Online Search Agents and Unknown Environments

CSCI 4511
Thursday, February 12th, 2015
So far..

• Agents using offline search algorithms:
  – Compute a **full solution** before acting
  – **Only** after solution is found, take action
So far..

• Agents using LOCAL search algorithms:
  – Still have access to the full state space
  – But only store current state and successors
Now..

• Online search:
  – Agent doesn't know entire state space
    • Just knows his current state and possible actions.
  – Interleaves computation and action:
    • Takes an action
    • Observes the environment
    • Compute the next action
    • Repeat
• Online search is good idea when:
  – Dynamic or semi dynamic environments
  – Non-deterministic domains
• Online search is NECESSARY when:
  – Unknown environments:
    • What are the states?
    • What do my actions do?

• Agent faces an exploration problem
Examples

• Robot motion
  – Exploration, finding exit

• Newborn
Online search problems

- Deterministic and Observable env.

- Agents know:
  - Actions(state)
  - Step-cost function $c(s, a, s')$
  - Goal-test

- Agents do not know:
  - Effect of their actions
  - State space
Figure 4.18 A simple maze problem. The agent starts at $S$ and must reach $G$, but knows nothing of the environment.
Objective

• Typically:
  • Reach the goal while minimizing cost
  • Can also be to explore environment

• Performance: state space explored vs entire state space
Problems

• Dead ends:
  – If actions not reversible and all actions tried

• For now, assume safely explorable state space
  – Example: 8-puzzle problem, mazes
Online search agents

• Compare offline vs online
  – Think about A* or UCS

  – Can we expand a node in **any** location?

  – Physical location

  – Can we adapt a known search algorithm to online setting?
Online search agents

- Compare offline vs online
  - Think about A* or UCS
  - Can we expand a node in *any* location?
  - Physical location
- Online DFS: only with reversible actions
function ONLINE-DFS-AGENT(s') returns an action

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

static: result, a table, indexed by action and state, initially empty
unexplored, a table that lists, for each visited state, the actions not yet tried
unbacktraacked, a table that lists, for each visited 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 then unexplored[s'] ← ACTIONS(s')
if s is not null then do
    result[a, s] ← s'
    add s to the front of unbacktraacked[s']
if unexplored[s'] is empty then
    if unbacktraacked[s'] is empty then return stop
    else a ← an action b such that result[b, s'] = POP(unbacktraacked[s'])
else a ← POP(unexplored[s'])
s ← s'
return a

Figure 4.20 An online search agent that uses depth-first exploration. The agent is applicable only in bidirected search spaces.
Online local search

• Remember algorithms discussed last class:
  – Applicable to online search?
Online local search

• Remember algorithms discussed last class:
  – Applicable to online search?
  – Hill-climbing search
Online local search

- Remember algorithms discussed last class:
  - Applicable to online search?
  - Hill-climbing search: Local maxima?
Online local search

• Remember algorithms discussed last class:
  – Applicable to online search?
  – Hill-climbing search: Local maxima?
  – Random Walk!
Figure 4.21 An environment in which a random walk will take exponentially many steps to find the goal.
Online local search

• Remember algorithms discussed last class:
  – Applicable to online search?
  – Hill-climbing search: Local maxima?
  – Random Walk! $\rightarrow$ very inefficient
  – Agents need memory
LRTA*

• Learning Real Time A*
  – Step-cost fcn $c(s, a, s')$
  – 'current best estimate' $H(s)$

• Builds a 'map' based on actions

• Actions not tried assumed to lead to goal
Figure 4.22  Five iterations of LRTA* on a one-dimensional state space. Each state is labeled with $H(s)$, the current cost estimate to reach a goal, and each arc is labeled with its step cost. The shaded state marks the location of the agent, and the updated values at each iteration are circled.
function LRTA*-AGENT($s'$) returns an action

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

static: result, a table, indexed by action and state, 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 $H[s'] ← h(s')$
unless $s$ is null

result[$a, s] ← s'$

$H[s] ← \min_{b \in \text{ACTIONS}(s)} \text{LRTA*-COST}(s, b, \text{result}[b, s], H)$

$a ←$ an action $b$ in \text{ACTIONS}(s') that minimizes \text{LRTA*-COST}(s', b, \text{result}[b, s'], 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 4.23  LRTA*-AGENT selects an action according to the values of neighboring states, which are updated as the agent moves about the state space.
LRTA*

- Guaranteed to find goal in environment:
  - Finite
  - Safely explorable
Learning in online search

• Learn a 'map' of the environment
  – (s,a,?)

• Acquire more accurate estimates of cost of a state