CSE 473: Artificial Intelligence
Autumn 2014

Problem Spaces & Search

Dan Weld

With slides from
Dan Klein, Stuart Russell, Andrew Moore, Luke Zettlemoyer

Logistics

- Piazza
- PS0 Due for optional grading end of Wed 10/1

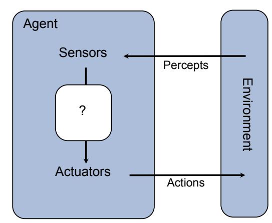
2

Outline

- Search Problems
- Uninformed Search Methods
  - Depth-First Search
  - Breadth-First Search
  - Uniform-Cost Search
- Heuristic Search Methods
  - Best First / Greedy Search

Agent *vs.* Environment

- An agent is an entity that perceives and acts.
- A rational agent selects actions that maximize its utility function.
- Characteristics of the percepts, environment, and action space dictate techniques for selecting rational actions.



Actions? Percepts?

5

Actions? Percepts?

Recommender System

6

Types of Agents

- Reflex

- Goal oriented

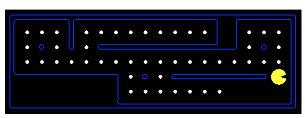
- Utility-based

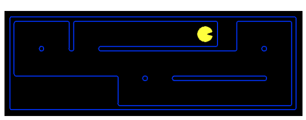
7

Famous Reflex Agents

Goal Based Agents

- Plan ahead
- Ask "what if"
- Decisions based on (hypothesized) consequences of actions
- Must have a model of how the world evolves in response to actions
- Act on how the world WOULD BE



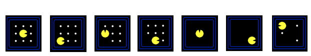


Search thru a Problem Space (aka State Space)

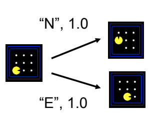
- Input:
  - Set of states
  - Operators [and costs]
  - Start state
  - Goal state [test]
- Output:
  - Path: start ⇒ a state satisfying goal test
    [May require shortest path]
    [Sometimes just need a state that passes test]

Example: Simplified Pac-Man

- Input:
  - A state space



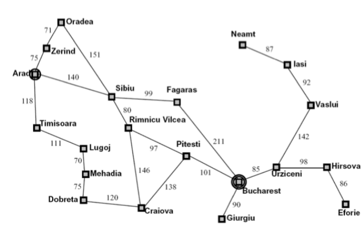
  - A successor function



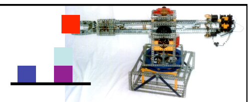
  - A start state
  - A goal test
- Output:

Ex: Route Planning: Arad → Bucharest

- Input:
  - Set of states
  - Operators [and costs]
  - Start state
  - Goal state (test)
- Output:

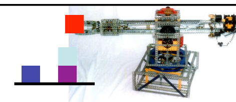


Ex: Blocks World



- Input:
  - Set of states
    Partially specified plans
  - Operators [and costs]
    Plan modification operators
  - Start state
    The null plan (no actions)
  - Goal state (test)
    A plan which provably achieves
    The desired world configuration
- Output:

Multiple Problem Spaces



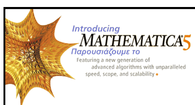
Real World

States of the world (e.g. block configurations)
Actions (take one world-state to another)

Robot's Head

- Problem Space 1
  - PS states =
    - models of world states
  - Operators =
    - models of actions

- Problem Space 2
  - PS states =
    - partially spec. plan
  - Operators =
    - plan modificat'n ops



Algebraic Simplification

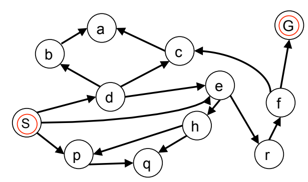
$$\partial_r^2 u = -\left[E' - \frac{l(l+1)}{r^2} - r^2\right] u(r)$$
$$e^{-2s}\left(\partial_s^2 - \partial_s\right) u(s) = -\left[E' - l(l+1)e^{-2s} - e^{2s}\right] u(s)$$
$$e^{-2s}\left[e^{\frac{1}{2}s}\left(e^{-\frac{1}{2}s}u(s)\right)'' - \frac{1}{4}u\right] = -\left[E' - l(l+1)e^{-2s} - e^{2s}\right] u(s)$$
$$e^{-2s}\left[e^{\frac{1}{2}s}\left(e^{-\frac{1}{2}s}u(s)\right)''\right] = -\left[E' - \left(l+\frac{1}{2}\right)^2 e^{-2s} - e^{2s}\right] u(s)$$
$$v'' = -e^{2s}\left[E' - \left(l+\frac{1}{2}\right)^2 e^{-2s} - e^{2s}\right] v$$

