Problem-Solving by Search (part 3)

CSCI 5582, Fall 2007
Administrivia

• Reading for this week: R+N Chapter 4, sections 1-3
• Problem Set 1 posted; due Sept. 25
• Hard copy ONLY (except for CAETE students)
Depth-First-Search [Nodes]
    IF there are no more nodes
        THEN FAIL
    ELSE ThisNode ← First (Nodes)
        IF ThisNode is the goal
            THEN Return ThisNode
        ELSE
            ChildNodes ← Children (ThisNode)
            Depth-First-Search
            (Append-to-Front
             ChildNodes
             Rest (Nodes))
What could we do with more information?

- Depth- and breadth-first search just add new states at the beginning and end (respectively) of the “still-untried” list.

- Suppose we could look at a bunch of problem states and judge which looked “most promising” (in some sense to be defined). Then we could order our list of untried states in “most-to-least promising order”.
Best-First-Search [Nodes]
   IF there are no more nodes THEN FAIL
   ThisNode <-- First(Nodes)
   IF ThisNode is the goal
      THEN return ThisNode
   ELSE
      ChildNodes <-- Children(ThisNode)
      Best-First-Search
         (Sort-by-Quality
            ChildNodes
            Remove-Node(ThisNode Nodes))
How do we measure the "quality" of a state in our search?

- Least cost so far (uniform cost search)
- Apparently closest to goal (greedy search)
- Combine least cost and (apparent) nearness to goal (A* search)
The Basic Idea Behind A* Search

1) Devise an "estimation" function, $h$, that gives you a plausible underestimate of the distance from a given state to the goal.
2) Call the cost of getting from the initial state to the current state $g$.
3) To measure the quality of a given state $s$, sum the cost ($g$) of arriving at this state and the estimate ($h$) of the cost required to get from $s$ to the goal.
4) Use best-first search, where $g+h$ is the overall quality function.
How do you estimate distance to the goal? (Or: What’s that “h” function?)

Try to find a metric that will be a close (informative) one, but that won’t overestimate (it should be optimistic).

*Example:* 8-9 puzzle
- Number of misplaced tiles
- Sum of Manhattan distance of misplaced tiles (better)
- Best solution length for (say) tiles 1-4

*Example:* Choosing a path on a map from city A to B
- Euclidean distance to B.
AStar-Best-First-Search [Nodes]
   IF there are no more nodes THEN FAIL
   ThisNode <-- First(Nodes)
   IF ThisNode is the goal
      THEN return ThisNode
   ELSE
      ChildNodes <-- Children(ThisNode)
      Astar-Best-First-Search
         (Sort-by-Lowest-Cost-plus-Estimate
          ChildNodes
          Remove-Node(ThisNode Nodes))
How hard is a (search) problem?

- How big is the problem space?
- How long is the best solution path? (the “optimality” issue)
- How long does it take to find the best solution (the time-of-search issue)
- How complex is the strategy needed to find the best solution? (Does this problem reward or require expertise?)
- What, if any, kinds of special knowledge or heuristics do we have?
The “optimal” path is awful: Towers of Hanoi
The optimal path is (relatively) short, but finding the solution can take eons: the traveling salesman problem
A few handy heuristics

- Redefine the “edges” (moves) of the problem graph, as by chunking (e.g., Rubik’s cube)
- Eliminate chunks of the problem graph, as by constraints (e.g., SEND+MORE = MONEY problem)
- Decompose the problem into subproblems (Rubik’s cube again; or Rush Hour)
Means-Ends Analysis

A technique for finding the next step to take in a search process. Here, we find the most important difference between our current state and the goal state and use that difference (recursively) to dictate our next move.
To SOLVE a problem: If the current state is the goal state, then we’re done. If not, first REDUCE the difference between the current state and goal state to produce a new current state. Then SOLVE the problem from the (presumably better) current state and the original goal.

To REDUCE the difference between the current state and the goal state, find the most important difference between the current state and the goal state. Find an operator that tends to reduce that difference, and APPLY that operator to the current state.

To APPLY an operator to the current state, see if there are any differences between the current state and any preconditions for the operator. If there are, REDUCE the difference between the current state and the preconditions for the operator. If not return the result of APPLYing the operator directly to the current state.
Local search

• Up until now, we’ve imagined a two-phase method: find a solution path (search the problem space graph), and then return that path so that it can be executed. In local search, we simply search the graph directly, hoping to land on a goal state.

• Two simple methods: hill-climbing and beam search.
Hill-Climb [Current-Node]
  IF Current-Node is the goal
  THEN return Current-Node
  ELSE
    ChildNodes ← Children (Current-Node)
    BestChild ← LooksBest (ChildNodes)
    Hill-Climb [BestChild]

Beam-Search [Set-of-Nodes]
  If Set-of-Nodes contains a goal state
  THEN return that state
  ELSE
    AllChildNodes ← Find-All-Children (Set-of-Nodes)
    BestNChildren ← N-Best-Looking (AllChildNodes)
    Beam-Search (BestNChildren)