CSCI 360
Introduction to Artificial Intelligence

Instructor: Wei-Min Shen
Status Check and Review

• Status check
  – Have you registered in Piazza?
  – Have you run the Project-1?
  – Have you submitted your pre-reading reports?
    • Here are some good ones for your reference

• Review of last lecture
  – Agent? Types of agents?
  – Environment? Types of environment?

• Today’s lecture = ?
How to represent a problem?
  – states, actions, initials, goals, (e.g., tic-tac-toe)

How to solve a problem?
  – Search: from here to there, initials to goals
  – Depth-first, breadth-first

How good is your solution? (fig 6.1, ALFE)
  – How good is your state? How costly is an action?
  – Best-first, Dynamic Programming, A*, etc.
  – Can you guarantee anything? (optimal vs heuristic)

How much do you want to pay for your solution?
  – How deep/wide can you go?
    – Predetermined vs dynamic (e.g., iterative deepening)
  – One way or many ways (bi-directional)?

How big is a problem? Can you put the whole world in your head?
  – Tower of Hanoi, chess, robot-and-world,

HM: state space for TOH, assign values for state and actions
How to find the extremes of \( f(x) \)?

Why is it so important?

Why is it so hard? (no one has offered a general solution!)
  - Which way to go?
  - How much can you see? (local vs global)
  - How many points can you remember?
  - How big is your step? (skip THE point?)
  - How well can you guess?
  - How much do you know about the function?
  - How do you know you are done?
  - Will the function change by itself?

HW: your own answers for the above questions
This Week’s Lecture

• Problem Solving and Search Techniques
• Optimization Techniques
• Home Work 1 assignments
Solving Problems by Search

• How to represent a real-world problem?
  – Examples, state space, states, actions, initials, goals.
• How to solve a problem?
  – Search: from here to there, initials to goals
  – Depth-first, breadth-first
• How good is your solution? (fig 6.1, ALFE)
  – How good is your state? How costly is an action?
  – Best-first, Dynamic Programming, A*, etc.
  – Can you guarantee anything? (optimal vs heuristic)
• How much do you want to pay for your solution?
  – How deep/wide can you go?
    • Predetermined vs dynamic (e.g., iterative deepening)
  – One way or many ways (bi-directional)?
• How big is a problem? Can you put the whole world in your head?
  – Tower of Hanoi, chess, robot-and-world,
What is a “Problem”?

• Any ideas?
• Hint: Think it as related to Agent
What is a “Problem”? 

- Any ideas?
- Hint: Think it as related to Agent
  - Percepts, actions, environment, goals, utilities
- Problem:
  - States, initial state, goal states,
  - Solution: a sequence of actions leads to the goal
Problem Examples: 8Puzzle, PAC-Man

Initial State

Goal State(s)
Tower of Hanoi, A Cleaning Robot

Can “dirt” be a goal?
Travel Example and Project-1
## State Space

**How to represent a real-world problem**

<table>
<thead>
<tr>
<th><strong>Graph Search</strong></th>
<th><strong>Search Problem</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>- Graph</td>
<td>- State space</td>
</tr>
<tr>
<td>- Vertices</td>
<td>- States</td>
</tr>
<tr>
<td>- Edges</td>
<td>- Actions=operators</td>
</tr>
<tr>
<td>- Start vertices</td>
<td>- Initial state</td>
</tr>
<tr>
<td>- Goal vertices</td>
<td>- Goal states or goal test</td>
</tr>
<tr>
<td>- Solution is a (minimum cost) path from the start to the goal</td>
<td>- Solution is a (minimum cost) action sequence from the start to the goal state (or a state that satisfies the goal test)</td>
</tr>
</tbody>
</table>

A BIG Question: How do the “sensors” and “percepts” enter the picture?
Vacuum Robot and Its World

**states??**: integer dirt and robot locations (ignore dirt amounts etc.)

**actions??**: Left, Right, Suck, NoOp

**goal test??**: no dirt

**path cost??**: 1 per action (0 for NoOp)
Traveling from Arad To Bucharest

What is a “state” here?
State Graphs vs. Search Trees

Each NODE in the search tree is an entire PATH in the state graph (note how many nodes in the graph appear multiple times in the search tree)

We almost always construct both on demand – and we construct as little as possible
State Space and Search Tree

A Search Tree

State space with actions, costs/rewards (Fig6.1, ALFE)

Terminology:
- node (parent node, child node)
- leaf
- fringe = frontier
- depth
- node expansion
Search Tree Nodes

• A **node** in the search tree is a data structure containing the state plus other data
Solving Problems by Search

• How to represent a problem?
  – Examples (e.g., 8puzzle, TOH, Roomba, Travel), states, actions, initials, goals.

