Online Search
David Cline
Localization

• An agent attempts to discover its location by sensing the environment and moving around in it.

• Over time the agent's percepts and movements may narrow the belief state until only one state remains.
Which square look like the second to left square on the bottom?
Searching with Sensing

- Online search agent interleaves sensing (percepts) with actions.
  - Actions may be non-deterministic
  - Agent may not know what actions do
  - Agent can only sense current world state.
  - The goal may be exploration
Online Search

• Agent Knowledge

  - Actions(s)
    • applicable actions in state s
  - Cost(s, a, s')
    • cost of having performed an action. (a robot may not know that an action will damage it)
  - GoalTest(s)
Problems with online search

- Dead ends
  - no algorithm can avoid all dead ends
- Irreversible actions
  - safely explorable environments
- Performance
  - competitive ratio
- Adversary argument
  - imagine environment as an adversary
- Unknown results of actions
  - random walk
Possible solutions

• Give agent memory and heuristic function
  – If an agent remembers states, it may be able to explore more effectively.
  – If an agent has a heuristic function $h(s)$ that it can direct search towards the goal.
Online DFS agent

• Build a map as it explores the state-action space
• Agent can only expand a node it currently occupies
  - Uses physical backtracking
  - Only works if actions can be "undone"
• Use depth-first strategy
  - Helps avoid traversing back and forth continually (local search strategy)
Online DFS agent

function ONLINE-DFS-AGENT(s′) returns an action
inputs: s′, a percept that identifies the current state
persistent: result, a table indexed by state and action, initially empty
            untried, a table that lists, for each state, the actions not yet tried
            unbacktracked, a table that lists, for each state, the backtracks not yet tried
            s, a, the previous state and action, initially null

if GOAL-TEST(s′) then return stop
if s′ is a new state (not in untried) then untried[s′] ← ACTIONS(s′)
if s is not null then
    result[s, a] ← s′
    add s to the front of unbacktracked[s′]
if untried[s′] is empty then
    if unbacktracked[s′] is empty then return stop
    else a ← an action b such that result[s′, b] = POP(unbacktracked[s′])
else a ← POP(untried[s′])
s ← s′
return a

Figure 4.21 An online search agent that uses depth-first exploration. The agent is applicable only in state spaces in which every action can be “undone” by some other action.

• (Q) What does this agent store?
Example

- (Q) Run the online DFS algorithm on the following

  - The agent has 4 actions
    - u, d, r, l
LRTA* (learning real-time A*)

- Builds a state-action map, just like online DFS
- Store a "best" heuristic cost $h(s)$ for each explored state.
- Update $h(s)$ as algorithm progresses
- Assume going to the goal through unexplored neighbors of $s$ to goal has total cost $h(s)$
  - Optimism under uncertainty principle

(Q) What does optimism under uncertainty imply about how the algorithm will proceed?
**LRTA***

**function** LRTA*-AGENT(s') **returns** an action

**inputs:** s', a percept that identifies the current state

**persistent:** result, a table, indexed by state and action, initially empty
H, a table of cost estimates indexed by state, initially empty
s, a, the previous state and action, initially null

if GOAL-TEST(s') then return stop
if s' is a new state (not in H) then H[s'] ← h(s')
if s is not null
result[s, a] ← s'
H[s] ← \( \min_{b \in \text{ACTIONS}(s)} \text{LRTA}^*-\text{COST}(s, b, \text{result}[s, b], H) \)
a ← an action b in \text{ACTIONS}(s') that minimizes \text{LRTA}^*-\text{COST}(s', b, \text{result}[s', b], H)
s ← s'
return a

**function** LRTA*-COST(s, a, s', H) **returns** a cost estimate
if s' is undefined then return h(s)
else return \( c(s, a, s') + H[s'] \)

**Figure 4.24** LRTA*-AGENT selects an action according to the values of neighboring states, which are updated as the agent moves about the state space.