Search

Chapter 3

(acknowledgement goes to Gillian Smith(NEU) and Hwee Tou Ng’s for slides used in this lecture)
Search as Problem Solving

- Need to search a space of possible solutions
- There are many sequences of actions, each with their own utility.
- We want to find, or search for, the best one.
Problem formulation

- A set of discrete states (usually finite, usually huge)
- Start state
- Subset of states designated as goals (usually)
- A set of actions the agent can perform

Objective

- Find a sequence of operators that lead to the goal state
Example: Romania

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

**Formulate problem:**
- Goal: at a city; e.g., at Bucharest
- states: at various cities
- actions: choose next city
Example: Romania
Problem Formulation

A problem is defined as follows:

initial state e.g., “at Arad”

actions set of possible actions in current state x.

transition model $Result(x,a) =$ state that follows from applying action a in state x. e.g., $Result(X2 (Arad), A4 (Arad \rightarrow Zerind)) = X3 (Zerind)$

goal test, e.g., $x =$ “at Bucharest”

path cost (additive)
  ➲ e.g., sum of distances, number of actions executed, etc.
  ➲ $c(x,a,y)$ is the step cost, assumed to be $\geq 0$

A solution is a sequence of actions leading from the initial state to a goal state
Start with a specific state

State space forms a tree structure
- Root = start state
- Each node represents a state
- Actions are branches, children are all possible next-states

Search involves expanding a frontier of potential next states
Tree search example
Implementation: general tree search

\[
\text{function Tree-Search( problem, fringe) returns a solution. or failure}
\]

\[
\text{frontier} \leftarrow \text{Insert(Make-Node(Initial-State[problem]), Frontier)}
\]

\[
\text{loop do}
\]

\[
\text{if fringe is empty then return failure}
\]

\[
\text{node} \leftarrow \text{Remove-Front (frontier)}
\]

\[
\text{if Goal-Test[problem](State[node]) then return Solution(node)}
\]

\[
\text{frontier} \leftarrow \text{InsertAll(Expand(node, problem), Frontier)}
\]

\[
\text{function Expand( node, problem) returns a set of nodes}
\]

\[
\text{successors} \leftarrow \text{the empty set}
\]

\[
\text{for each action, result in Successor-Fn[problem](State[node]) do}
\]

\[
\text{s} \leftarrow \text{a new Node}
\]

\[
\text{Parent-Node[s] \leftarrow node; Action[s] \leftarrow action; State[s] \leftarrow result}
\]

\[
\text{Path-Cost[s] \leftarrow Path-Cost[node] + Step-Cost(node, action, s)}
\]

\[
\text{Depth[s] \leftarrow Depth[node] + 1}
\]

\[
\text{add s to successors}
\]

\[
\text{return successors}
\]
Kinds of Search

- Uninformed
- Informed
- Optimization
- Adversarial
Uninformed Search Strategies

- Breadth-First Search
- Depth-First Search
- Limited Depth Search
- Iterative Deepening
Order of node expansion?
- **Shallowest** unexpanded node

Frontier (FIFO Queue)

A
Breadth-First Search

Order of node expansion?
- **Shallowest** unexpanded node
Breadth-First Search

Order of node expansion?

- **Shallowest** unexpanded node

Frontier (FIFO Queue)
C
D
E
Breadth-First Search

Order of node expansion?

Shallowest unexpanded node

Frontier
(FIFO Queue)

D
E
F
G
Example BFS
Search strategy is defined by picking the order of node expansion.

Evaluation criteria:
- Completeness
- Time complexity
- Space Complexity
- Optimality

Time and space complexity measured in:
- $b$ – maximum branching factor for the search tree
- $d$ – depth of least-cost solution
- $m$ – maximum depth of the state space
Breadth-First Search

- Complete?
  - Yes! As long as $b$ is finite

- Optimal?
  - Yes

- Time cost?
  - $O(b^d)$

- Space cost?
  - $O(b^d)$
Breadth-First Search

- **Complete?**
  - Yes! As long as \( b \) is finite

- **Optimal?**
  - Yes! Assuming?

- **Time cost?**
  - \( O(b^d) \)

- **Space cost?**
  - \( O(b^d) \)

Suppose \( b=10 \), a node uses 1000 bytes, 1m nodes/sec can be generated

- Depth 10: \(~3\) hours, \(~10\) terabytes
- Depth 12: \(~13\) days, \(~1\) petabyte
- Depth 14: \(~3.5\) years, \(~99\) petabytes
Depth-First Search

- Order of node expansion?
  - Deepest unexpanded node

Frontier (LIFO Queue):
- A

Diagram:
- A
  - B
    - D
  - C
    - E
    - F
  - G
Depth-First Search

Frontier
(LIFO Queue)

- C
- B
Depth-First Search

Frontier (LIFO Queue)

C
E
D
Depth-First Search

Frontier (LIFO Queue)

