

Online search algorithms

CS4881 Artificial Intelligence

Jay Urbain

Outline

- Online search
- Exploratory search
- Sample applications
- Online search problems
- Online-DFS-Agent
- Online local search
- Random walk
- LTRA*

Online search

- We have concentrated on agents that use **offline search** algorithms.
 - They compute a complete solution before setting foot in the real world.
 - Then execute the solution without recourse of their percepts.
 - Talk about analysis paralysis!
- In contrast, an **online search** agent operates by interleaving computation and action.
 - First it takes an action
 - Then it observes the environment and computes the next action.

In what types of environments is online search required?

Online search

- *Online search* is a good idea in dynamic or semidynamic domains –
 - Domains where there is a penalty for sitting around and computing too long.
- *Online search* is an even better idea for stochastic domains – *think taxi cab agent!*
- In general, *offline search* would need to come up with an *exponentially* large contingency plan that considers all possible happenings.
- *Online search* need only consider what actually *does* happen.
Example:
 - A chess playing agent is well advised to make its first move long before it has figured out the complete course of the game. *Why?*

Exploratory Search

- Online search is necessary for any *exploration* problem.
- States and actions are *unknown* to the agent.
- An agent in this state of *ignorance* must use its actions as experiments to determine what to do next.
- Requires agent to *interleave computation and action*.
- Discovery of how the world works, is in part, an *online search process*.

Sample Problems

Examples:

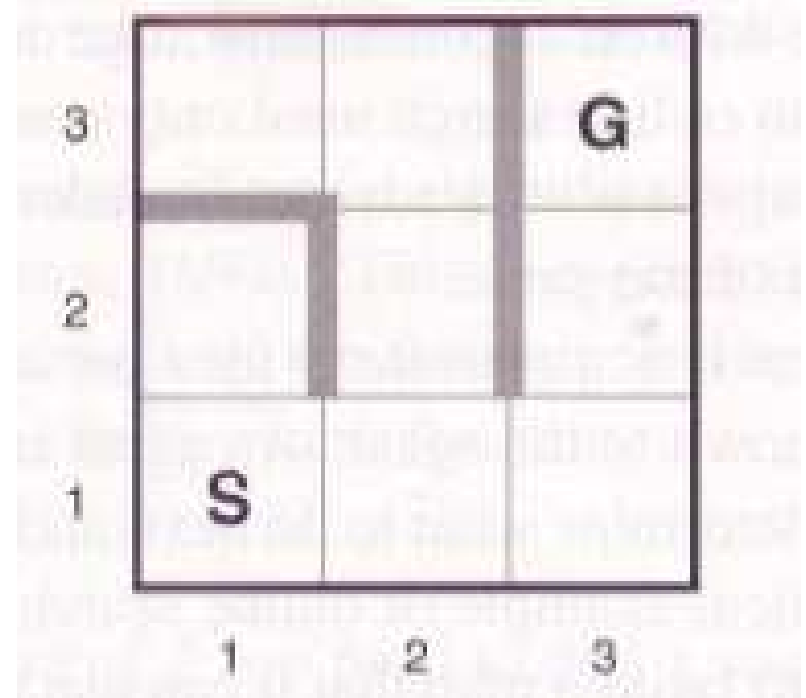
- Robot that is placed in a new building and must explore it to build a map that it can use for getting from A to B.
- Methods for escaping from labyrinths!
- Web search agent builds knowledge and structure of the problem as it explores environment to achieve desired goal, receives feedback from user (relevance feedback).
- Newborn exploring their environment:
 - Has many actions, does not know outcomes of actions, and has experience in only a few of the possible states it can reach.
- Model for human language acquisition.

Online Search Problems

- Online search problem can be solved *only* by an agent executing actions, rather than by a purely computational process.
- Assume agent knows the following:
 - *ACTIONS(s)*, which returns a list of actions allowed in state s .
 - The *step-cost* function $c(s, a, s')$.
 - Note: can not be used until agent knows s' .
 - *GOAL-TEST(s)*.

