Lecture 2: Search - 1

Victor R. Lesser

CMPSCI 683
Fall 2004

Today's lecture

- Why is search the key problem-solving technique in AI?
- Formulating and solving search problems.
- Understanding and comparing several "blind" search techniques.

Overview of Material You Should Know!!

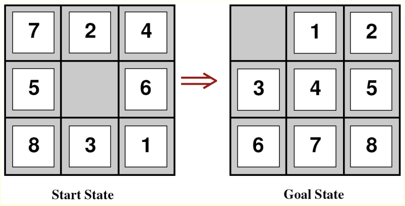
You should have read material in Chapters 1-3

Searching for Solutions

- **The state space model provides a formal definition of a problem and what constitutes a solution to a problem.**
  - Complete and Partial Solutions
- **A solution is found by *searching* through the state space until a (goal) state with "specific" properties is found**
  - Local and Global Properties
- **A solution is**
  - a sequence of operators that will change an initial state to a goal state
  - the attributes of a state that has certain properties
- **Search involves *exploring* (explicitly generating) parts of the state space until a solution is found (or the entire space is explored).**
  - The point of search is to find a solution without exploring the entire state space (without generating the complete state space).

Examples of "toy" problems

- **Puzzles such as the 8-puzzle:**

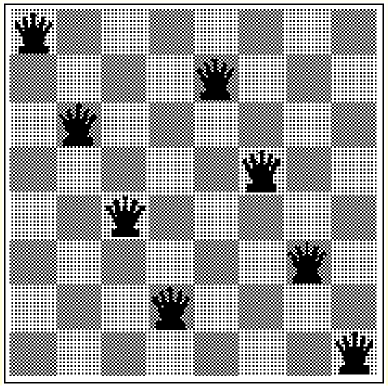


- **Cryptarithmetic problems:**

```
 SEND
 MORE
MONEY
```

The 8-queens problem

- A popular benchmark for AI search
- Solved for n up to 500,000



Explicit solution for $n \geq 4$

[Hoffman, Loessi, and Moore, 1969]

If n is even but not of the form 6k+2:

For j = 1, 2, ..., n/2 place queens on elements

(j, 2j), (n/2+j, 2j-1)

If n is even but not of the form 6k:

For j = 1, 2, ..., n/2 place queens on elements

(j, 1+[(2(j-1) + n/2 - 1) mod n]), (n+1-j, n-[(2(j-1) + n/2 - 1) mod n])

If n is odd:

Use case A or B on n-1 and extend with a queen at (n,n)

Is this a good benchmark problem for testing search techniques?

Real-world problems

- Signal interpretation (e.g. speech understanding)
- Theorem proving (e.g. resolution techniques)
- Combinatorial optimization (e.g. VLSI layout)
- Robot navigation (e.g. path planning)
- Factory scheduling (e.g. flexible manufacturing)
- Symbolic computation (e.g. symbolic integration)

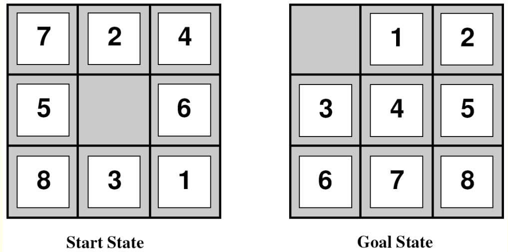
Can we find closed form solutions to these problems?

Formulating search problems

- States and state spaces
- Operators - representing possible actions
  - Successor function
- Initial state and goal test
- Path cost function

Examples: 8-puzzle, 8-queen, path planning, map coloring. What are the corresponding states, operators, initial state, goal test, and path cost.

