. branching factor, d the depth of a solution. Then at most

$$b + b^2 + b^3 + \ldots + b^d + b^{d+1} - b$$

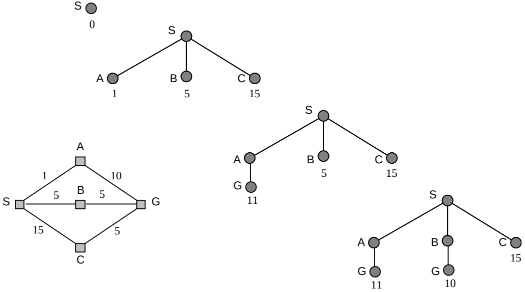
nodes need to be generated, i.e. $O(b^{d+1})$.

Example: $b = 10$, 10000 nodes/s; 1000bytes/node:

Depth	Nodes	Time	Memory
2	1100	.11 seconds	1 megabyte
4	111,100	11 seconds	106 megabytes
6	10^7	19 minutes	10 gigabytes
8	10^9	31 hours	1 terabytes
10	10^{11}	129 days	101 terabytes
12	10^{13}	35 years	10 petabytes
14	10^{15}	3,523 years	1 exabyte

Uniform cost search

A modified breadth-first search, which always expands node n with the least path cost $g(n)$.



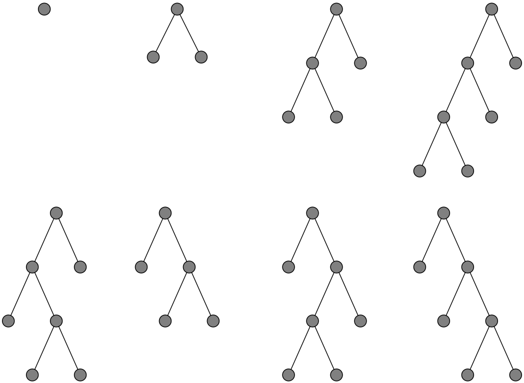
(a) (b)

Note: Always finds the cheapest solution, provided

$$g(successor(n)) \geq g(n) \text{ for all } n.$$

Depth-First Search

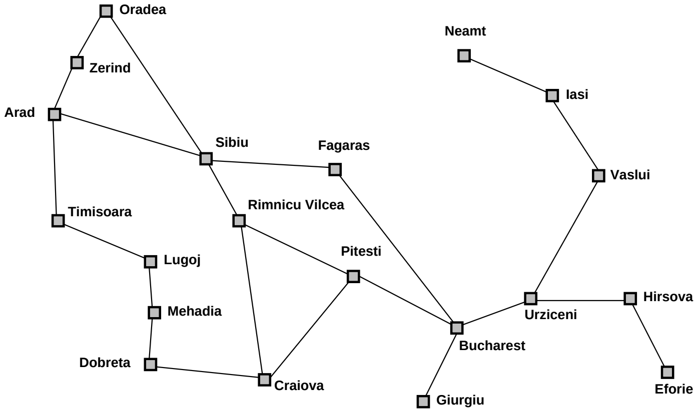
Always expand the node at the deepest level.



Depth-limited Search

Perform depth-first search, but only up to a pre-specified depth.

Route planning: For n cities the maximal depth is $n - 1$.



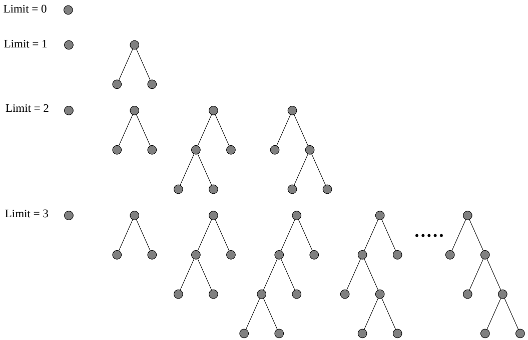
Actually, a maximal depth of 9 suffices (called the diameter of the problem).

Iterative Deepening 1

- Combines depth- and breadth-first search.
- Optimal and complete like breadth-first search, but requires less space.

function ITERATIVE-DEEPENING-SEARCH(*problem*) **returns** a solution sequence
  inputs: *problem*, a problem

  for *depth* $\leftarrow$ 0 **to** ∞ **do**
    if DEPTH-LIMITED-SEARCH(*problem*, *depth*) succeeds **then return** its result
  end
  return failure



Iterative Deepening 2

Number of expansions:

$$(d)b + (d-1)b^2 + \ldots + 3b^{d-2} + 2b^{d-1} + 1b^d$$

Compared to breadth-first search: $b + b^2 + b^3 + \ldots + b^{d-1} + b^d + b^{d+1} - b$.

Note: iterative deepening is better!

Example: $b = 10$, $d = 5$

Breadth-first search:

$$10 + 100 + 1000 + 10000 + 100000 + 999990 = 1111110$$

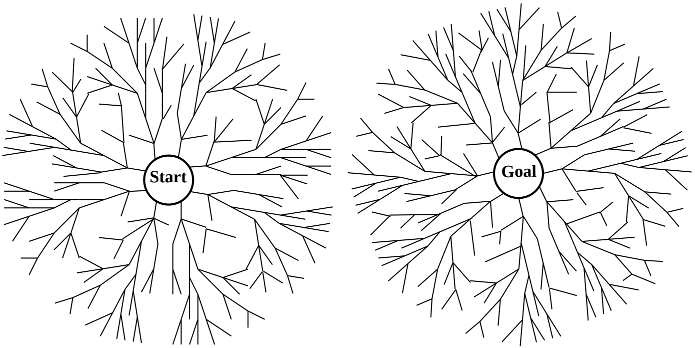
Iterative deepening:

$$50 + 400 + 3000 + 20000 + 100000 = 123450$$

Time complexity: $O(b^d)$
Space complexity: $O(b \times d)$

Note: Preferred method if search depth unknown!

Bidirectional Search



Assuming that forward and backward search are symmetric, we can ideally obtain search times of $O(2 \times b^{d/2}) = O(b^{d/2})$.

Example: For $b = 10$, $d = 6$ we have 2222 nodes instead of 1111111!

Problems with Bidirectional Search

- Operators not always reversible. (Need to compute the predecessor node.)
- Sometimes there are very many goal states, which are described only incompletely. E.g., what are the predecessors of “checkmate” in chess?
- We need efficient methods to decide when the two search methods have met.
- Which kind of search method should one use for each direction? (In the figure it is breadth-first search for both, but that is not always optimal.)

Comparison

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	$O(b^{d+1})$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^{d+1})$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes[c]	Yes	No	No	Yes[c]	Yes[c,d]

b = maximal branching factor

d = depth of a solution path

m = maximal search depth

l = depth restriction

[a]: complete if b is finite; [b]: complete if step costs $\geq \epsilon > 0$;
[c]: optimal if step costs identical; [d]: if both directions use BFS.