C
E
Depth-First Search

Frontier (LIFO Queue)

C
Depth-First Search

Frontier
(LIFO Queue)

G
F
Depth-First Search

Frontier
(LIFO Queue)

G
Properties of depth-first search

- **Complete?** No: fails in infinite-depth spaces
  - Can modify to avoid repeated states along path

- **Time?** $O(b^m)$ with $m =$ maximum depth
  - 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! (we only need to remember a single path + expanded unexplored nodes)

- **Optimal?** No (It may find a non-optimal goal first)
Depth-Limited Search

Depth first search, but don’t go past a pre-set limit (L) while searching.
**Limited Depth Search**

- Combines benefits of depth- and breadth-first

- Number of nodes generated:
  
  \[ (d+1)b^0 + (d)b^1 + (d-1)b^2 + \ldots + (2)b^{d-1} + (1)b^d = O(b^d) \]
Iterative deepening search $L=0$
Iterative deepening search $L=1$
Iterative deepening search $L=2$
Iterative Deepening Search $L=3$
Iterative deepening search

- **Complete?** Yes
- **Time?** $O(b^d)$
- **Space?** $O(bd)$
- **Optimal?** Yes, if step cost = 1 or increasing function of depth.
Iterative Deepening Search

- Combines benefits of depth- and breadth-first
- Number of nodes generated:
  \[(d+1)b^0 + (d)b^1 + (d-1)b^2 + \ldots + (2)b^{d-1} + (1)b^d = O(b^d)\]
- Spatial complexity: \(O(bd)\)
Summary of algorithms

<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>Complete?</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Time</td>
<td>$O(b^{d+1})$</td>
<td>$O(b^{C^*}/\epsilon)$</td>
<td>$O(b^m)$</td>
<td>$O(b^l)$</td>
<td>$O(b^d)$</td>
</tr>
<tr>
<td>Space</td>
<td>$O(b^{d+1})$</td>
<td>$O(b^{C^*}/\epsilon)$</td>
<td>$O(bm)$</td>
<td>$O(bl)$</td>
<td>$O(bd)$</td>
</tr>
<tr>
<td>Optimal?</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Third edition has $O(b^d)$

preferred uninformed search strategy
Classwork
Example: 8-Queens

- **states?** - any arrangement of $n \leq 8$ queens
  - or arrangements of $n \leq 8$ queens in leftmost $n$ columns, 1 per column, such that no queen attacks any other.

- **initial state?** no queens on the board

- **actions?** - add queen to any empty square
  - or add queen to leftmost empty square such that it is not attacked by other queens.

- **goal test?** 8 queens on the board, none attacked.

- **path cost?** 1 per move
Environment types

- **Fully observable** (vs. partially observable): An agent's sensors give it access to the complete state of the environment at each point in time.
- **Deterministic** (vs. stochastic): The next state of the environment is completely determined by the current state and the action executed by the agent. (If the environment is deterministic except for the actions of other agents, then the environment is strategic)
- **Episodic** (vs. sequential): An agent’s action is divided into atomic episodes. Decisions do not depend on previous decisions/actions.
Environment types

- **Static** (vs. **dynamic**): The environment is unchanged while an agent is deliberating. (The environment is **semidynamic** if the environment itself does not change with the passage of time but the agent's performance score does)

- **Discrete** (vs. **continuous**): A limited number of distinct, clearly defined percepts and actions.

- **Single agent** (vs. **multi-agent**): An agent operating by itself in an environment. Does the other agent interfere with my performance measure?
What type of environment/problem space would these algorithms be optimal for?
“What type of agent?”
- An intelligent agent

“Can you formulate a poker agent in terms of a search problem?”
- Sorry I cannot

Show the state space
- $O(bd)$
informed Search
Informed Search

- Use problem-specific knowledge beyond the problem definition to find solutions more efficiently

- **Heuristic**: which node should be expanded next?
Heuristic Search

- Use an evaluation function $f(n)$ for each node
  - Estimate of “desirability”
  - Includes a heuristic function, $h(n)$

- Expand the most desirable unexpanded node
Greedy best-first search

- Evaluation function \( f(n) = h(n) \) (heuristic) = estimate of cost from \( n \) to \( goal \)

- e.g., \( h_{SLD}(n) \) = straight-line distance from \( n \) to Bucharest

- Greedy best-first search expands the node that appears to be closest to goal
Example: Heuristic for Path-Finding
Greedy best-first search example
Greedy best-first search example
Greedy best-first search example
Greedy best-first search example
Properties of greedy best-first search

- **Complete?** No – can get stuck in loops, e.g., Iasi → Neamt → Iasi → Neamt →

- **Time?** $O(b^m)$, but a good heuristic can give dramatic improvement

- **Space?** $O(b^m)$ -- keeps all nodes in memory

- **Optimal?** No
Avoid expanding paths that are already expensive

- Evaluation function: $f(n) = g(n) + h(n)$
  - $g(n) = \text{cost of path so far}$
  - $h(n) = \text{estimated cost from n to goal}$
  - $f(n) = \text{estimated total cost of path through n to goal}$
A* Search Example: Arad to Bucharest