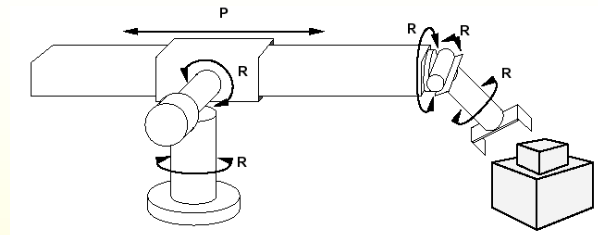
Example: The 8-puzzle



- **States: integer locations of tiles**
- **Actions: move blank left, right, up, down**
- **Goal test: goal state (given)**
- **Path cost: 1 per move**

Note: Solving *n*-puzzle problems optimally is NP-hard

Example: robot assembly



- **States: real-valued coordinates of robot joint angles; parts of the object to be assembled**
- **Actions: continuous motion of robot joints**
- **Goal test: complete assembly**
- **Path cost: time to execute**

Example: Route Finding

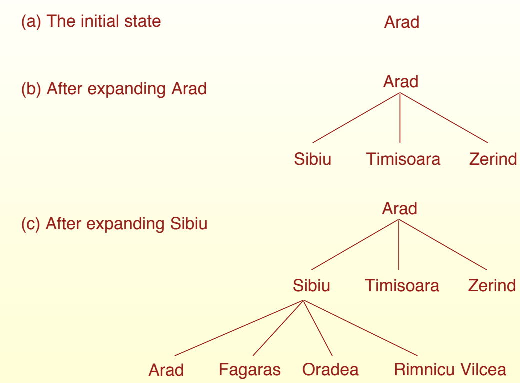
State space representation:

- There is a state corresponding to each city;
- Initial state is the start city state;
- Goal state is the destination city state;
- Operators correspond to roads:
- there is an operator "$city_a \rightarrow city_b$"
- Iff there is a road from $city_a$ to $city_b$.

Example: Route Finding *(cont'd)*

– **Initial state is *Arad*; Goal state is *Bucharest*.**

Partial search tree:



The final search tree shows six partial solutions (open search nodes).

What is a solution?

- A solution to a search problem is a sequence of operators that generate a path from the initial state to a goal state.
- An optimal solution is a minimal cost solution.
- Solution cost versus search cost.

Review of State Space Terminology

- State space**: a *graph* of the set of states reachable from the initial states via some operator sequence.**

  (The state space is sometimes also called the *problem space* or the *search space*.)
- Path**: a sequence of operators from one state to some other state.**
- Solution**: a *path* from an initial state to a goal state.**
- Partial solution**: a *path* from an initial state to a (non-dead-end) intermediate state.**
  - Encompasses family of possible solutions
- Goal test: **predicate that tests if a state is a goal state (goal states may be explicitly listed or specified by a property).**
- Path cost**: function that assigns a cost to a path (often denoted g).**
- **It is the *sum* of costs of the operators/actions of the path.**

Searching for Solutions (*con'd.)*

- Search tree**: tree (or graph) of states (really *nodes*) explored by the search process.**
  - search tree (or search graph) is a *subgraph* of the state space.
- **Search involves maintaining and incrementally extending a *set* of *partial solutions*.**
- **We refer to these partial solutions as search** nodes **(nodes in the search tree).**
- **The process of extending a partial solution is called** expanding **a node.**
  - Basically, expanding a node involves using all/? operators applicable to the latest state of the node to identify reachable states and so generate new partial solutions (nodes).
  - It is common to refer to nodes by their latest states, but a node really represents a partial solution (operator sequence).

Problem Formalization Issues

- **Key issues in defining states:**
  - which objects/relations to represent;
  - which configurations need to be mapped into separate states.
- **Key issues in defining operators:**
  - may have to make explicit, unstated assumptions in the problem description;
  - how state-specific/general should operators be;
  - how much domain-specific knowledge should be "compiled" into the operators.
- **Developing an effective state space representation of a problem is choosing an appropriate** abstraction**.**
  - Without abstraction, agents would be swamped by the details of the real-world.

Abstraction

- **There are two main aspects of abstraction:**
  - removing unnecessary detail from the state descriptions (and so the operators);
  - removing legal operators that are useless or inefficient for achieving goals.
- **A good abstraction:**
  - removes as much detail as possible to make it easy enough to find a solution;
  - maintains the **validity** of the solutions (for the conceptual goals).
- **An *abstract solution* represents a large number of detailed paths.**
- **Often there is a trade-off between simplicity and generality (the representation becomes so specific to the given problem that it cannot be used for even very similar problems).**

Abstraction Examples

Two standard AI search problems can be used to explore the concept of abstraction.