Online Search Problems

- Agent cannot access the successors of a state except by actually trying all the actions in the state.
- In *maze* problem, agent does not know that going *Up* from (1,1) leads to (1,2).
- *Ignorance can be reduced* – robot explorer might know how its movement actions work, but may be ignorant of the location of obstacles.



Online Search Problems

Assumptions:

- Agent can recognize state it has visited before.
- Actions are *deterministic*.
- Agent may have access to admissible heuristic *$h(s)$* .
- Agent's objective is to *reach goal state while minimizing cost*.
- Cost is the *total path cost* agent travels.

Online Search Problems

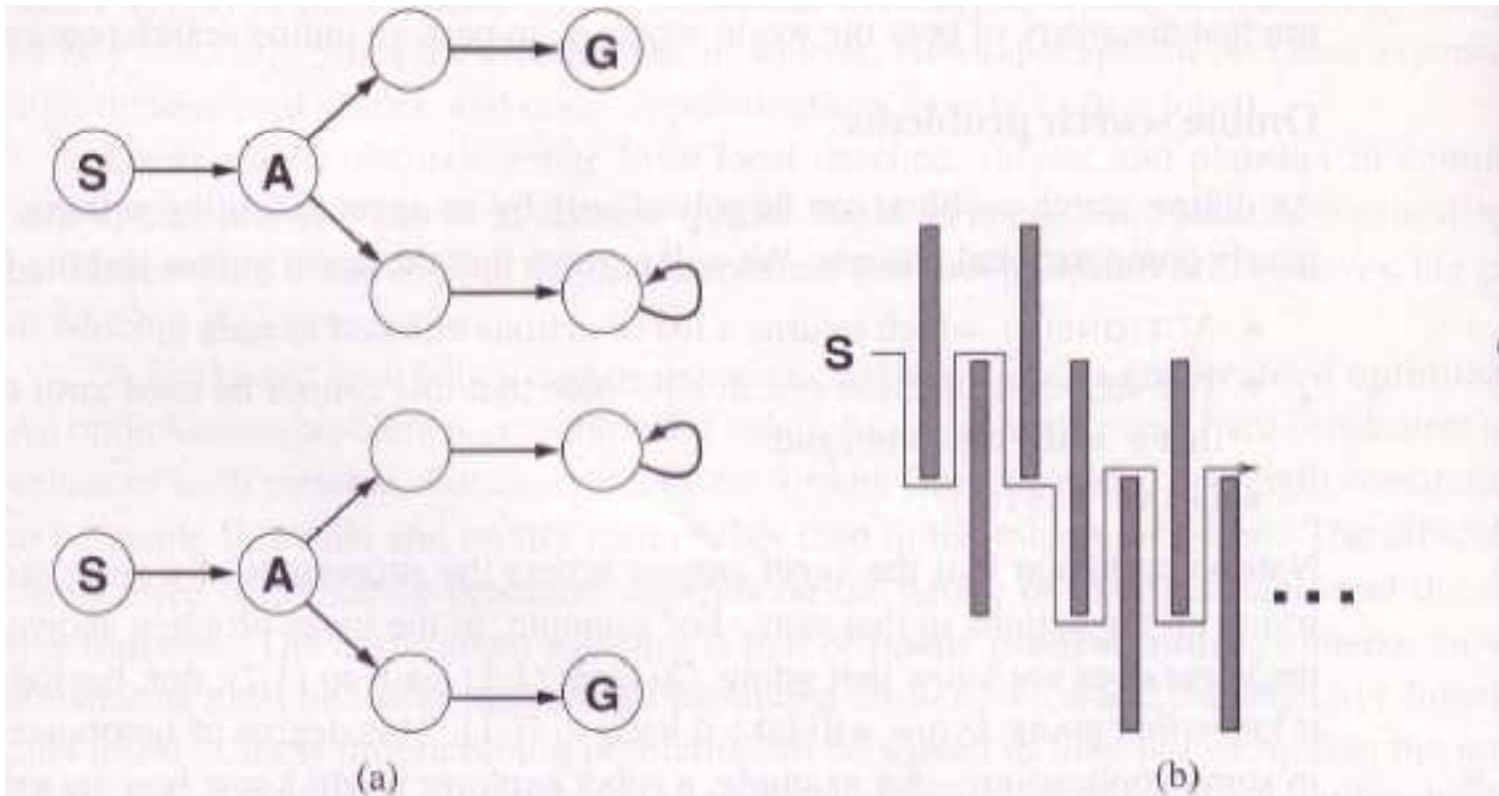
- Common to compare online path cost with the cost of the path the agent would take if it knew the search space in advance.
 - Just plug in an informed search algorithm.
 - Known as “*competitive ratio*”
- *In a worst-case scenario what is the best competitive ratio you can achieve?*
- *In what circumstances would this occur?*