(a) The initial state

<table>
<thead>
<tr>
<th>City</th>
<th>Distance</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>Hirssova</td>
<td>151</td>
</tr>
<tr>
<td>Iasi</td>
<td>226</td>
</tr>
<tr>
<td>Lugoj</td>
<td>244</td>
</tr>
<tr>
<td>Mehodia</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>
A* Search Example: Arad to Bucharest

(b) After expanding Arad

- **Arad**
  - **Sibiu**
    - 393 = 140 + 253
  - **Timisoara**
    - 447 = 118 + 329
  - **Zerind**
    - 449 = 75 + 374

<table>
<thead>
<tr>
<th>City</th>
<th>Distance</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>Dobreta</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>178</td>
</tr>
<tr>
<td>Neamt</td>
<td>77</td>
</tr>
<tr>
<td>Oradea</td>
<td>151</td>
</tr>
<tr>
<td>Pitesti</td>
<td>226</td>
</tr>
<tr>
<td>Rimnicu Vilcea</td>
<td>244</td>
</tr>
<tr>
<td>Sibiu</td>
<td>241</td>
</tr>
<tr>
<td>Timisoara</td>
<td>234</td>
</tr>
<tr>
<td>Urziceni</td>
<td>380</td>
</tr>
<tr>
<td>Vaslui</td>
<td>98</td>
</tr>
<tr>
<td>Zerind</td>
<td>449</td>
</tr>
</tbody>
</table>
A* Search Example: Arad to Bucharest

(c) After expanding Sibiu

- Arad
- Fagaras
- Oradea
- Rimnicu Vilcea

- Sibiu
- Timisoara
  - 447=118+329
- Zerind
  - 449=75+374

Distance table:

<table>
<thead>
<tr>
<th>City</th>
<th>Distance</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>415</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>178</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>241</td>
</tr>
<tr>
<td>Timisoara</td>
<td>253</td>
</tr>
<tr>
<td>Urziceni</td>
<td>329</td>
</tr>
<tr>
<td>Vaslui</td>
<td>199</td>
</tr>
<tr>
<td>Zerind</td>
<td>374</td>
</tr>
</tbody>
</table>
A* Search Example: Arad to Bucharest

(d) After expanding Rimnicu Vilcea
A* Search Example: Arad to Bucharest

(e) After expanding Fagaras

<table>
<thead>
<tr>
<th>City</th>
<th>Distance</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>Dobrogea</td>
<td>242</td>
</tr>
<tr>
<td>Eforie</td>
<td>161</td>
</tr>
<tr>
<td>Fagaras</td>
<td>0</td>
</tr>
<tr>
<td>Giurgiu</td>
<td>0</td>
</tr>
<tr>
<td>Hirsova</td>
<td>0</td>
</tr>
<tr>
<td>Iasi</td>
<td>0</td>
</tr>
<tr>
<td>Lugoj</td>
<td>0</td>
</tr>
<tr>
<td>Mehadia</td>
<td>178</td>
</tr>
<tr>
<td>Neamt</td>
<td>77</td>
</tr>
<tr>
<td>Oradea</td>
<td>151</td>
</tr>
<tr>
<td>Pitesti</td>
<td>98</td>
</tr>
<tr>
<td>Rimnicu V.</td>
<td>193</td>
</tr>
<tr>
<td>Sibiu</td>
<td>241</td>
</tr>
<tr>
<td>Timisoara</td>
<td>234</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>

Arad to Bucharest:
- Arad to Sibiu: 646 = 280 + 366
- Sibiu to Bucharest: 591 = 338 + 253
- Fagaras to Oradea: 671 = 291 + 380
- Oradea to Rimnicu V.: 526 = 366 + 160
- Rimnicu V. to Pitesti: 417 = 317 + 100
- Pitesti to Sibiu: 553 = 300 + 253
- Timisoara to Zerind: 447 = 118 + 329
- Zerind to Arad: 449 = 75 + 374
A* Search Example: Arad to Bucharest
A* Search

- Complete?
  - Yes

- Optimal?
  - Yes!

- Time?
  - Exponential

- Space?
  - Keeps all nodes in memory
Admissible Heuristics

- A heuristic is admissible if, for every node n, \( h(n) \leq h^*(n) \)
  - \( h^*(n) \): true cost to reach the goal from state n

- Example: path-finding heuristic never overestimates actual road distance

- Generate from a relaxed problem
  - Fewer restrictions than original problem
Classwork
A* and Games

Given A* algorithm, define $h(n)$ and $f(n)$, environment, trace the algorithm

Video – Summary and outlook

http://videolectures.net/aaai2010_thayer_bis/