<u>Missionaries and Cannibals:</u>

Three missionaries and three cannibals are on one side of a river. There is a boat available that can hold up to two people and can be used to cross the river. If the cannibals ever outnumber the missionaries in any location then a missionary will get eaten. Determine how the boat can be used to safely carry all the missionaries and cannibals across the river.

<u>Trip/route Planning:</u>

Determine how to get from one location to another. Assume that you know what city you are in, and have a map and a car.

Missionaries and Cannibals

Straightforward representation of states:
(boat-loc,m1-loc,m2-loc,...,c3-loc)
[loc $\in$ {side1,side2,river/boat}].

Results in 3^7 = 2187 states.

Can you simplify by abstraction?

Missionaries and Cannibals (*cont'd.*)

Abstraction Simplification
- the particular missionaries and cannibals on each side do not matter—only numbers;
- do not have to have explicit states with people in the boat (once in boat will only want to cross to other side);
- now once know number of a type on one side know number on the other side.

Abstract states:

(boat-side1?,#m's-side1,#c's-side1)

Results in 2 $\times$ 4 $\times$ 4 = 32 states.

Missionaries and Cannibals (*cont'd.*)

State abstraction also usually reduces the number of operators:

"move 1 m and 1 c from side1 to side2"

vs. the previous

"move m1 and c1 from side1 to side2,"

"move m1 and c2 from side1 to side2," etc.

I

Missionaries and Cannibals (*cont'd.*)

Useless operators can also be removed:

(1,m,c) → (2,m- 1,c)

[single missionary goes to goal side in boat].

The abstract solution using "move number of people" operators

is still a valid solution to the conceptual goal

(simply have to randomly select particular people when executing).

Trip/Route Planning

- In its full generality, states for this problem would be very complex since they would describe "complete" configurations of the world:

  "at latitude and longitude x-y, time is t, radio on, raining, car z meters ahead, etc."

- To simplify, we focus on the problem of finding a sequence of city to city traversals that accomplish the goal.

  In this case, our abstract states simply become: "in city x."

- We can further simplify by identifying important cities (i.e., major cities and cities with road junctions) and identifying the subset of relevant cities (we don't need to include Amherst in the state space if we are trying to get to Boston from Worcester).

Trip/Route Planning (*cont'd.*)

- Likewise, in its full generality, there would be a very large number of operators to be considered and it would take a very large number of operators to achieve a solution:
  - e.g., "go heading *h* at speed *s*," "turn radio on," etc.
- With the abstract states, operators are of the form: "go from city *a* to city *b*" [where there is a road from city *a* to city *b*].
- A solution to the abstract problem solves the basic goal, but does not give us the detail required for, say, a robot vehicle to actually navigate the trip.
- Still, the abstract problem solution allows us to see if a solution is even possible and to judge its approximate cost.

Searching for solutions

Search control strategies

- **Order in which the search space is explored**
- **The extent to which partial solutions are kept and used to guide the search process**
- **The degree to which the search process is guided by domain knowledge**
- **The degree to which control decisions are made dynamically at run-time**

Basic to All Search Control

- **Choose state(s) to expand next**
- **Choose operator(s) to expand the state(s)**
- **Execute the set of (state, operator) pairs**
- **Update the search graph with some of the new states created**
- **Decide if search should be terminated**
  - Finding an acceptable solution
  - There are no solutions to the problem
    - Within resource constraints

Search Strategies

- **A key issue in search is limiting the portion of the state space that must be explored to find a solution.**
- **The portion of the search space explored can be affected by the order in which states (and thus solutions) are examined.**
- **The search strategy determines this order by determining which node (partial solution) will be expanded next.**

Searching for solutions (*cont'd*)

- The extent to which partial solutions are kept and used to guide the search process
- The degree to which the search process is guided by domain knowledge
- The degree to which control decisions are made dynamically at run-time

Evaluation of Search Strategies

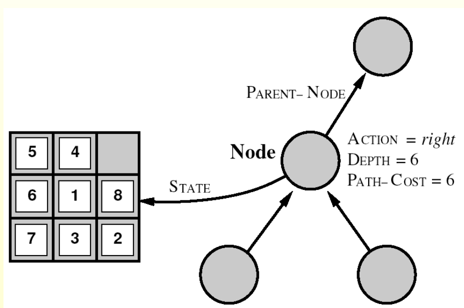
- **Completeness** - does it guarantee to find a solution when there is one?
- **Time complexity** - how long does it take to find a solution?
- **Space complexity** - how much memory does it require?
- **Optimality** - does it return the best solution when there are many?

