CSE 3402: Intro to Artificial Intelligence
Search I

- Required Readings: Chapter 3. We won’t cover the material in section 3.6 in much detail.
- Lecture slides adapted from those of Fahiem Bacchus.

Why Search

- Successful
  - Success in game playing programs based on search.
  - Many other AI problems can be successfully solved by search.
- Practical
  - Many problems don’t have a simple algorithmic solution. Casting these problems as search problems is often the easiest way of solving them. Search can also be useful in approximation (e.g., local search in optimization problems).
  - Often specialized algorithms cannot be easily modified to take advantage of extra knowledge. Heuristics in search provide a natural way of utilizing extra knowledge.
- Some critical aspects of intelligent behaviour, e.g., planning, can be naturally cast as search.
Example, a holiday in Jamaica

Things to consider

• Prefer to avoid hurricane season.
• Rules of the road, larger vehicle has right of way (especially trucks).
• Want to climb up to the top of Dunns river falls.
But you want to start your climb at 8:00 am before the crowds arrive!
• Want to swim in the Blue Lagoon

• Want to hike the Cockpit Country

• No roads, need local guide and supplies.
• Easier goal, climb to the top of Blue Mountain

• Near Kingston.
• Organized hikes available.
• Need to arrive on the peak at dawn, before the fog sets in.
• Can get some Blue Mountain coffee!

How do we plan our holiday?

● We must take into account various preferences and constraints to develop a schedule.
● An important technique in developing such a schedule is “hypothetical” reasoning.
  ■ e.g., if I fly into Kingston and drive a car to Port Antonio, I’ll have to drive on the roads at night. How desirable is this?
  ■ If I’m in Port Antonio and leave at 6:30am, I can arrive at Dunns river falls by 8:00am.
How do we plan our holiday?

- This kind of hypothetical reasoning involves asking
  - “what state will I be in after the following sequence of events?”
- From this we can reason about what sequence of events one should try to bring about to achieve a desirable state.
- Search is a computational method for capturing a particular version of this kind of reasoning.

Search

- There are many difficult questions that are not resolved by search. In particular, the whole question of how does an intelligent system formulate its problem as a search problem is not addressed by search.
- Search only shows how to solve the problem once we have it correctly formulated.
The formalism.

To formulate a problem as a search problem we need the following components:

- Formulate a state space over which to search. The state space necessarily involves abstracting the real problem.
- Formulate actions that allow one to move between different states. The actions are abstractions of actions you could actually perform.
- Identify the initial state that best represents your current state and the desired condition one wants to achieve.
- Formulate various heuristics to help guide the search process.

Once the problem has been formulated as a state space search, various algorithms can be utilized to solve the problem.

A solution to the problem will be a sequence of actions/moves that can transform your current state into state where your desired condition holds.
Example 1: Romania Travel.

Currently in Arad, need to get to Bucharest by tomorrow to catch a flight.

Example 1.

- **State space.**
  - States: the various cities you could be located in.
    - Note we are ignoring the low level details of driving, states where you are on the road between cities, etc.
  - Actions: drive between neighboring cities.
  - Initial state: in Arad
  - Desired condition (Goal): be in a state where you are in Bucharest. (How many states satisfy this condition?)

- **Solution** will be the route, the sequence of cities to travel through to get to Bucharest.
Example 2. The 8-Puzzle

- Can slide a tile into the blank spot.
  (Equivalently, can think of it as moving the blank around).

State space.
- States: The different configurations of the tiles. How many different states?
- Actions: Moving the blank up, down, left, right. Can every action be performed in every state?
- Initial state: as shown on previous slide.
- Desired condition (Goal): be in a state where the tiles are all in the positions shown on the previous slide.

Solution will be a sequence of moves of the blank that transform the initial state to a goal state.
Example 2. The 8-Puzzle

- Although there are $9!$ different configurations of the tiles (362,880), in fact the state space is divided into two disjoint parts.
- Only when the blank is in the middle are all four actions possible.
- Our goal condition is satisfied by only a single state. But one could easily have a goal condition like:
  - The 8 is in the upper left hand corner.
  - How many different states satisfy this goal?


- In the previous two examples, a state in the search space corresponded to a unique state of the world (modulo details we have abstracted away).
- However, states need not map directly to world configurations. Instead, a state could map to the agent’s mental conception of how the world is configured: the agent’s knowledge state.

