Uninformed Search

Chapter 3

(Based on slides by Stuart Russell, Subbarao Kambhampati, Dan Weld, Oren Etzioni, Henry Kautz, and other UW-AI faculty)
## Agent’s Knowledge Representation

<table>
<thead>
<tr>
<th>Type</th>
<th>State representation</th>
<th>Focus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atomic</td>
<td>States are indivisible; No internal structure</td>
<td>Search on atomic states;</td>
</tr>
<tr>
<td>Propositional (aka Factored)</td>
<td>States are made of state variables that take values (Propositional or Multi-valued or Continuous)</td>
<td>Search+inference in logical (prop logic) and probabilistic (bayes nets) representations</td>
</tr>
<tr>
<td>Relational</td>
<td>States describe the objects in the world and their inter-relations</td>
<td>Search+Inference in predicate logic (or relational prob. Models)</td>
</tr>
<tr>
<td>First-order</td>
<td>+functions over objects</td>
<td>Search+Inference in first order logic (or first order probabilistic models)</td>
</tr>
</tbody>
</table>
Illustration with Vacuum World

Atomic:

S1, S2, S3... S8
state is seen as an indivisible snapshot

All Actions are SXS matrices..

If you add a second roomba
the state space *doubles*

Propositional/Factored:
States made up of 3 state variables
Dirt-in-left-room T/F
Dirt-in-right-room T/F
Roomba-in-room L/R

Each state is an assignment of Values to state variables
2^3 Different states

Actions can just mention the variables they affect

Note that the representation is compact (logarithmic in the size of the state space)

Relational:

World made of objects: Roomba; L-room, R-room, dirt

Relations: In (<robot>, <room>); dirty(<room>)

If you add a second roomba, or more rooms, only the objects increase.

If you want to consider noisiness, you just need to add one other relation
Atomic Agent

**Input:**
- Set of states
- Operators [and costs]
- Start state
- Goal state [test]

**Output:**
- Path: start $\Rightarrow$ a state satisfying goal test
- [May require shortest path]
What is Search?

• Search is a class of techniques for systematically finding or constructing solutions to problems.
  
• Example technique: generate-and-test.
• Example problem: Combination lock.

1. Generate a possible solution.
2. Test the solution.
3. If solution found THEN done ELSE return to step 1.
Why is search interesting?

• Many (all?) AI problems can be formulated as search problems!

• Examples:
  • Path planning
  • Games
  • Natural Language Processing
  • Machine learning
  • ...

6
Example: The 8-puzzle

- states?
- actions?
- goal test?
- path cost?
Example: The 8-puzzle

- **states?** locations of tiles
- **actions?** move blank left, right, up, down
- **goal test?** = goal state (given)
- **path cost?** 1 per move

- [Note: optimal solution of $n$-Puzzle family is NP-hard]
Search Tree Example:
Fragment of 8-Puzzle Problem Space
Example: robotic assembly

- **states?**: real-valued coordinates of robot joint angles parts of the object to be assembled
- **actions?**: continuous motions of robot joints
- **goal test?**: complete assembly
- **path cost?**: time to execute
Example: Romania

- 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
Example: N Queens

• Input:
  – Set of states
  – Operators [and costs]
  – Start state
  – Goal state (test)

• Output
Implementation: states vs. nodes

- A state is a (representation of) a physical configuration
- A node is a data structure constituting part of a search tree includes state, parent node, action, path cost $g(x)$, depth

- The Expand function creates new nodes, filling in the various fields and using the SuccessorFn of the problem to create the corresponding states.
Search strategies

• A search 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
  – space complexity: maximum number of nodes in memory
  – optimality: does it always find a least-cost solution?
  – systematicity: does it visit each state at most once?

• 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 strategies

- **Uninformed** search strategies use only the information available in the problem definition
- Breadth-first search
- Depth-first search
- Depth-limited search
- Iterative deepening search
Repeated states

• Failure to detect repeated states can turn a linear problem into an exponential one!
Depth First Search

- Maintain stack of nodes to visit
- Evaluation
  - Complete? No
  - Time Complexity? $O(b^m)$
  - Space Complexity? $O(bm)$

http://www.youtube.com/watch?v=dtoFAvtVE4U
Breadth First Search: shortest first

- Maintain queue of nodes to visit
- Evaluation
  - Complete? Yes (b is finite)
  - Time Complexity? $O(b^d)$
  - Space Complexity? $O(b^d)$
  - Optimal? Yes, if stepcost=1
Uniform Cost Search: cheapest first

- Maintain queue of nodes to visit
- Evaluation
  - Complete? Yes (b is finite)
  - Time Complexity? $O(b^{(C*/e)})$
  - Space Complexity? $O(b^{(C*/e)})$
  - Optimal? Yes

http://www.youtube.com/watch?v=z6IUnb9ktkE