• How to solve a problem?
  – Search: from here to there, from the initials to the goals
  – Depth-first, breadth-first

• How good is your solution? (fig 6.1, ALFE)
  – How good is your state? How costly is an action?
  – Best-first, Dynamic Programming, A*, etc.
  – Can you guarantee anything? (optimal vs heuristic)

• How much do you want to pay for your solution?
  – How deep/wide can you go?
    • Predetermined vs dynamic (e.g., iterative deepening)
  – One way or many ways (bi-directional)?

• How big is a problem? Can you put the whole world in your head?
  – Tower of Hanoi, chess, robot-and-world,

• HM: state space for TOH, assign values for state and actions.
Problem Solving Methods (Overview)

- Breadth-first search
- Depth-first search
- Best-first search
  - Goodness = (past + future)
  - Uniform cost (estimate past and future)
  - A* search algorithm (estimate the future)
  - Dynamic Programming (back from the future)
Tree search algorithms

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

```
function TREE-SEARCH(problem, strategy)
  returns a solution, or failure

  initialize the search tree using the initial state 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 resulting nodes to the search tree
  end
```

**Breath-first strategy:** the least depth node

**Depth-first strategy:** the deepest depth node
function TREE-SEARCH(problem) return a solution or failure

frontier ← MAKE-QUEUE(MAKE-NODE(problem.INITIAL-STATE))

loop do
  if EMPTY?(frontier) then return failure
  node ← CHOOSE-NODE-By-Strategy(frontier)
  if problem.GOAL-TEST applied to node.STATE succeeds
    then return SOLUTION(node)
  frontier ← INSERT-ALL(EXPAND(node, problem), frontier)

• The frontier is the set of nodes that have been generated but not yet expanded
Strategy: Breadth-First Search

- Expand **shallowest** nodes first
  - Left to right at any depth
Breath-first Search example

State Graph
Depth-First Search

- Expand **deepest** nodes first
  - Left to right at any depth
Depth-First Search

- Expand deepest nodes first
  - Left to right at any depth
  - Backtrack when hit “terminal”
Depth-first Search example

State Graph

Question:
Will the depth grow forever?
Where is the “terminal”?
Loop Problems

Figure 3.2  A simplified road map of part of Romania.
Graph Search (Preventing Loop)

function GRAPH-SEARCH(problem) return a solution or failure

frontier ← MAKE-QUEUE(MAKE-NODE(problem.INITIAL-STATE))
explored_set ← empty

loop do
  if EMPTY?(frontier) then return failure
  node ← CHOOSE-NODE-By-Strategy(frontier)
  if problem.GOAL-TEST applied to node.STATE succeeds
    then return SOLUTION(node)
  explored_set ← INSERT(node, explored_set)
  for each new_node in EXPAND(node, problem) do
    if NOT(MEMBER?(new_node, frontier)) and
      NOT(MEMBER?(new_node, explored_set))
    then frontier ← INSERT(new_node, frontier)
Solving Problems by Search

- How to represent a problem?
  - Examples (e.g., 8puzzle, TOH, Roomba, Travel), states, actions, initials, goals.
- How to solve a problem?
  - Search: from here to there, from the initials to the goals
    - Depth-first, breadth-first
- How good is your solution? (fig 6.1, ALFE)
  - How good is your state? How costly is an action?
  - Best-first, Dynamic Programming, A*, etc.
  - Can you guarantee anything? (optimal vs heuristic)
- How much do you want to pay for your solution?
  - How deep/wide can you go?
    - Predetermined vs dynamic (e.g., iterative deepening)
    - One way or many ways (bi-directional)?
- How big is a problem? Can you put the whole world in your head?
  - Tower of Hanoi, chess, robot-and-world,
- HM: state space for TOH, assign values for state and actions.
Best-First search algorithms

```plaintext
function TREE-SEARCH(problem, strategy)
    returns a solution, or failure
    initialize the search tree using the initial state 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 resulting nodes to the search tree
    end
```