Online Search Problems

- Worst case competitive ratio can be infinite.
 - Some actions can be *irreversible* – can reach dead-end state from which goal state is unreachable.
- Claim:
 - *No algorithm can avoid dead ends in all state spaces.*
- Adversary argument:
 - Imagine an adversary that constructs the state space while the agent explores it and can put the goals and dead ends wherever it likes.
 - *Major research problem in natural terrain robotics – staircases, cliffs, ramps.*
- ***For now we assume state space is safely explorable and reversible.***

Online Search Problems

- a) State space that can lead to dead end**
- b) Adversary blocks route to goal with wall in 2D environment.**



Online search agent

- After each *action*, agent receives *percept* (feedback) defining *state* reached.
- Agent augments it's *map of the environment* with this information.
- Current map is used to decide where to go next.
- Online agent can *only expand a node it occupies* –
 - Can't just jump around a search tree like A* informed search.
 - Better to expand nodes in local order – depth-first search.

Online Search Problems

```
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
           unbacktracked, 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[ $s'$ ]  $\leftarrow$  ACTIONS( $s'$ )
  if  $s$  is not null then do
    result[ $a, s$ ]  $\leftarrow s'$ 
    add  $s$  to the front of unbacktracked[ $s'$ ]
  if unexplored[ $s'$ ] is empty then
    if unbacktracked[ $s'$ ] is empty then return stop
    else  $a \leftarrow$  an action  $b$  such that result[ $b, s'$ ] = POP(unbacktracked[ $s'$ ])
  else  $a \leftarrow$  POP(unexplored[ $s'$ ])
   $s \leftarrow s'$ 
  return  $a$ 
```

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

```
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 the 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
```

Online-DFS-Agent

- Trace through maze problem with Online-DFS-Agent.
 - *How many links* in state-space will *Online-DFS-Agent* visit for worst case scenario in reaching it's goal?
 - Will *Online-DFS-Agent* work in environments where actions are irreversible?
 - Do any such agents have a *bounded* competitive ratio?

Online-Local Search

- Like *depth-first search*, *local hill-climbing search* has the property of *locality* in its node expansions.
 - Unfortunately, it can easily leave agent at *local maxima*.
 - Random restart does not work, since *agent can't teleport itself to remote node in search space*.

