Foundations of Artificial Intelligence

Problem-Solving as Search

CS472 – Fall 2007
Thorsten Joachims

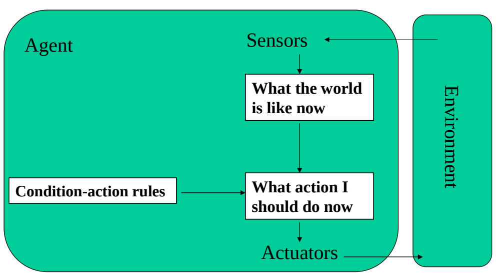
Intelligent Agents

Agent:
Anything that can be viewed as perceiving its **environment** through **sensors** and acting upon that environment through **actuators**.

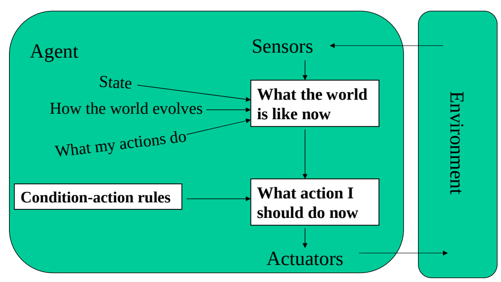
Agent Function:
Agent behavior is determined by the agent function that maps any given percept sequence to an action.

Agent Program:
The agent function for an artificial agent will be implemented by an agent program.

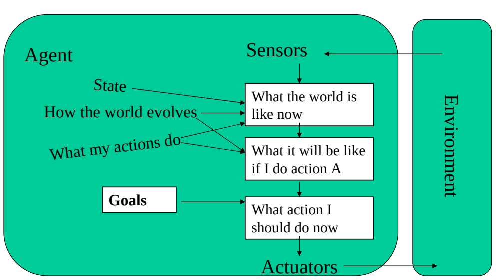
A Simple Reflex Agent



Agent with Model and Internal State



Goal-Based Agent



Problem Solving as Search

- **Search is a central topic in AI**
  - Originated with Newell and Simon's work on problem solving. Famous book:
    "Human Problem Solving" (1972)
  - Automated reasoning is a natural search task
  - More recently: Given that almost all AI formalisms (planning, learning, etc.) are NP-complete or worse, some form of search is generally **unavoidable (no "smarter" algorithm available).**

Defining a Search Problem

State space - described by

- **initial state** - starting state
- **actions** - possible actions available
- **successor function; operators** - given a particular state *x*, returns a set of < *action*, *successor* > pairs

Goal test - determines whether a given state is a goal state.

Path cost - function that assigns a cost to a path

The 8 Puzzle

5	4	
6	1	8
7	3	2

Initial State

1	2	3
8		4
7	6	5

Goal State

Cryptarithmetic

```
  SEND
+ MORE
------
 MONEY
```

Find (non-duplicate) substitution of digits for letters such that the resulting sum is arithmetically correct.

Each letter must stand for a different digit.

Solving a Search Problem: State Space Search

Input:

- Initial state
- Goal test
- Successor function
- Path cost function

Output:

- Path from initial state to goal state.
- Solution quality is measured by the path cost.

Generic Search Algorithm

```
L = make-list(initial-state)
loop
   node = remove-front(L) (node contains path
                           of how the algorithm
                           got there)
   if goal-test(node) == true then
      return(path to node)
   S = successors (node)
   insert (S,L)
until L is empty
return failure
```

Search procedure defines a search tree

Search tree

- **root node** - initial state
- **children of a node** - successor states
- **fringe of tree** - L: states not yet expanded

Search strategy - algorithm for deciding which leaf node to expand next.

- **stack**: Depth-First Search (DFS).
- **queue**: Breadth-First Search (BFS).

Solving the 8-Puzzle

5	4	
6	1	8
7	3	2

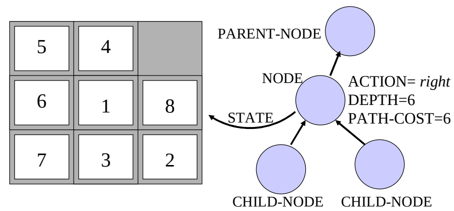
Start State

1	2	3
8		4
7	6	5

Goal State

What would the search tree look like after the start state was expanded?

Node Data Structure



Evaluating a Search Strategy

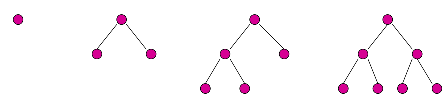
Completeness:
Is the strategy guaranteed to find a solution when there is one?

Time Complexity:
How long does it take to find a solution?

Space Complexity:
How much memory does it need?

Optimality:
Does strategy always find a lowest-cost path to solution? (this may include different cost of one solution vs. another).

