MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.034 Artificial Intelligence, Fall 2003
Recitation 3, September 18/19

Search Me! Prof. Bob Berwick

Agenda

1. **Administrivia: 'Enrichment sites'; Worksheet answers**
2. **Intelligence = "Knowledge Based System" (KBS) = Knowledge + Search**
3. **How to search;**

1. Administrivia

- Good outline & review of search from a more formal point of view, by Korf, at the NEC CiteSeer website: http://citeseer.nj.nec.com/korf98artificial.html

2. What is intelligence? Ans: How to find answers - Good, *optimal* searching

Search is composed of 6 main features:

- DATA STRUCTURE (aka Problem Representation) – mapping from **problem states** into a **graph,** from **graph** into **search tree**
    1. The **start** state.
    2. The **goal** state (or states)
    3. Given an arbitrary state, the **successors** of that state (or a successor function that computes this) (extended=expanded)
    4. A **queue** to keep track of how we are searching through the graph (NB terminology: visited=enqueued)
- CONTROL STRUCTURE
- SEARCH STRATEGY that determines **order** in which we search the queue.

Search arises in many AI contexts, both as finding a goal, and in the more obvious one of finding a path (through a problem space or a real space). Let's look at some examples of the goal-finding sort first. These are perhaps the most 'natural' to visualize (as in the online demo), but perhaps less frequently used in AI. Search techniques are used in AI to find a sequence of steps that will get us from some initial state to some goal state(s). You can either search backwards, from the goal state, or forwards, from the initial state. And whichever direction you choose you can use various search algorithms to do the search. We've so far discussed simple breadth first search and depth first search. These are both systematic, exhaustive search techniques that will eventually try all the nodes in the search space (if its finite!). The appropriate direction of search and the appropriate algorithm depend on the nature of the problem you are trying to solve, and in particular the properties of the search space.

To talk about moving through a space, it is natural to introduce the notions of graphs and trees. A directed graph is like a set of one-way streets – a finite set of vertices (nodes) and links (edges) connecting the nodes. An undirected graph - two-way streets. A cycle (loop) in a graph is a sequence of edges that starts and ends at the same node. A tree is a directed graph without cycles. We can turn graph search problems into tree search problems by (1) replacing each undirected links by 2 directed links (going in opposite directions) and (2) avoiding cycles on any path. Let's try this out on the first simple exercise on the problem sheet (page 4).

The search strategies we will learn can be categorized into 2 groups:

- Uninformed Search: they have no information about the number of steps or the path cost from the current state to the goal – all they can do is distinguish a goal state from a nongoal state. So sometimes this search called **blind search**. There are two basic patterns: **depth-first search (DFS)** and **breadth-first search (BFS).**
- Heuristic Search/informed search: Has some knowledge of what is in the 'right direction' or the 'best' direction/cost.

We will evaluate strategies in terms of four criteria:

· Completeness: is the strategy guaranteed to find a solution when there is one?
· Time complexity: how long does it take to find a solution?
· Space complexity: how much memory does it need to perform the search?
· Optimality: does the strategy find the highest quality solution when there are several different solutions?

2. How to search (from tutor slides)

Search framework pseudocode

1. Initialize *Q* with partial path (*S*) as only entry; set *Enqueued* = *S* (*Note change from slide pseudcode which uses "Visited")*
2. If *Q* is empty, fail. Else, pick some partial path *N* from *Q*
3. If *head*(*N*) = *G*(oal), return *N* (we've reached the goal); *N* is the successful path from *S* to *G*.
4. Else Remove *N* from *Q*
5. Find all the descendants of *head*(*N*) not in *Enqueued* and create all the one-step expansions (extensions) of *N* to each descendant.
6. Add to *Q* all the extended paths; add descendants *of head*(*N*) to *Enqueued*
7. Go to step 2.

Note 1: There are two choices remaining:

- Where to pick elements *N* from *Q* in step 2.
- Where to add the new path extensions to *Q* in step 6.

Note 2: We could stop at step 6 if the extended paths at that point reach the goal, but this won't work for optimal searches, so we use the more general test in Step 3.

Search Strategy	**Which N(ode) of Q(ueue) picked?**	**Where to add expansions of N to Q?**
Depth-first (w/ backup)	First element of Queue	Front of Q
Breadth-first	First element of Queue	End of Q
Hill-climbing	First element of Queue	Front of Q, sorted by heuristic value
Hill-climbing (no backup)	First element of Queue	Replace Q with sorted expansions of N
Beam (width *k*, no backup)	Best *k* elt of Q by heuristic value	Anywhere in Q

Implementing Depth-First Search

Our control choices: (1) Pick *N* from the ***first*** element of the *Q*; (2) Add new path extensions to the ***front*** of *Q*.

Implementing Breadth-first search

Our control choices: (1) Pick *N* from the ***first*** element of the *Q*; (2) Add new path extensions to the *end* of *Q*.

