Problem Solving Using Search

Sattiraju Prabhakar
CS771: Classes 4, 5, 6
Wichita State University

Topics
• Problem Solving Agents
• Example Problems
• Searching for Solutions
• Uninformed Search Strategies

Goals and Problem Solving
• Intelligent Behavior:
  – Agents are supposed to maximize their performance measure
• An approach to intelligent behavior:
  – By trying to achieve a set of states in environment
• Goals:
  – A pre-specified set of environmental states that can be recognized by the agent as desirable.
• Types of Goals:
  – A goal can be an environmental state, or
  – It can be a property of an environmental state
What is Problem Solving?

- What is a problem for an intelligent agent?
  - To achieve a goal state in the environment
- Problem Specification: (More complete on page 7)
  - Stat State: The present state of the environment
  - Goal State: The desirable state or property of the states in the environment
- Problem Solving:
  - Finding a sequence of actions that will allow the agent to transform the start of the environment to a goal state

Some Definitions

- Goal Formulation:
  - Generating the goal states from the current state
- Problem Formulation:
  - Given a goal, considering what actions and states to consider
- Search:
  - The process of looking for sequences of actions that will transform the start to goal state

Problem Formulation

- Start State
- Goal State
- All relevant states: This is called the state space
- All relevant actions

Simple Problem Solving Agent

```python
def SIMPLE_PROBLEM_SOLVING_AGENT(state, goal):
    # todo
```

Figure 3.2
### Problem Solving Agents

#### Example Problems

#### Searching for Solutions

##### Uninformed Search Strategies

---

**Vacuum World**

- **States:** (see figure)
  - The agent can be in one of 2 locations
  - For each, the dirt can be in either one of the locations
  - When the agent is in location, where the dirt is, it can lead to no dirt state
- **Initial State:** Can be any one of these states
- **Successor Function:** This generates the states resulting from applying an action on a state (see figure)
- **Goal Test:** This checks if all the squares are clean
- **Path Cost:** Each step costs 1. Total cost is the number of steps

---

**State Space for Vacuum World**

---

**8-Puzzle Problem**

- **States:** Each state is a location of eight tiles and the blank tile
- **Initial State:** Any state can be designated
- **Actions:** (Left, Right, Up, Down)
- **Successor Function:** Generates legal states for each application of an action
- **Goal Test:** Checks to see if a tile configuration matches the goal configuration
- **Path Cost:** Each step costs 1.
8-Puzzle Problem

Start State

Goal State

8-queens Problem

- States: Any arrangement of 0-8 queens on the board
- Initial State: No queens on the board
- Successor Function: Add a queen to any empty square
- Goal Test: 8 queens on the board and none attacked

Route-Finding Problem

- States: Each state has a location and time
- Initial State: This can be anything
- Successor Function:
  - Actions: Taking taxi, Walking, Flying
  - Consequent locations
- Goal Test: Destination and at a time
- Path Cost: Monetary value of each action
Touring Problem

On holiday in Romania; currently in Arad.
Flight leaves tomorrow from Bucharest.

Formulate goal:
be in Bucharest

Formulate problem:
states: various cities
actions: drive between cities

Find solution:
sequence of cities, e.g., Arad, Sibiu, Fagaras, Bucharest

Path Costs

<table>
<thead>
<tr>
<th>City</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arad</td>
<td>0</td>
</tr>
<tr>
<td>Bucharest</td>
<td>0</td>
</tr>
<tr>
<td>Craiova</td>
<td>160</td>
</tr>
<tr>
<td>Dobrota</td>
<td>242</td>
</tr>
<tr>
<td>Eforie</td>
<td>161</td>
</tr>
<tr>
<td>Fagaras</td>
<td>176</td>
</tr>
<tr>
<td>Giurgiu</td>
<td>77</td>
</tr>
<tr>
<td>Hiersova</td>
<td>151</td>
</tr>
<tr>
<td>Iasi</td>
<td>226</td>
</tr>
<tr>
<td>Lugoj</td>
<td>244</td>
</tr>
<tr>
<td>Mehadia</td>
<td>241</td>
</tr>
<tr>
<td>Neamt</td>
<td>0</td>
</tr>
<tr>
<td>Oradea</td>
<td>380</td>
</tr>
<tr>
<td>Pitesti</td>
<td>100</td>
</tr>
<tr>
<td>Rimnicu Vilcea</td>
<td>193</td>
</tr>
<tr>
<td>Sibiu</td>
<td>253</td>
</tr>
<tr>
<td>Timisoara</td>
<td>329</td>
</tr>
<tr>
<td>Uziceni</td>
<td>80</td>
</tr>
<tr>
<td>Vaslui</td>
<td>199</td>
</tr>
<tr>
<td>Zerind</td>
<td>374</td>
</tr>
</tbody>
</table>