Search trees

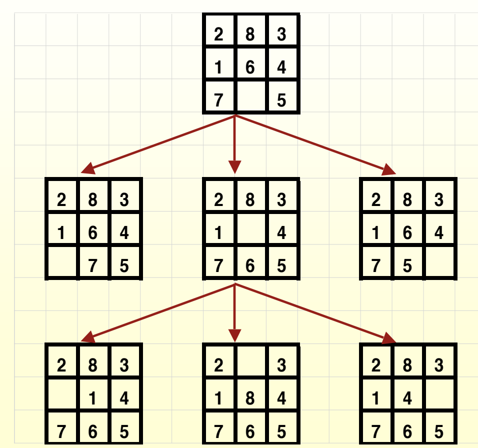
- A **search tree** is a graph representing the search process
- **Nodes** are the data structures from which the search tree is constructed
- **Implicit graphs** and **explicit graphs**
- **Branching factor** and **solution depth**
- **Generating** and **expanding** states
- **Open** and **closed** lists of nodes

Representing a node

```
(defstructure node
  state
  parent-node
  operator
  depth
  path-cost)
```



Example of a search tree



Tree search algorithms

- **Basic idea: offline, simulate exploration of state space by generating successors of already-explored states**

```
function TREE-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 Strategy Classification

- **Search strategies can be classified in the following general way:**
  – Uninformed/blind search;
  – Informed/heuristic search;
  – Multi-level/multi-dimensional/multi-direction;
  – Game/Adversarial search
    - Game search deals with the presence of an opponent that takes actions that diminish an agent's performance (see AIMA Chapter 6).

Blind search strategies

Blind or uninformed strategies use only the information available in the problem definition

– Breadth-first search (open list is FIFO queue)
– Uniform-cost search (shallowest node first)
– Depth-first search (open list is a LIFO queue)
– Depth-limited search (DFS with cutoff)
– Iterative-deepening search (incrementing cutoff)
– Bi-directional search (forward and backward)

Depth-first search

```
(defun dfs (nodes goalp successors)
 (cond
  ((null nodes) nil)
  ((funcall goalp (first nodes))
   (first nodes))
  (t (dfs (append (funcall successors
   (first nodes)) (rest nodes))
  goalp successors))))
```

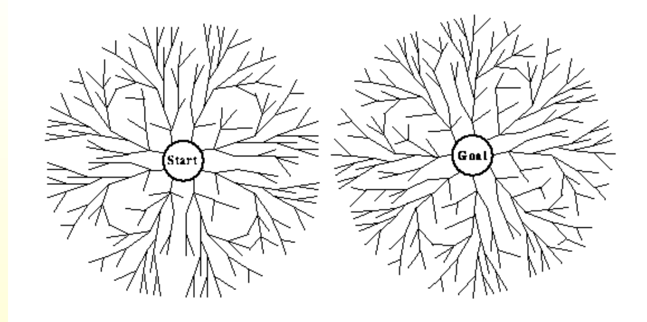
- **Time and space complexity?**

Breadth-first search