**Best-first strategy**: choose the “best” node: goodness = (past cost + future cost)

When do we use this strategy?
Traveling from Arad To Bucharest

We have the knowledge about the actual driving distances
Best-first Search example

Choose the best node to expand: (past cost) + (the best future cost)

Sibiu: \((140) + (80 + 97 + 101) = ?\)

Timisoanra: \((118) + (111 + 70 + 75 + 120 + 138) = ?\)

Zerind: \((75) + (71 + 151 + 80 + 97 + 101) = ?\)
Two Big Questions about “Best”?

• What if the best future cost is unknown to us?
• What if the best future cost is too expensive to compute?
What if we don’t know the future?

• Goodness = past cost + the best future cost
  – What if the best future cost is unknown to us?
• Estimate Goodness: uniform cost (naïve)
• Estimate Goodness = past cost + estimated future cost
  – Evaluation function $f(n) = g(n) + h(n)$
    • $g(n)$ the cost (so far) to reach the current node $n$
    • $h(n)$ estimated cost to get from the current node $n$ to the goal
    • $f(n)$ estimated total cost of path through $n$ to the goal
Traveling from Arad To Bucharest

Suppose the future driving distances to the goal is unknown to us. But we know the straight-line distances to the goal.

Sensor: ?

<table>
<thead>
<tr>
<th>Place</th>
<th>Straight-line distance to Bucharest</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arad</td>
<td>366</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>178</td>
</tr>
<tr>
<td>Giurgiu</td>
<td>77</td>
</tr>
<tr>
<td>Hirsova</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>234</td>
</tr>
<tr>
<td>Oradea</td>
<td>380</td>
</tr>
<tr>
<td>Pitesti</td>
<td>98</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>Urziceni</td>
<td>80</td>
</tr>
<tr>
<td>Vaslui</td>
<td>199</td>
</tr>
<tr>
<td>Zerind</td>
<td>374</td>
</tr>
</tbody>
</table>
Using the estimated future cost

Choose the best node to expand: (past cost + estimated future cost)
Sibiu: $140 + 253$
Timisoanra: $118 + 329$
Zerind: $75 + 374$
What if our estimation is wrong?

Examples are plenty
   “My father told me so”
   “I did drive that road before”
   “I have an iPhone”
   “Google map”
   ...... 

How “wrong” can we tolerate?
   Over-estimate?
   Under-estimate?
   By how much?
Estimate by “Admissible Heuristics”

• A heuristic is admissible if it \textit{never overestimates} the future cost to reach the goal

• Admissible heuristics are “optimistic”

Formally:

1. \( h(n) \leq h^*(n) \) where \( h^*(n) \) is the true cost from \( n \) to the goal

2. \( h(n) \geq 0 \), with \( h(G)=0 \) for any goal \( G \)

E.g. \( h_{SLD}(n) \) (GPS) is admissible because it never overestimates the actual road distance to the goal
A* search algorithm

```plaintext
function TREE-SEARCH(problem, strategy)
    returns a solution, or failure

    initialize the search tree using the initial state 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 resulting nodes to the search tree
        end
    end

Best-first strategy: choose the best node: goodness = (past cost + future cost)
A* strategy: choose the “admissibly estimated best” node:
    goodness = (past cost + admissibly estimated future cost)
```
What if “future is too hard to compute”?