Start and Goal States
Problem Solving Agents
Example Problems
Searching for Solutions
Uninformed Search Strategies

Review (I)

• Our focus is upon goal based agents
  – Goal based agents select actions to arrive at states that satisfy a goal test
  – These are much more efficient than simple reflex agents
• Internal States:
  – Goal based agents have internal state(s) that corresponds to various state(s) of environment.
  – Agent has Successor Function that allows it to compute the consequences of its actions in the environment
    • Without a successor function, it is not possible for the agent to know what will happen in the environment

Review (II)

• Showing Internal States:
  – In various forms
• We can describe one kind of performance of agents using Problem Solving
  – Problems: Problems need to be specified using:
    • {Start State, States, Actions, Successor Function, Goal Test, Path Cost}
  – Solutions to problems:
    • Searching in the states for a state that satisfies goal test

Tutorial: What is an Internal State?

• Is it a percept?
• Is it a description of the agent?
• Is it a description of the environment?
• What is a state?
  – Is it a complete description of the behavior of the environment?
  – Is it a snapshot of the environment?
• Why is it useful?
  – Does it tell the agent what the environment is doing at a time?
  – Does it tell the agent what to do?
**Tutorial: Agents without internal states?**

- For Vacuum Cleaner agent, what is an internal state?
- Draw a Vacuum Cleaner Agent without an internal state.
  - What happens when the agent has to keep the locations A and B 100% clean, and 10% of the time the camera shuts-off due to overheating
  - A vacuum cleaner agent has a cheap camera. It takes 10 milliseconds to generate an image. But it has powerful cleaner which sucks without any delay
  - Show your answers in the form of state diagram

---

**Tutorial: Showing Internal States**

- For the following agents show what the internal states look like.
  - Vacuum Cleaner Agent
  - Touring Agent (See map of Romania)
  - 8-puzzle problem solver
  - 8-queens puzzle solver
  - Missionaries-Cannibals problem solver
  - Monkey-Banana Problem solver

---

**Tutorial: Computing Environmental States**

- What is a Successor Function?
- For the following agents, write successor functions: (use state diagrams)
  - Vacuum Cleaner Agent
  - 8-puzzle problem
  - Tic-Tac-Toe
  - Missionaries Cannibals problem
  - Touring problem

---

**Tutorial: Problem Specification**

- Write problem specifications for the following problems:
  - Tic-Tac-Toe
  - Water-Jug Problem
  - Missionaries-Cannibals Problem
  - Monkey-Banana Problem
  - Touring Problem
**Tutorial: Algorithms and Agents**

- What are the differences:
  - What is a problem in Algorithms and what is a problem in Agents?
    - Why are they different?
  - What does an Algorithm do for finding a solution?
    - What does an Agent do for finding a solution?

---

**Why Search?**

- Objective:
  - To make an agent find a solution for the problem
- What is a solution for an agent?
  - It is a means by which the agent starts at the start state in environment and reaches the goal state.
  - This means is a sequence of actions
- By making the agent find this solution, we are making it perform well in the environment as it should

---

**What is Search?**

- Agent searches in states
  - States corresponds to all possible states of the environment
- Search explores States:
  - Search moves from state to state in States until goal test is satisfied
  - This moving from state to state is enabled using actions (successor function)
- Issues:
  - On every state, you can apply a number of actions

