Lecture 7 - Search Algos, Trees

Wednesday, January 30, 2019 9:01 AM

last class - static states & static transitions bt them

search tree created dynamism

search
- informed (shortcut)
- uninformed (brute force)

metrics:

① $d(x,y) \geq 0$

② $(d(x,y)=0) \iff (x=y)$

③ $d(y,x) = d(x,y)$

④ $d(x,z) \leq d(x,y) + d(y,z)$

search algorithm:
ordered investigation of a search tree for a goal state

- correctness if search algo returns an answer, it's a goal state
- completeness search algorithm addresses each possible input

one of these can hold but not the other (looks @ all inputs complete but doesn't return goal states correctness)

PC^*

$S \models_{PC^*} P \implies S \vdash_{PC^*} P$

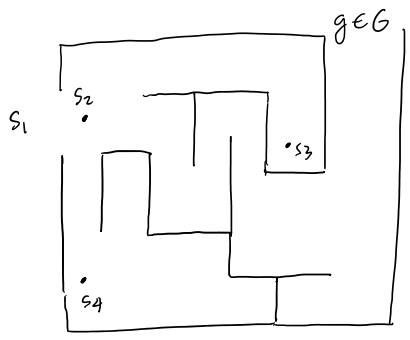
depth first search (DFS) initial state s_1
recursive guess & check scheme
(use generator function to choose movement based on possible options)

let n be an arbitrary search node.

child(n) refers to a set of nodes $m \in M$ such that $n \to m$ is in our search tree.

① check if $f_T(n)$ is true, if so halt & return $f_T(n)$

② otherwise, for each $m \in$ child(n), return the DFS(m).



order: ↓ ← ↑ →

depending on the order, you may get caught in an infinite loop

goal is to use an algorithm that finds goal state before descending down depth (which may or may not be finite)

① DFS may require a specific order of checks

② DFS is not guaranteed to halt

③ if it does halt, 2 diff claims abt complexity:
- ⓐ performance is linear in the size of the graph
- ⓑ worst-case storage is size of graph itself (problematic as search tree could be infinite)

how do we make DFS halt more appropriately?
- make a visited list (if you've seen a node, don't see it again)

prove DFS with a list is complete over mazes (mazes are finite) ← complete dis

Breadth-first search (BFS):

Let Q be a queue data structure (FIFO)

push → Q → pop
$q_n \cdots q_1$

start w/ $Q = \langle s_1 \rangle$ initial state

① pop Q, call it n

② test $f_T(n)$, if true halt & return

③ otherwise, for each $m \in$ Child(n), push m to Q

④ repeat

properties
- BFS is complete → assume a finite Ω and consider how Ω is enumerated by the search tree
- BFS produces shortest solution in terms of search nodes (# steps in produced solution)

structure of recursion created a kind of "stack" structure (FILO)

Heuristic search

function h is a heuristic iff ① $h(n) \geq 0$
② $(h(n)=0)$ iff $(n \in G)$
③ $(h(n)=\infty)$ iff (n is a dead end & can't lead to a soln)

h is admissible iff it never overestimates

3 kinds of heuristic search

local optimality
- best-first search → operates over weighted graphs (cost functions bt transitions) and expands lowest cost path. can apply to unweighted graphs by assigning all paths to weight 1
- hill-climbing → like best-first but only considers next step & expands lowest not always returning actual solution
- hill climbing WITH MOMENTUM → forces a certain # of steps even after obtaining what could be a local minimum/maximum
  training neural networks
- A* (subset of best first search) → let h(n) be the straight line distance to the end of a maze
  claim: h is admissible
  consider the following functions, r(x) where x is a graph node
  $r(x) = g(x) + h(x)$
  g(x) is the observed cost of the path taken to x

A* extends the path where r(x) is minimized
$\boxed{\arg\min_n r(n)}$

A* halts when $r(n)$ is $\leq g(m) + h(m)$ for all considered m, n
h is monotone (non-increasing) → non worsening
if h is monotone, A* search never processes a node more than once
h gets smaller as you get closer