Midterm Exam Review

Note Title

Production Systems & Problem Solving

Formalizing a "problem"
and a solution approach

Problem: **State Representation** — Initial State

Knowledge Base

Rules ("Moves")

- precondition: is Rule r applicable to state s ?
- action: what is the result when Rule r is applied to state s ?

Goal condition goal(s): Does state s satisfy goal condition?

Note: for a given state, many rules may be applicable

A solution strategy must decide which rule to apply when many are applicable.

Solution Strategies: Irrevocable, Tentative

Irrevocable: Keep track of current state only; applying rule r to state s produces new value of state s. When algorithm completes, state s is the solution state.

Tentative: Remember what has been done up til now; rules that have been applied & didn't work may be revoked — try something new, etc

Implicit Graph: Begin with Start Node, build graph by choosing a node and generating successors with a Successor Function (e.g., "ApplyMove(rule, s)")

a. Flailing Wildly

b. Hill Climbing
- variations
- simulated annealing

irrevocable

use evaluation function f(state)

by never choosing a state with a lower evaluation than its predecessor, we ensure we will not return to any previous state.

However we may get "stuck" at a "local optimum" and not reach a global optimum of f(s).

allow algorithms to choose "best neighbor" even if that value is lower than current state

..... potential problems

- try from many different starting positions
- use different "tie-breaking rules", etc

allows occasional steps backward but requires overall forward movement.

C. Tentative Strategies

Backtracking → maintain list of states visited on current path only
[forgetting things that failed]
Problem: repeating past mistakes.

use a depth bound (n)
advantage – if a solution can be found in n moves or less, backtrack will find it.
if not, will fail

iterative deepening: try solving w/ depth n
if no solution, increase n

heuristics, informed search
– instead of trying moves in arbitrary or random order, try them according to "most promising" first, etc.

GraphSearch – Keeps track of a subtree of the solution space

each node has exactly one parent
(except root = initial state, with no parent)

as new nodes generated, either

- never seen before: use node that generated it as its parent
- seen before: choose between current parent + node that generated it.
  [do this so that path from root to new node is min. cost or shortest

Note: this method can produce a
breadth-first search
or depth-first search
or best-first search

depending entirely on the order in which un-explored nodes are examined.

"Algorithm A" [which is really a class of evaluation functions] says:

> nodes on the OPEN list should be stored in order from least to highest value of a function f(n) having the form
>
> $$f(n) = depth(n) + \underline{h(n)}$$
>
> → estimate of distance from n to a goal node
>
> thus, f(n) estimates the total distance from start to a goal node if going through node n.

Note: If $h(n)$ is chosen carefully enough to ensure that it never overestimates the distance from n to goal (i.e., $h(n) \leq h^*(n)$) it is called an admissible heuristic and use of it with Algorithm A is known as Algorithm A* and guarantees a shortest path solution to a goal.

$h^*(n)$ = actual distance (usually unknown)

Note: if $h_1(n)$ and $h_2(n)$ are both admissible (i.e., $h_1(n) \leq h^*(n)$ $h_2(n) \leq h^*(n)$ for all nodes n)

then $h_2(n)$ is more informed than $h_1(n)$ if $h_1(n) \leq h_2(n)$ for all n

and provides better performance than $h_1(n)$

(i.e., fewer (or at least no more) node expansions to find solution

Note $h(n) = 0$ in all cases is admissible

satisfies these conditions

$\Rightarrow$ so $f(n) = \text{depth}(n) + 0$

provides breadth-first search

(because it ensures all nodes at level K will be examined before any nodes at level K+1, ...

Other stuff:

Agents: Performance Measures
Environment
Actuators
Sensors

deterministic,
stochastic
fully or partially observed...