- We have a vacuum cleaner and two rooms.
- Each room may or may not be dirty.
- The vacuum cleaner can move left or right (the action has no effect if there is no room to the right/left).
- The vacuum cleaner can suck; this cleans the room (even if the room was already clean).

Physical states


Knowledge level State Space

- The state space can consist of a set of states. The agent knows that it is in one of these states, but doesn’t know which.

Goal is to have all rooms clean.
### Example 3. Vacuum World.

**Knowledge level State Space**

- Complete knowledge of the world: agent knows exactly which state it is in. State space states consist of single physical states:
  - Start in \{5\}: `<right, suck>`

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Goal is to have all rooms clean.

---

### Example 3. Vacuum World.

**Knowledge level State Space**

- No knowledge of the world. States consist of sets of physical states.
  - Start in \{1,2,3,4,5,6,7,8\}, agent doesn’t have any knowledge of where it is.
  - Nevertheless, the actions `<right, suck, left, suck>` achieves the goal.

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Goal is to have all rooms clean.

Initial state.
\{1,2,3,4,5,6,7,8\}

Right


Suck

Left


Suck
More complex situations.

● The agent might be able to perform some sensing actions. These actions change the agent’s mental state, not the world configuration.

● With sensing can search for a contingent solution: a solution that is contingent on the outcome of the sensing actions
  ■ <right, if dirt then suck>

● Now the issue of interleaving execution and search comes into play.

More complex situations.

● Instead of complete lack of knowledge, the agent might think that some states of the world are more likely than others.

● This leads to probabilistic models of the search space and different algorithms for solving the problem.

● Later we will see some techniques for reasoning and making decisions under uncertainty.
Algorithms for Search.

● Inputs:
  ■ a specified initial state (a specific world state or a set of world states representing the agent’s knowledge, etc.)
  ■ a successor function $S(x) = \{\text{set of states that can be reached from state } x \text{ via a single action}\}$.
  ■ a goal test a function that can be applied to a state and returns true if the state is satisfies the goal condition.
  ■ A step cost function $C(x,a,y)$ which determines the cost of moving from state $x$ to state $y$ using action $a$. ($C(x,a,y) = \infty$ if $a$ does not yield $y$ from $x$)

Algorithms for Search.

● Output:
  ■ a sequence of states leading from the initial state to a state satisfying the goal test.
  ■ The sequence might be
    ● annotated by the name of the action used.
    ● optimal in cost for some algorithms.
Algorithms for Search

● Obtaining the action sequence.
  ■ The set of successors of a state $x$ might arise from different actions, e.g.,
    ● $x \rightarrow a \rightarrow y$
    ● $x \rightarrow b \rightarrow z$
  ■ Successor function $S(x)$ yields a set of states that can be reached from $x$ via a (any) single action.
    ● Rather than just return a set of states, we might annotate these states by the action used to obtain them:
      ● $S(x) = \{<y,a>, <z,b>\}$
        y via action $a$, $z$ via action $b$.
      ● $S(x) = \{<y,a>, <y,b>\}$
        $y$ via action $a$, also $y$ via alternative action $b$.

Tree search

● Assuming search space is a tree, not a graph.
● We use the successor state function to simulate an exploration of the state space.
● Initial call has Frontier = initial state.
  ■ Frontier/fringe is the set of states we haven’t yet explored/expanded.

TreeSearch(Frontier, Successors, Goal?)
  If Frontier is empty return failure
  Curr = select state from Frontier
  If(Goal?(Curr)) return Curr.
  Frontier’ = (Frontier – {Curr}) U Successors(Curr)
  return TreeSearch(Frontier’, Successors, Goal?)
Tree search in Prolog

\[
\text{treeS}([[\text{State}|\text{Path}] | \_], \text{Soln}) :- \\
\text{Goal}?(\text{State}), \text{reverse}([[\text{State}|\text{Path}], \text{Soln})].
\]

\[
\text{treeS}([[\text{State}|\text{Path}] | \text{Frontier}], \text{Soln}) :- \\
\text{GenSuccessors}((\text{State}, \text{Path}, \text{NewPaths}), \\
\text{merge}(\text{NewPaths}, \text{Frontier}, \text{NewFrontier}), \\
\text{treeS}((\text{NewFrontier}, \text{Soln})].
\]