• Compute the best future cost for a node must consider all possible paths from the node to the goal
  – This is too expensive because there are so many such paths
• Solution
  – Use Dynamic Programming
    • Not compute path by path
    • But compute stage by stage (breath-first)
Dynamic Programming (ALFE 6.1.1)

Backward recursion: compute the future cost by back from the goal* stage by stage

\[ V(s_1) = \max \{ R(s_1, a_1) + V(s_4), R(s_1, a_2) + V(s_5) \} = \max \{ 1 + 2, 3 + 1 \} = 4 \]
\[ V(s_2) = \max \{ R(s_2, a_0) + V(s_4), R(s_2, a_1) + V(s_5), R(s_2, a_2) + V(s_6) \} \]
\[ = \max \{ 6 + 2, 2 + 1, 5 + 1 \} = 8 \]
\[ V(s_3) = \max \{ R(s_3, a_0) + V(s_5), R(s_3, a_1) + V(s_6) \} = \max \{ 2 + 1, 3 + 1 \} = 4 \]

\[ V(s_0) = \max \{ R(s_0, a_0) + V(s_1), R(s_0, a_1) + V(s_2), R(s_0, a_2) + V(s_3) \} \]
\[ = \max \{ 1 + 4, 2 + 8, 3 + 4 \} = 10 \]

Backward recursion equation:

\[ V(s_i) = \max_a \{ R(s_i, a) + V(s_j) \} \quad \text{where} \quad s_i \xrightarrow{a} s_j \]
Best-First by Dynamic Programming

function TREE-SEARCH(problem, strategy)

returns a solution, or failure

initialize the search tree using the initial state 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 resulting nodes to the search tree

end

Best-first strategy: choose the best node: goodness = (past cost + future cost)

A* strategy: choose the best node: goodness = (past cost + admissibly estimated future cost)

Dynamic programming: choose the best node: goodness = (past cost + V(s))
Evaluation of search strategies

- A search strategy is defined by picking the order of node expansion.

- Search algorithms are commonly evaluated according to the following four criteria:
  - **Completeness**: does it always find a solution if one exists?
  - **Time complexity**: how long does it take as function of number of nodes?
  - **Space complexity**: how much memory does it require?
  - **Optimality**: does it guarantee the least-cost solution?

- Time and space complexity are measured in terms of:
  - $b$ – max branching factor of the search tree
  - $d$ – depth of the least-cost solution
  - $m$ – max depth of the search tree (may be infinity)
Solving Problems by Search

• How to represent a problem?
  – Examples (e.g., 8puzzle, TOH, Roomba, Travel), states, actions, initials, goals.

• How to solve a problem?
  – Search: from here to there, from the initials to the goals
  – Depth-first, breadth-first

• How good is your solution? (fig 6.1, ALFE)
  – How good is your state? How costly is an action?
  – Best-first, Dynamic Programming, A*, etc.
  – Can you guarantee anything? (optimal vs heuristic)

• How much do you want to pay for your solution?
  – How deep/wide can you go?
    • Predetermined vs dynamic (e.g., iterative deepening)
  – One way or many ways (bi-directional)?

• How big is a problem? Can you put the whole world in your head?
  – Tower of Hanoi, chess, robot-and-world,

• **HM**: state space for TOH, assign values for state and actions.
Other Search-related Techniques

• Iterative deepening
  – Limit the search depth, incrementally increase it

• Bi-directional search
  – Search from the initial to the goal, as well as from the goal to the initial
Solving Problems by Search

• How to represent a problem?
  – Examples (e.g., 8puzzle, TOH, Roomba, Travel), states, actions, initials, goals.

• How to solve a problem?
  – Search: from here to there, from the initials to the goals
  – Depth-first, breadth-first

• How good is your solution? (fig 6.1, ALFE)
  – How good is your state? How costly is an action?
  – Best-first, Dynamic Programming, A*, etc.
  – Can you guarantee anything? (optimal vs heuristic)

• How much do you want to pay for your solution?
  – How deep/wide can you go?
    • Predetermined vs dynamic (e.g., iterative deepening, fixed depth)
  – One way or many ways (bi-directional)?

• How big is a problem? Can you put the whole world in your head?
  – Tower of Hanoi, chess, robot-and-world,

• HM: state space for TOH, assign values for state and actions.
Complexity: How big is a problem?

- 8-puzzle has 362,880 states
- 15-puzzle has $10^{12}$ states
- 24-puzzle has $10^{25}$ states

When you solve $N=64$, it would be the end of the world!

