Foundations of Artificial Intelligence

CS472/3

Lecture #3

Bart Selman

Slide CS472–1

Today's Lecture

Problem Solving as Search, cont.

Uninfomed search

Readings: R&N, Chapter 3.

Slide CS472–2

Evaluating a Search Strategy

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

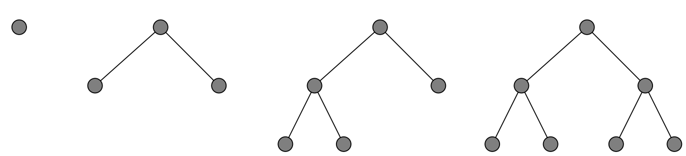
Time Complexity: how long does it take to find a solution?

Space Complexity: how much memory does it need?

Optimality: does the strategy find the highest-quality solution when there are several different solutions?

Slide CS472–3

Uninformed search: BFS



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

Slide CS472–4

Time and Memory Requirements for BFS – $O(b^d)$

Let b = branching factor, d = solution depth, then the maximum number of nodes expanded is: $1 + b + b^2 + ... + b^d$

Depth	Nodes	Time	Memory
0	1	1 millisecond	100 bytes
2	111	.1 seconds	11 kilobytes
4	11,111	11 seconds	1 megabyte
6	10^6	18 minutes	111 megabytes
8	10^8	31 hours	11 gigabytes
10	10^{10}	128 days	1 terabyte
12	10^{12}	35 years	111 terabytes
14	10^{14}	3500 years	11,111 terabytes

b = 10, 1000 nodes/second; 100 byte/node.

Slide CS472–5

BFS

Memory is serious problem!

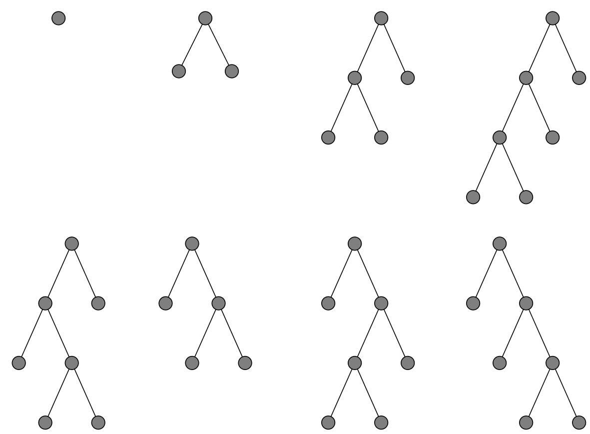
DFS a much better alternative.

Exponential time also a factor, but we'll see later on that a few more "tricks" enable us to effectively search huge state spaces.

E.g., chess: 10^{160} / planning: 10^{30}.

Slide CS472–6

Uninformed search: DFS



Slide CS472–7

DFS vs. BFS

	Complete?	Optimal?	Time	Space
BFS	YES	"YES"	b^d	b^d
DFS	finite depth	NO	b^m	bm

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

Slide CS472–8

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 DFS.

Iterative Deepening

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 DFS.

Slide CS472–11

Space requirements? Same as DFS. Each search is just a DFS.

Time requirements. Would seem very expensive!! **BUT** not much different from single BFS or DFS to depth d.

Reason: Almost all work is in the final couple of layers.
E.g., binary tree: 1/2 the nodes are in the bottom layer.
With $b = 10$, 9/10th of the nodes in final layer!

So, repeated runs are on much smaller trees (become exponentially smaller).

Slide CS472–12

Example: b=10, d=5, the number of nodes expanded in DFS

$1 + 10 + 100 + 1000 + 10,000 + 100,000 = 111,111$

bottom level is expanded once, second to bottom twice...

$(d+1)1 + (d)b + (d-1)b^2 + ... + 2b^{d-1} + 1b^d$ i.e.,:

$6 + 50 + 400 + 3,000 + 20,000 + 100,000 = 123,456$

only about 11% more!

Ratio of ID to DFS: (b+1)/(b-1).

Cost of repeating the work is not prohibitive.

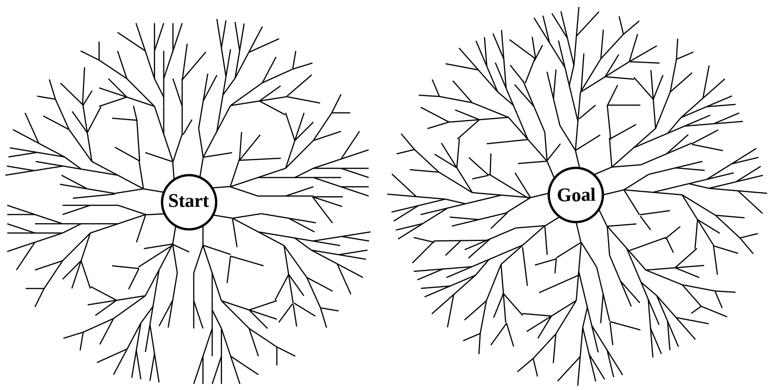
(Note: quite a clever insight.)

Cost of Iterative Deepening

space: $O(bd)$ (as DFS); **time:** $O(b^d)$

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

Bidirectional Search



Slide CS472–15

- Search forward from the start state and backward from the goal state simultaneously and stop when the two searches meet in the middle
- If branching factor = b from both directions, and solution exists at depth d, then need only $O(2b^{d/2}) = O(b^{d/2})$ steps.
- Example b = 10, d = 6 then BFS needs 1,111,111 nodes and bidirectional search needs only 2,222.
- Issues: what does it mean to search backwards from a goal? What if there is more than one goal state? (chess).

Slide CS472–16

Uniform-cost Search

Use BFS, but always expand the lowest-cost node on the fringe as measured by path cost $g(n)$ to find optimal solution.

See p. 75 R&N.

Comparing Search Strategies

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Time	b^d	b^d	b^m	b^l	b^d	$b^{d/2}$
Space	b^d	b^d	bm	bl	bd	$b^{d/2}$
Optimal?	Yes	Yes	No	No	Yes	Yes
Complete?	Yes	Yes	No	Yes, if $l \geq d$	Yes	Yes