- Input:
  - Set of states
  - Operators [and costs]
  - Start state
  - Goal state (test)
- Output:

16

State Space Graphs

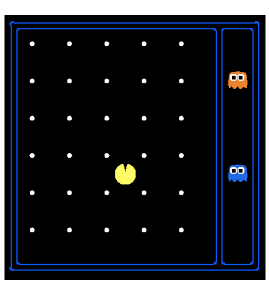
- State space graph:
  - Each node is a state
  - The successor function is represented by arcs
  - Edges may be labeled with costs
- We can rarely build this graph in memory (so we don't)



Ridiculously tiny search graph for a tiny search problem

State Space Sizes?

- Search Problem:
  Eat all of the food
- Pacman positions:
  10 x 12 = 120
- Pacman facing:
  up, down, left, right
- Food configurations: 2^{30}
- Ghost1 positions: 12
- Ghost 2 positions: 11

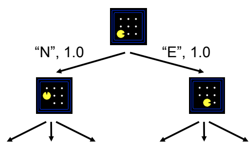


$120 \times 4 \times 2^{30} \times 12 \times 11 = 6.8 \times 10^{13}$

Search Methods

- Blind Search
  - Depth first search
  - Breadth first search
  - Iterative deepening search
  - Uniform cost search
- Local Search
- Informed Search
- Constraint Satisfaction
- Adversary Search

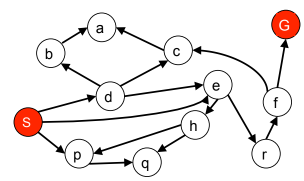
Search Trees



- A search tree:
  - Start state at the root node
  - Children correspond to successors
  - Nodes contain states, correspond to PLANS to those states
  - Edges are labeled with actions and costs
  - For most problems, we can never actually build the whole tree

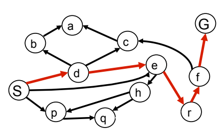
Example: Tree Search

State Graph:



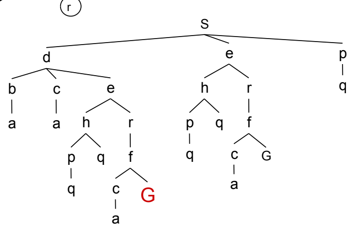
What is the search tree?

State Graphs vs. Search Trees



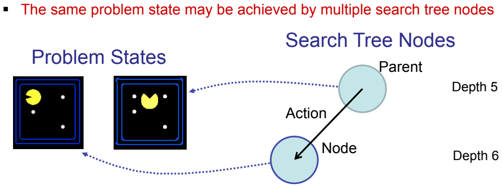
Each NODE in in the search tree denotes an entire PATH in the problem graph.

We construct both on demand – and we construct as little as possible.



States vs. Nodes

- Nodes in state space graphs are problem states
  - Represent an abstracted state of the world
  - Have successors, can be goal / non-goal, have multiple predecessors
- Nodes in search trees are plans
  - Represent a plan (sequence of actions) which results in the node's state
  - Have a problem state and one parent, a path length, a depth & a cost
  - The same problem state may be achieved by multiple search tree nodes



Building Search Trees

- Search:
  - Expand out possible plans
  - Maintain a fringe of unexpanded plans
  - Try to expand as few tree nodes as possible

General Tree Search

```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
   initialize the search tree using the initial state of problem
   loop do
      if there are no candidates for expansion then return failure
      choose a leaf node for expansion according to strategy
      if the node contains a goal state then return the corresponding solution
      else expand the node and add the resulting nodes to the search tree
   end
```

Detailed pseudocode is in the book!

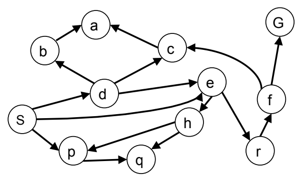
- Important ideas:
  - Fringe
  - Expansion
  - Exploration strategy

    which fringe node to expand next?