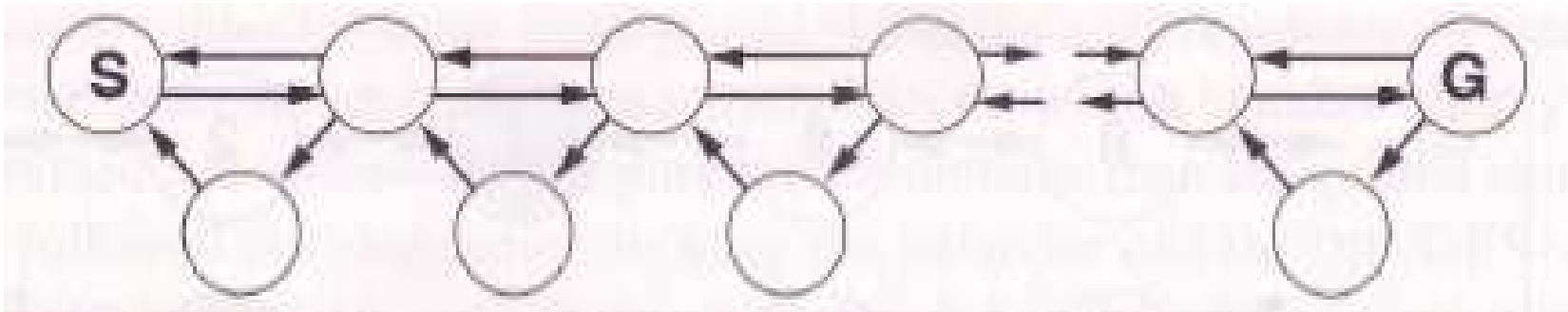
Random Walk

- Random walk *randomly selects one of the available actions* from the current state.
- Preferences can be given to actions that have not yet been tried.
- Random walk will *eventually* find a goal or complete its exploration if state-space is *finite**
- Can be very slow, can be surprisingly fast.

**Infinite case is tricky: Complete in 1D & 2D, chance of 0.3405 in 3D – see Hughes 1995.*

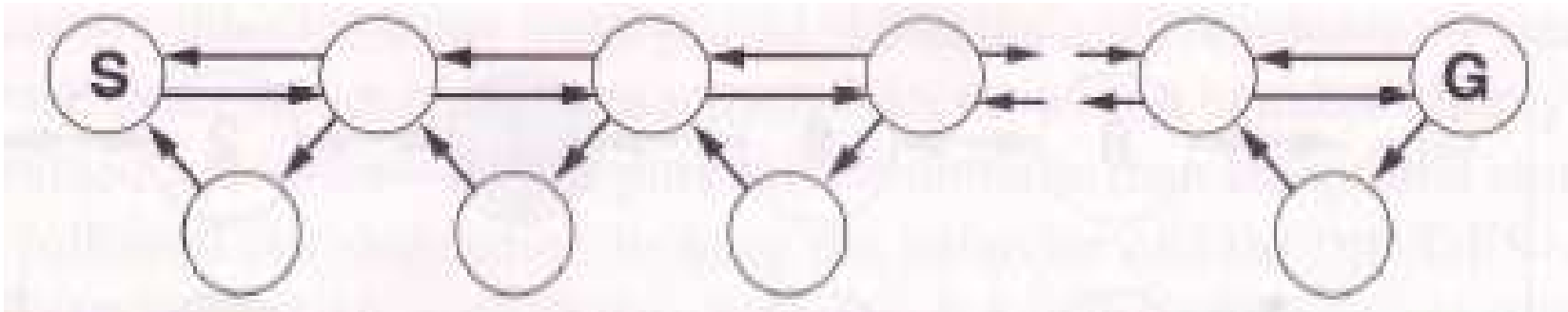
Random Walk

- Environment where random walk will take exponential many states to the goal. *Why?*



Random Walk

- Environment where random walk will take exponential many states to the goal. *Why?*



- At each step, backward progress twice as likely as forward progress.*

*Learning realtime A**

- Augmenting *hill climbing* with *memory* rather than randomness is more effective.
- Idea: store current best estimate $h(s)$ of the cost to reach goal.
- Heuristic estimate $h(s)$ updated as agent gains experience in the state-space.
- Cost to reach goal through a neighbor s' is cost to get to s' + estimated cost to goal – $c(s, a, s') + H(s')$.
- Called *learning realtime A** (LTRA*)

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

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 $H[s'] \leftarrow h(s')$

if s is not null

$\text{result}[s, a] \leftarrow s'$

$H[s] \leftarrow \min_b \text{LRTA}^*\text{-COST}(s, b, \text{result}[s, b], H)$

b (element of) ACTIONS(s)

$a \leftarrow$ an action b in ACTIONS(s') that minimizes $\text{LRTA}^*\text{-COST}(s', b, \text{result}[s', b], H)$

