Foundations of Artificial Intelligence

CS472/3 — Fall 1998
Lecture #3

Bart Selman

Slide CS472–1

Today's Lecture

Problem Solving as Search, cont.
Uninfomed search, wrap-up.
Informed Search

Readings: R&N, Chapter 3&4.

Slide CS472–2

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 CS472–3

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 CS472–4

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.)

Slide CS472–5

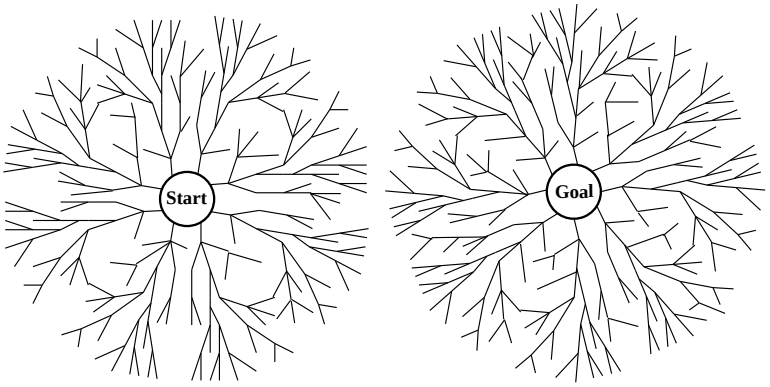
Cost of Iterative Deepening

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

Slide CS472–6

Bidirectional Search



Slide CS472–7

- 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 CS472–8

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.

Slide CS472–9

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

Slide CS472–10

Constraint Satisfaction Problems (CSP)

A powerful representation for (discrete) search problems.
Led to "constraint programming".

A **Constraint Satisfaction Problem (CSP)** is defined by:

X is a set of n variables $X_1, X_2, \ldots, X_n$,
each defined by its finite domain $D_1, D_2, \ldots, D_n$.

C is a set of constraints $C_1, C_2, \ldots, C_m$.

Constraints

A **constraint** C_i restricts the set of possible values
that can be assigned to the variables in the constraint.

In other words, a constraint specifies which values are
compatible for the variables in the constraint.

A **solution** is an assignment of values to the variables
that satisfies all constraints.

Example: Graph Coloring

Any NP-complete problem can be efficiently formulated as a finite CSP problem.

Why? — . . .

Constraint Satisfaction Problems (CSP)

For a given CSP the problem is one of the following:

- find all solutions
- find one solution
  - just a feasible solution, or
  - a "reasonably good" feasible solution, or
  - the optimal solution given an objective
- determine if a solution exists

How to View a CSP as a Search Problem?

Initial State – state in which all the variables are unassigned.

Operators – assign a value to a variable from a set of possible values.

Goal test – check if all the variables are assigned and all the constraints are satisfied.

Solving CSPs

figure backtrack search

Slide CS472–17

Branching Factor

Hypothesis 1 – any unassigned variable at a given state can be assigned a value by an operator: branching factor as high as sum of all domains.

Better approach – since order of variable assignment not relevant, consider as the successors of a node just the different values of a *single* unassigned variable: max branching factor = max domain.

Maximum Depth of Search Tree

n the number of variables; all the solutions are at depth n.

What are the implications in terms of using DFS vs. BFS?

Slide CS472–18

CSP – Goal Decomposed into Constraints

How to exploit it?

Backtracking only insert successors if *consistent* with constraints.

Constraint propagation "looking ahead" to remove inconsistencies.

Slide CS472–19

Forward Checking — each time variable is instantiated, remove other inconsistent values

Consistency — state is arc-consistent, if every variable has some value that is consistent with each of its constraints (consider pairs of variables)

K-Consistency generalizes arc-consistency. Consistency of groups of K variables.

Branching:

Most-constrained variable first; Most-constraining variable; Least-constraining value. (p. 105 R&N)

Slide CS472–20

Example forward-checking / arc consistency.

Slide CS472–21

Send More Money as a CSP

Variables:

$S = \{0, \ldots, 9\}$; $E = \{0, \ldots, 9\}$;

$N = \{0, \ldots, 9\}$; $D = \{0, \ldots, 9\}$; $M = \{0, \ldots, 9\}$;

$O = \{0, \ldots, 9\}$; $R = \{0, \ldots, 9\}$; $Y = \{0, \ldots, 9\}$;

Constraints:

send $= 1000 \times S + 100 \times E + 10 \times N + D$;

more $= 1000 \times M + 100 \times O + 10 \times R + E$;

money $= 10000 \times M + 1000 \times O + 100 \times N + 10 \times E + Y$;

send + more = money;

each letter has a different digit (S $\neq$ E, S $\neq$ N, etc);

Slide CS472–22

Dramatic recent progress in **Constraint Satisfaction.**

For example, we can now handle problems with **10,000** to **100,000** variables, and up to **1,000,000** constraints.

Slide CS472–23

Informed Methods: Heuristic Search

Informed Methods use problem-specific knowledge.

Heuristic search is an attempt to search the most promising paths first. Uses heuristics, or rules of thumb, to find the best node to expand next.

Relies on an *evaluation function* — indicates the desirability of expanding a node. E.g., *path cost.*

$h(n)$ = estimated cost of the cheapest path from the state at node n to a goal state (*Heuristic Function*: e.g., straight line on a map)

Slide CS472–24

Given a list of nodes to be expanded, choose the one that the heuristic function estimates as the most promising.

Slide CS472–25

Best-First Search

1. Set L to be the initial node(s).
2. Let n be the node on L that is "most promising" according to eval function ($h(n)$ / $f(n)$). If L is empty, fail.
3. If n is a goal node, stop and return it (and the path from the initial node to n).
4. Otherwise, remove n from L and add all of n's children to L (labeling each with its path from the initial node). Return to step 2.

Slide CS472–26