---

**Tutorial: On every state you can apply a number of actions**

- For the following agents, show for at least one state, what are the different applicable actions
  - Vacuum cleaner agent
  - 8-puzzle problem
  - touring problem
  - missionaries-cannibals problem
Tree Search

Basic idea:
- offline, simulated exploration of state space
- by generating successors of already-explored states
  (a.k.a. expanding states)

Function TreeSearch(problem, strategy) returns a solution, or failure
initializes 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

Tree Search Example: Start at the Start State of the Problem

Tree Search Example: Expand node using Successor Function

Tree Search Example: Select a node and expand
**Tutorial: Tree Search**

- For the following problems, generate search tree
  - 8-puzzle problem
  - Vacuum cleaner agent
  - Blocks world problem

**Implementation: States vs Nodes**

A state is a (representation of) a physical configuration.
A node is a data structure comprising part of a search tree.
   - includes parent, children, depth, path cost (g(x))
   - states do not have parent, children, depth, or path cost

```
State Space
```

The `EXPAND` function creates new nodes, filling in the various fields and using the `SUCCESSOR-FN` of the problem to create the corresponding states.

**Implementation general tree search**

```
function TREE-SEARCH(problem, fringe) returns a solution, or failure
    fringe = INSERT(MAKE-NODE(INITIAL-STATE(problem)), fringe)
    loop do
        if fringe is empty then return failure
        node = extracts from fringe
        if GOAL-TEST(node) succeeds then return node
        fringe = INSERT(ALL-EXPAND(node, problem), fringe)

function EXPAND(node, problem) returns a set of nodes
    add to fringe the set:
    for each action, result = SUCCESSOR-FN(problem)(STATE[node]), do
        add to fringe the node:
        STATE[new] = result
        fringe = fringe + [MAKE-NODE(result, ACTION, node, problem)]
    add to fringe: fringe + [MAKE-NODE(result, ACTION, parent, problem)]
```

**Search Spaces and General Search**

```
Search Space
```

States: State_1, State_2, State_3, State_4, State_5

Start State: State_1

Goal State: State_5
Problem Solving Agents
Example Problems
Searching for Solutions
Uninformed Search Strategies

Search Strategies

A strategy is defined by picking the order of node expansion

Strategies are evaluated along the following dimensions:
  completeness—does it always find a solution if one exists?
  time complexity—number of nodes generated/expanded
  space complexity—maximum number of nodes in memory
  optimality—does it always find a least-cost solution?

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 $\infty$)

Uninformed search

Uninformed strategies use only the information available in the problem definition

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

Breadth First search

Expand shallowest unexpanded node

Implementation:

$\text{fringe}$ is a FIFO queue, i.e., new successors go at end

\begin{tikzpicture}
  \node (A) {$A$} child {node (B) {$B$}} child {node (C) {$C$}};
  \node (D) {$D$} child {node (E) {$E$}} child {node (F) {$F$}} child {node (G) {$G$}};
\end{tikzpicture}
Breadth first search

Expand shallowest unexpanded node

Implementation:
fringe is a FIFO queue, i.e., new successors go at end

Breadth First Search

Expand shallowest unexpanded node

Implementation:
fringe is a FIFO queue, i.e., new successors go at end

Complexity Analysis

• Search Cost (Time cost):
  – Amount of time taken for search
• Path Cost, g(n): The cost of path from initial node to the goal node
• Total Cost:
  – Search Cost + Path Cost
• Space Complexity: How much memory needed to perform search
Properties of BFS

• Optimal:
  – If the path cost is a non-decreasing function of the depth of the node

• Time Complexity Analysis:
  – The number of nodes at level $i$, $b^i$
  – When the goal is found at level (depth) $d$, the nodes expanded are: $b^{d+1}$
  – Worst case, the last node to be expanded at level $d$ is the goal node
    • Nodes in level $d+1$, $b^{d+1} - b$

Properties of BFS

• Search Cost:
  – $= 1 + b + b^2 + b^3 + \ldots + b^d + (b^{d+1} - b) = O(b^{d+1})$

• Space Complexity:
  – Every node in fringe should remain in the memory:
    • Space complexity is same as time complexity