---

**breadth-first search**

<table>
<thead>
<tr>
<th>depth</th>
<th>nodes</th>
<th>time</th>
<th>memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1 millisecond</td>
<td>100 bytes</td>
</tr>
<tr>
<td>2</td>
<td>111</td>
<td>0.1 seconds</td>
<td>11 kilobytes</td>
</tr>
<tr>
<td>4</td>
<td>1,111</td>
<td>11 seconds</td>
<td>1 megabyte</td>
</tr>
<tr>
<td>6</td>
<td>$10^6$</td>
<td>18 minutes</td>
<td>111 megabytes</td>
</tr>
<tr>
<td>8</td>
<td>$10^8$</td>
<td>31 hours</td>
<td>11 gigabytes</td>
</tr>
<tr>
<td>10</td>
<td>$10^{10}$</td>
<td>128 days</td>
<td>1 terabyte</td>
</tr>
<tr>
<td>12</td>
<td>$10^{12}$</td>
<td>35 years</td>
<td>111 terabytes</td>
</tr>
<tr>
<td>14</td>
<td>$10^{14}$</td>
<td>3500 years</td>
<td>11,111 terabytes</td>
</tr>
</tbody>
</table>

$b = 10$; 1000 nodes per second; 100 bytes per node
Time complexity of depth-first

- In the worst case:
  - the (only) goal node may be on the right-most branch,

\[
\text{Time complexity} = b^d + b^{d-1} + \ldots + 1 = \frac{b^{d+1} - 1}{b - 1}
\]

Thus: \(O(b^d)\)
Space complexity of depth-first

- Largest number of nodes in QUEUE is reached in bottom left-most node.
- Example: $m = 3, \ b = 3$ :
  - QUEUE contains all red nodes. Thus: 7.
  - In General: $((b-1) \times m) + b$
  - Order: $O(m \times b)$
Comparing uninformed search strategies

<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 (if applicable)</th>
<th>Bidirectional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>$b^d$</td>
<td>$b^d$</td>
<td>$b^m$</td>
<td>$b^l$</td>
<td>$b^d$</td>
<td>$b^{(d/2)}$</td>
</tr>
<tr>
<td>Space</td>
<td>$b^d$</td>
<td>$b^d$</td>
<td>$bm$</td>
<td>$bl$</td>
<td>$bd$</td>
<td>$b^{(d/2)}$</td>
</tr>
<tr>
<td>Optimal?</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Complete?</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes, if $l \geq d$</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

$b$ – max branching factor of the search tree
$d$ – depth of the least-cost solution
$m$ – max depth of the state-space (may be infinity)
$l$ – depth cutoff
Summary

• Problem formulation usually requires abstracting away real-world details to define a state space that can be explored using computer algorithms.

• Once problem is formulated in abstract form, complexity analysis helps us picking out best algorithm to solve problem.

• Variety of uninformed search strategies; difference lies in method used to pick node that will be further expanded.

• Iterative deepening search only uses linear space and not much more time than other uniformed search strategies.
Solving Problems by Search

• How to represent a problem?
  – Examples (e.g., 8puzzle, TOH, Roomba, Travel), states, actions, initials, goals.
• How to solve a problem?
  – Search: from here to there, from the initials to the goals
  – Depth-first, breadth-first
• How good is your solution? (fig 6.1, ALFE)
  – How good is your state? How costly is an action?
  – Best-first, Dynamic Programming, A*, etc.
  – Can you guarantee anything? (optimal vs heuristic)
• How much do you want to pay for your solution?
  – How deep/wide can you go?
    • Predetermined vs dynamic (e.g., iterative deepening)
  – One way or many ways (bi-directional)?
• How big is a problem? Can you put the whole world in your head?
  – Tower of Hanoi, chess, robot-and-world,
• HM: state space for TOH, assign values for state and actions.
Home Work

• Tower of Hanio (n=3)
  – 1. Write the state space, states, actions
  – 2. Assume the cost of an action to move a disk is the weight of the disk, compute the “best future cost” $V(s_0)$ for the start state using dynamic programming

Start state $s_0$  

The weight of the disk: disk1=1, disk2=2, disk3=3