Implementing Heuristic Search

Add: measure of "goodness" of node (a number)

Control choices: hill-climbing: pick first node in *Q* & sort expansions of this node by 'goodness'

best-first: pick best of all nodes in *Q*

don't use enqueued (visited) list

4. Representing problems as search problems – beyond maps

The key lesson for this recitation: searching is not just about maps! It applies whenever we can abstract a problem as a choice amongst alternatives, a set of states of the problem (the problem or state space), a special start state, a goal state, and a way to get from one state to another (the next/valid or legal moves) Let's do a few examples so you can see how this conversion works.

- How do we represent the States? (one state) – tells us how to represent the state space
- How do we represent the start state? How do we represent the goal state?
- How do we represent legal moves (transitions) from state to state? (Move operators)
- What order do we try paths in?
- How do we represent progress (distance) from the goal? (Path cost)

To get a feel for this, as our second problem let's try encoding the following (familiar?) puzzle as a search problem, the **8-puzzle:**

Breadth-first Search for 8-puzzle game

In the breadth- first search, the root node is expanded first, then all the nodes generated by the root node are expanded next, and then their successors, and so on. In general, the nodes at depth d in the search tree are expanded before the nodes at depth d+1.

Order: At each depth, there is no preference among the nodes to be expanded next.

Move operators: What about using operators such as "move the 4 tile into the blank space." What would be a better choice & why?

Path Cost: each step costs 1, so the path cost is just the length of the path.

Depth-first Search for 8-puzzle game

Depth-first search always expands one of the nodes at the deepest level. Only when the search hits a dead end (a non-goal node with no expansion), then the search goes back and expands nodes at shallower levels. In this case, there will always be a blank space that one of the adjacent tiles can be moved into, so there is no dead end for 8-puzzle game. The drawback of this search is that it can get stuck going down the wrong path. Many problems have very deep or even infinite search trees, so this search will never be able to recover from an unlucky choice at one of the nodes near the top of the tree. The search will always continue downward without backing up, even when a shallow solution exists. Thus, on these problems, this search will:

- Either get stuck in an infinite loop and never return a solution.
- Or it may eventually find a solution path that is longer than the optimal solution.

That means depth-first search is neither complete nor optimal.

Hill-climbing ("greedy search")

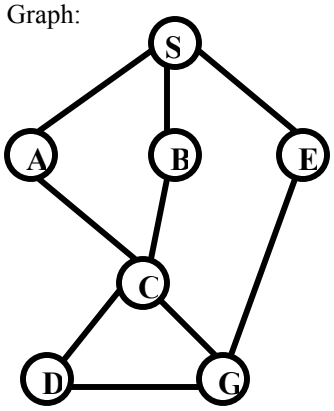
What would be a good hill-climbing heuristic for this problem (path cost or estimated distance to goal)? How can we measure distance to the goal? Are heuristic searches like this either complete or optimal?

See http://www.permadi.com/java/puzzle8/

Practice Problems

Problem 1: Turning graphs into search trees.

Graph:



Tree:

Problem 2: Simulating DFS and BFS. In the graph above, assuming **no enqueued list**, enumerate the order in which all nodes are expanded starting from *S* for each of the search strategies listed below. In other words, assume that the goal is a node not found in the graph. Also assume that nodes are expanded by delineating descendents left to right, and that the search strategies include backup. (How does it change if we do have an enqueued list? See the next problem.)

Applet: http://www.cs.ubc.ca/labs/lci/CIspace/Version4/search/index.html

Depth-first: **S,…**

Breadth-first: **S,…**

Problem 3: Now using an enqueued ("visited") list and simulating heuristic search. Now search the above graph using an enqueued list and each of the search strategies listed below. Assume that we're looking for a path from S to G, and that our search strategy checks for having reached the goal when a partial path is taken off the queue (and about to be expanded). Assume all search strategies except breadth-first include backup. When heuristics are needed, assume these for each node: S 1, A 5, B 3, E 7, C 6, D 2, G 0, where a lower number is 'better'. For each strategy below, enumerate the order in which nodes are expanded and give the path found from S to G.

Strategy	Node expansion order	Path S → G
Depth-first	**S**	**S**
Breadth-first	**S**	**S**
Hill-Climbing	**S**	**S**
Best-first	**S**	**S**

Here is a work sheet form to help.

Depth first search	Queue	Enqueued list	Next node to expand	Not added
Step 1				
Step 2				
Step 3				
Step 4				
Step 5				
Step 6				

Breadth first search	Queue	Enqueued list	Next node to expand	Not added
Step 1				
Step 2				
Step 3				
Step 4				
Step 5				
Step 6				
Step 7				

Hill climbing	Queue	Enqueued list	Next node to expand	Not added
Step 1				
Step 2				
Step 3				
Step 4				
Step 5				

Best- first search	Queue	Enqueued list	Next node to expand	Not added
Step 1				
Step 2				
Step 3				
Step 4				
Step 5				
Step 6				

Problem 4. The 8-puzzle (see above & web).