• Memory Requirements are a big problem
  • For large $d$, the time taken is very large

Tutorial

• Solve the following problems by developing the search tree by BFS. Write the queue at different stages
  – Touring problem
  – 8-puzzle

Depth-First Search

Expand deepest unexpanded node

Implementation:

$\text{fringe} = \text{LIFO queue, i.e., put successors at front}$

Diagram: Depth-First Search with nodes and edges.
Depth-First Search

Expand deepest unexpanded node

Implementation:

\[ fringe = \text{LIFO queue}, \text{i.e., put successors at front} \]
Depth-First Search

Expand deepest unexpanded node

**Implementation:**
 fringe = LIFO queue, i.e., put successors at front

[Diagram of a tree with nodes and edges, illustrating the depth-first search process.]
Depth-First Search
Expand deepest unexpanded node

Implementation:

\[ \text{fringe} = \text{LIFO queue}, \text{i.e., put successors at front} \]

Properties of Depth-First

Complete?? No: fails in infinite-depth spaces, spaces with loops
Modify to avoid repeated states along path
\Rightarrow complete in finite spaces

Time?? \( O(b^d) \): 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!

Optimal?? No
Tutorial

• Using Depth First Search Strategy, generate search trees for the following problems
  – Vacuum Cleaner agent
  – Blocks world problem

Depth-Limited Search

= depth-first search with depth limit l, i.e., nodes at depth l have no successors

Recursive implementation:

```
function DEPTH-LIMITED-SEARCH(problem, limit) returns solution/or failure
  RECURSE-DEEP-LIMITED-NODE(INITIAL-STATE(problem), limit)

function RECURSE-DEEP-LIMITED-NODE(node, limit) returns solution/or failure
  cutoff := false
  if GOAL-TEST(node) then return node
  else if Depth(node) = limit then return failure
  else for each successor s in EXPAND(node, problem) do
    result := RECURSE-DEEP-LIMITED-NODE(s, limit)
    if result := cutoff then cutoff := true
    else if result /= failure then return result
    if cutoff = cutoff then return cutoff else return failure
```

Iterative Deepening Depth-First Search

```
function_ITERATIVE-DEEPENING-SEARCH(problem) returns a solution
for depth := 0 to oc do
  result := DEPTH-LIMITED-SEARCH(problem, depth)
  if result /= cutoff then return result
end
```

Iterative-deepening search

Limit = 0
Iterative-deepening search

Properties

- Complete? Yes
- Time? \( (d+1)b^0 + db^1 + (d-1)b^2 + \ldots + b^d = O(b^d) \)
- Space? \( O(bd) \)
- Optimal? Yes, if step cost = 1
  - Can be modified to explore uniform-cost tree

Numerical comparison for \( b = 10 \) and \( d = 5 \), solution at far right:

\[
N(\text{IDS}) = 50 + 400 + 3,000 + 20,000 + 100,000 - 123,456
\]
\[
N(\text{BFS}) = 10 + 100 + 1,000 + 10,000 + 100,000 + 999,990 = 1,111,100
\]
## Summary of Properties

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Breadth-First</th>
<th>Uniform-Cost</th>
<th>Depth-First</th>
<th>Depth-Limited</th>
<th>Iterative Deepening</th>
</tr>
</thead>
<tbody>
<tr>
<td>Completeness</td>
<td>Yes*</td>
<td>Yes*</td>
<td>No</td>
<td>Yes, if ( I \geq 4 )</td>
<td>Yes</td>
</tr>
<tr>
<td>Time</td>
<td>( O(n^2) )</td>
<td>( O(N) )</td>
<td>( O(n^2) )</td>
<td>( \frac{I}{I^d} )</td>
<td>( \frac{I}{I^d} )</td>
</tr>
<tr>
<td>Space</td>
<td>( \frac{I}{I^d} )</td>
<td>( O(C/\lambda) )</td>
<td>( I )</td>
<td>( \lambda )</td>
<td>( \lambda )</td>
</tr>
<tr>
<td>Optimal?</td>
<td>Yes*</td>
<td>Yes*</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>