```
(defun bfs (nodes goalp successors)
 (cond
  ((null nodes) nil)
  ((funcall goalp (first nodes))
   (first nodes))
  (t (bfs (append (rest nodes)
 (funcall
   successors (first nodes)))
  goalp successors))))
```

- **Time and space complexity?**

Bidirectional search

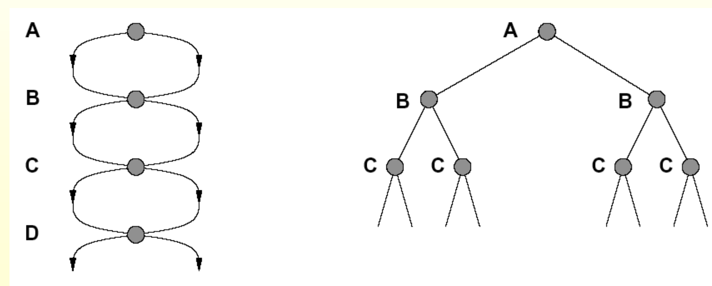
- 

Comparing search strategies

Breadth-first	Depth-first	Depth-limited	Iterative deepening	Bi-directional
Time $O(b^{d+1})$	Time $O(b^m)$	Time $O(b^l)$	Time $O(b^d)$	Time $O(b^{d/2})$
Space $O(b^{d+1})$	Space $O(bm)$	Space $O(bl)$	Space $O(bd)$	Space $O(b^{d/})^2$
Optimal? Yes*	Optimal? No	Optimal? No	Optimal? Yes*	Optimal? Yes*
Complete? Yes*	Complete? No	Complete? Yes* (if $l \geq d$)	Complete? Yes*	Complete? Yes*

Repeated states

Failure to detect repeated states can turn a linear problem into an exponential one!



Avoiding Repeated States

- Do not re-generate the state you just came from.
- Do not create paths with cycles.
- Do not generate any state that was generated before (using a hash table to store all generated nodes)

Problem Solving by Search

There are four phases to problem solving :

1. Goal formulation
- based on current world state, determine an appropriate **goal**;
- describes desirable states of the world;
- goal formulation may involve general goals or specific goals;

2. Problem formulation
- formalize the problem in terms of states and actions;
- **state space representation**;

3. Problem solution via search
- find sequence(s) of actions that lead to goal state(s);
- possibly select "best" of the sequences;

4. Execution phase
- carry out actions in selected sequence.

Agent vs. Conventional AI View

- **A completely autonomous agent would have to carry out all four phases.**
- **Often, goal and problem formulation are carried out prior to agent design, and the "agent" is given specific goal *instances* (agents perform only search and execution).**
  - general goal formulation, problem formulation, specific goal formulation, etc.
- **For "non-agent" problem solving:**
  - a solution may be simply a specific goal that is achievable (reachable);
  - there may be no execution phase.
- **The execution phase for a real-world agent can be complex since the agent must deal with uncertainty and errors.**

Goals vs.Performance Measures (PM)

- **Adopting goals and using them to direct problem solving can simplify agent design.**
- **Intelligent/rational agent means selecting best actions relative to a PM, but PMs may be complex (multiple attributes with trade-offs).**
- **Goals simplify reasoning by limiting agent objectives (but still organize/direct behavior).**
- Optimal **vs.** satisficing **behavior: best performance vs. goal achieved.**
- **May use both: goals to identify acceptable states plus PM to differentiate among goals and their possible solutions.**

Problem-Solving Performance

- **Complete search-based problem solving involves both the search process and the execution of the selected action sequence.**
  - **Total cost** of search-based problem solving is the sum of the search costs and the path costs (operator sequence cost).
- **Dealing with total cost may require:**
  - Combining "apples and oranges" (e.g., travel miles and CPU time)
  - Having to make a trade-off between search time and solution cost optimality (resource allocation).
  - These issues must be handled in the performance measure.

Knowledge and Problem Types

- **Problems can vary in a number of ways that can affect the details of how problem-solving (search) agents are built.**
- **One categorization is presented in AIMA: (related to accessibility and determinism)**
  - Single-state problems
    - Agent knows initial state and exact effect of each action;
    - Search over single states;
  - Multiple-state problems
    - Agent cannot know its exact initial state and/or the exact effect of its actions;
    - Must search over state sets;
    - May or may not be able to find a guaranteed solution;

Knowledge and Problem Types *(cont'd)*

- Contingency problems
  - Exact prediction is impossible, but states may be determined during execution (via sensing);
  - Must calculate tree of actions, for every contingency;
  - **Interleaving** search and execution may be better;
- Exploration problems
  - Agent may have no information about the effects of its actions and must experiment and learn
  - Search in real world vs. model.

Next lecture

- **Continuation of Simple Search**
  - How to use heuristics (domain knowledge) in order to accelerate search?
  - Reading: Sections 4.1-4.2.
- **Characteristics of More Complex Search**
  - Subproblem interaction
  - More complex view of operator/control costs
  - Uncertainty in search
  - Non-monotonic domains
  - Search redundancy