Review? Depth First Search

Strategy: expand deepest node first

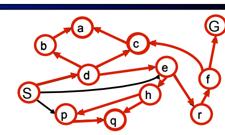
Implementation: Fringe is a LIFO queue (a stack)

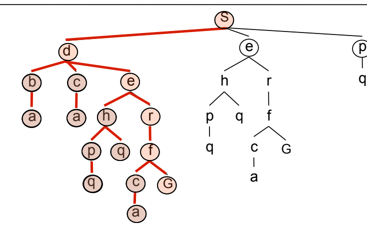


Review? Depth First Search

Expansion ordering:

(d,b,a,c,a,e,h,p,q,q,r,f,c,a,G)

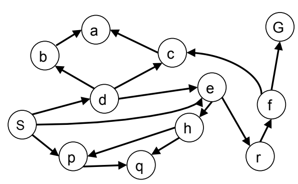




Review? Breadth First Search

Strategy: expand shallowest node first

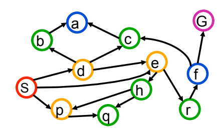
Implementation: Fringe is a FIFO queue



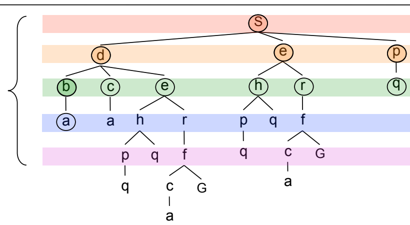
Review? Breadth First Search

Expansion order:

(S,d,e,p,b,c,e,h,r,q,a,a ,h,r,p,q,f,p,q,f,q,c,G)



Search Tiers



Search Algorithm Properties

- Complete? Guaranteed to find a solution if one exists?
- Optimal? Guaranteed to find the least cost path?
- Time complexity?
- Space complexity?

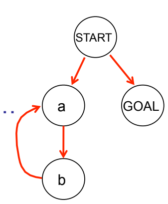
Variables:

n	Number of states in the problem
b	The maximum branching factor B (the maximum number of successors for a state)
C*	Cost of least cost solution
d	Depth of the shallowest solution
m	Max depth of the search tree

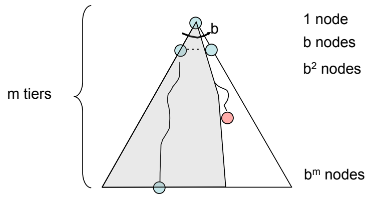
DFS

Algorithm		Complete	Optimal	Time	Space
DFS	Depth First Search	No	No	Infinite	Infinite

- Infinite paths make DFS incomplete…
  - How can we fix this?
  - Check new nodes against path from S
- Infinite search spaces still a problem



DFS

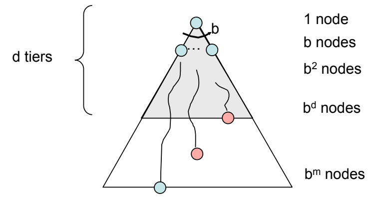


Algorithm		Complete	Optimal	Time	Space
DFS	w/ Path Checking	Y if finite	N	$O(b^m)$	$O(bm)$

* Or graph search – next lecture.

BFS

Algorithm		Complete	Optimal	Time	Space
DFS	w/ Path Checking	N unless finite	N	$O(b^m)$	$O(bm)$
BFS		Y	Y	$O(b^d)$	$O(b^d)$



Memory a Limitation?

- Suppose:
  - 4 GHz CPU
  - 32 GB main memory
  - 100 instructions / expansion
  - 5 bytes / node
- 40 M expansions / sec
  - Memory filled in 160 sec … 3 min

Comparisons

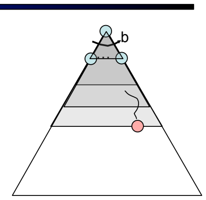
- When will BFS outperform DFS?
- When will DFS outperform BFS?

Iterative Deepening

Iterative deepening uses DFS as a subroutine:

1. Do a DFS which only searches for paths of length 1 or less.
2. If "1" failed, do a DFS which only searches paths of length 2 or less.
3. If "2" failed, do a DFS which only searches paths of length 3 or less.

….and so on.



Algorithm		Complete	Optimal	Time	Space
DFS	w/ Path Checking	Y	N	$O(b^m)$	$O(bm)$
BFS		Y	Y	$O(b^d)$	$O(b^d)$
ID		Y	Y	$O(b^d)$	$O(bd)$

Cost of Iterative Deepening

b	ratio ID to DFS
2	3
3	2
5	1.5
10	1.2
25	1.08
100	1.02

Speed

Assuming 10M nodes/sec & sufficient memory

	BFS			Iter. Deep.	
	Nodes	Time		Nodes	Time
8 Puzzle	10^5	.01 sec		10^5	.01 sec
2x2x2 Rubik's	10^6	.2 sec		10^6	.2 sec
15 Puzzle	10^{13}	6 days	1Mx	10^{17}	20k yrs
3x3x3 Rubik's	10^{19}	68k yrs	8x	10^{20}	574k yrs
24 Puzzle	10^{25}	12B yrs		10^{37}	10^{23} yrs

Why the difference?

Rubik has higher branching factor

15 puzzle has greater depth

# of duplicates

Slide adapted from Richard Korf presentation