$s \leftarrow s'$

return a

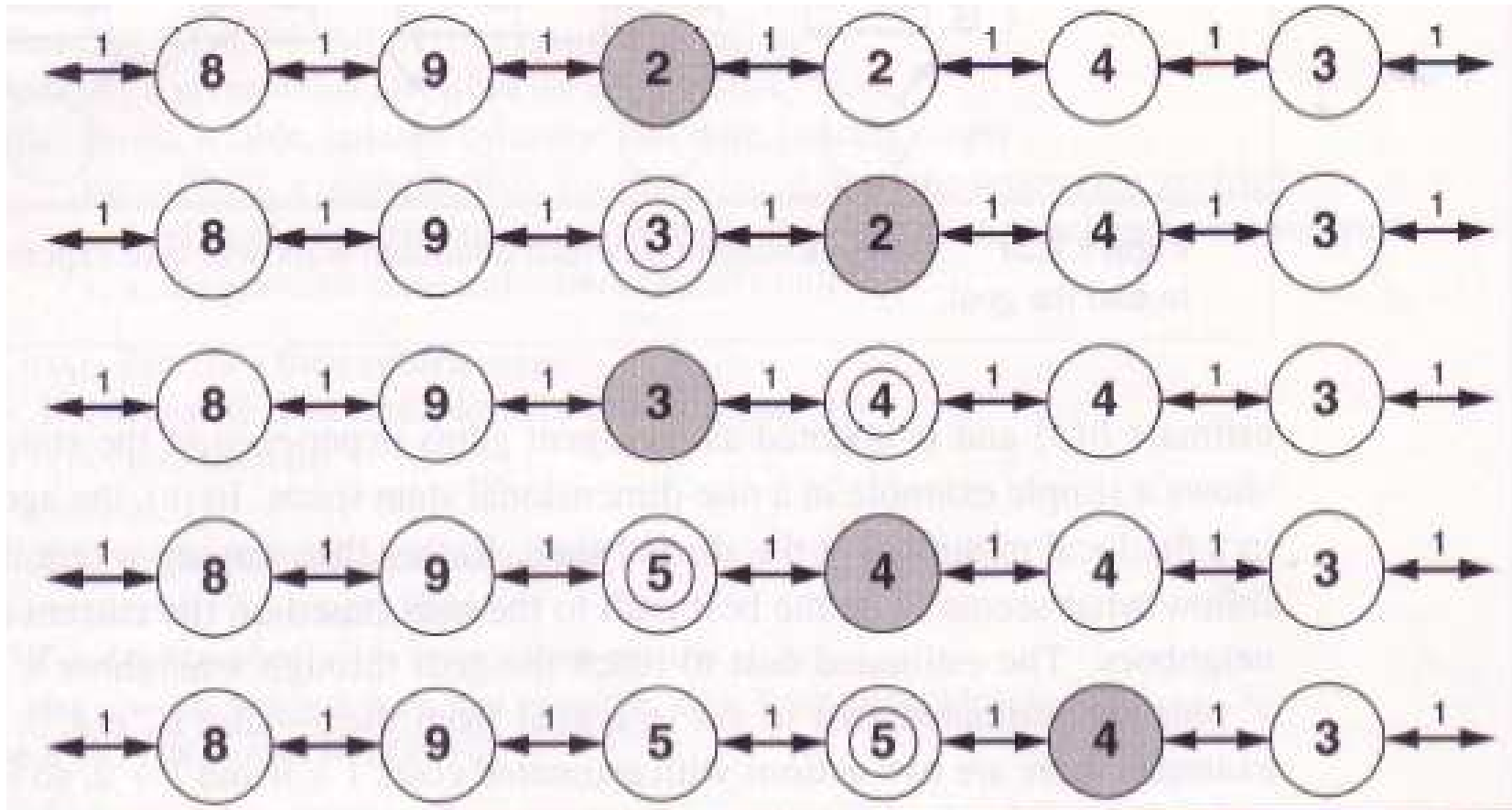
function LRTA*-COST(s , a , s' , H) returns a cost estimate

if s' is undefined then return $h(s)$

else return $c(s, a, s') + H[s']$

*Learning realtime A**

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



Summary

- Exploration problems arise when the *agent has no idea about the states and actions of its environments*.
- For safely explorable environments, online search agents can *build a map* and *find a goal if* it exists.
- *Updating heuristic estimates* from *experience* provides an effective method to escape from local minima.

Summary

- Many opportunities for learning:
 - Agent learns map of environment
 - Learn more accurate estimates of the value of a state.
 - Would be nice if agent could develop general rules it learned from exploring one state space to a similar state space, e.g., what does *up* mean?
 - More fun ahead!