Uninformed search: BFS

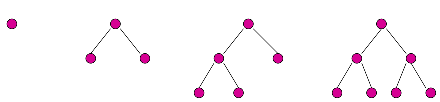


Consider paths of length 1, then of length 2, then of length 3, then of length 4,....

Time and Memory Requirements for BFS – $O(b^{d+1})$

Let b = branching factor, d = solution depth, then the maximum number of nodes *generated* is:

$b + b^2 + ... + b^d + (b^{d+1}-b)$



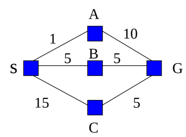
Time and Memory Requirements for BFS – $O(b^{d+1})$

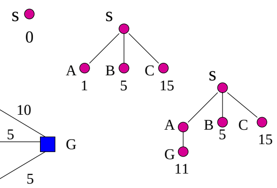
Example:
- b = 10
- 10000 nodes/second
- each node requires 1000 bytes of storage

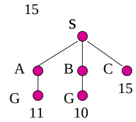
Depth	Nodes	Time	Memory
2	**1100**	**.11 sec**	**1 meg**
4	**111,100**	**11 sec**	**106 meg**
6	**10^7**	**19 min**	**10 gig**
8	**10^9**	**31 hrs**	**1 tera**
10	**10^{11}**	**129 days**	**101 tera**
12	**10^{13}**	**35 yrs**	**10 peta**
14	**10^{15}**	**3523 yrs**	**1 exa**

Uniform-cost Search

Use BFS, but always expand the lowest-cost node on the fringe as measured by path cost *g(n)*.

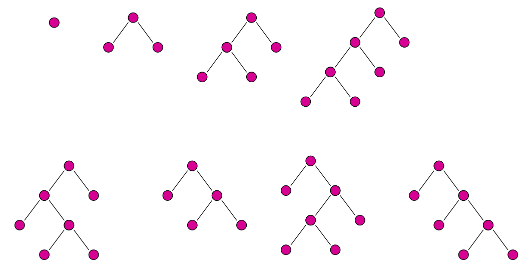






Requirement: $g(\text{Successor}(n)) \geq g(n)$

Uninformed search: DFS



DFS vs. BFS

	Complete	**Optimal**	**Time**	**Space**
BFS	YES	YES	$O(b^{d+1})$	$O(b^{d+1})$
DFS	Finite depth	NO	$O(b^m)$	$O(bm)$

m is maximum depth

Time

$m = d$: DFS typically wins
$m > d$: BFS might win
m **is infinite**: BFS probably will do better

Space

DFS almost always beats BFS

Which search should I use?

Depends on the problem.

If there may be infinite paths, then depth-first is probably bad.
If goal is at a known depth, then depth-first is good.

If there is a large (possibly infinite) branching factor, then breadth-first is probably bad.

(Could try **nondeterministic** search. Expand an open node at random.)

Iterative Deepening [Korf 1985]

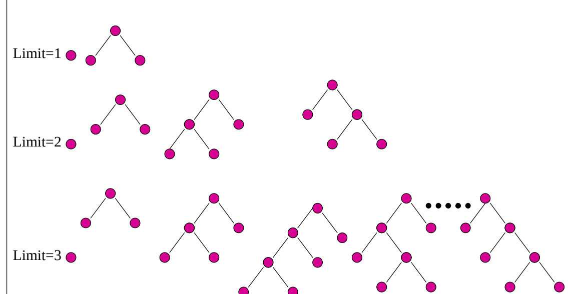
Idea:
Use an *artificial* depth cutoff, *c*.

If search to depth c succeeds, we're done.
If not, increase c by 1 and start over.

Each iteration searches using depth-limited DFS.

Limit=0

Iterative Deepening

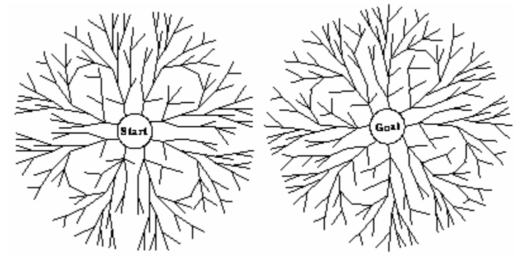


Cost of Iterative Deepening

space: *O(bd)* as in DFS, **time:** O(b^d)

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

Bidirectional Search



(from AIMA Figure 3.17)

Comparing Search Strategies

Criterion	Breadth -First	Uniform-Cost	Depth-First	Iterative Deepening	Bidirectional (if applicable)
Time	b^{d+1}	$b^{1+\frac{C^*}{\epsilon}}$	b^m	b^d	$b^{d/2}$
Space	b^{d+1}	$b^{1+\frac{C^*}{\epsilon}}$	bm	bd	$b^{d/2}$
Optimal?	Yes	yes	no	yes	yes
Complete?	Yes	Yes	No	Yes	Yes

*****Note that many of the ``yes's'' above have caveats, which we discussed when covering each of the algorithms.**