Foundations of Artificial Intelligence

CS472 — Fall 1998

Lecture #2

Bart Selman

Today's Lecture

Problem Solving as Search

Readings: R&N, Chapter 3.

Human Problem Solving

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 from of search is generally **unavoidable (no "smarter" alg. available)**.

Defining a Search Problem

State space – described by **an initial state** and the set of possible actions available (**operators**). A **path** is any sequence of actions that lead from one state to another.

Goal test – applicable to a single state to determine if it is the goal state.

Path cost – relevant if more than one path leads to the goal, and we want the shortest path.

Note: very general formulation. Can be somewhat unnatural.

Two Examples

Cryptarithmetic

(Newell and Simon 1972)

The 8-puzzle

Example I : Cryptarithmetic

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

Find substitution of digits for letters such that the resulting sum is arithmetically correct. Each letter must stand for a different digit.

Cryptarithmetic, cont.

States: a (partial) assignment of digits to letters.

Operators: the act of assigning digits to letters.

Goal test: all letters have been assigned digits and sum is correct.

Path cost: zero. All solutions are equally valid.

State Space Search

Input:

- Start state
- Goal state or goal test
- Operators

Output: legal sequence of nodes from initial node to goal node

Search space is generally **not** stored in its entirety.

example search space cryptarithmetic

DFS (depth-first search)

Cryptarithmetic

Is this (DFS) how humans tackle the problem?

And, if not, what do humans do?

Human problem solving appears much more **sophisticated!**

For example, we derive new constraints on the fly.

In a sense, we try to solve problems with **little** or **no** search!

In example, we can immediately derive that $M = 1$.

It then follows that $S = 8$ or $S = 9$. Etc. (derive more!)

Capturing such human problem solving strategies is surprisingly difficult. *For example, how do we know to first consider assigning M?*

Constraint programming techniques do provide some steps towards this kind of problem solving (next lecture).

Fortunately, computers are **very good at fast search!** Search speed can **compensate** for lack of higher-level insights into the problem structure.

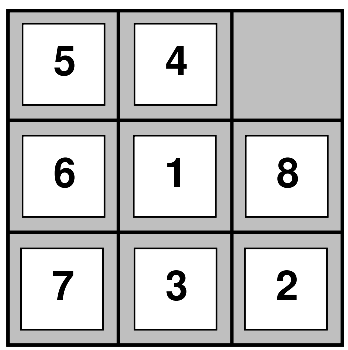
Example II : The 8-Puzzle

States: Specifies the location of each of the eight tiles in one of the nine squares

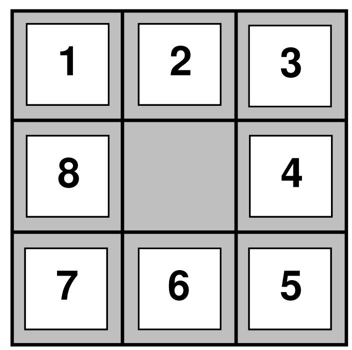
Operators: blank moves left, right, up, down

Goal test: state matches the goal configuration

Path cost: each step costs 1, so path cost = length of path

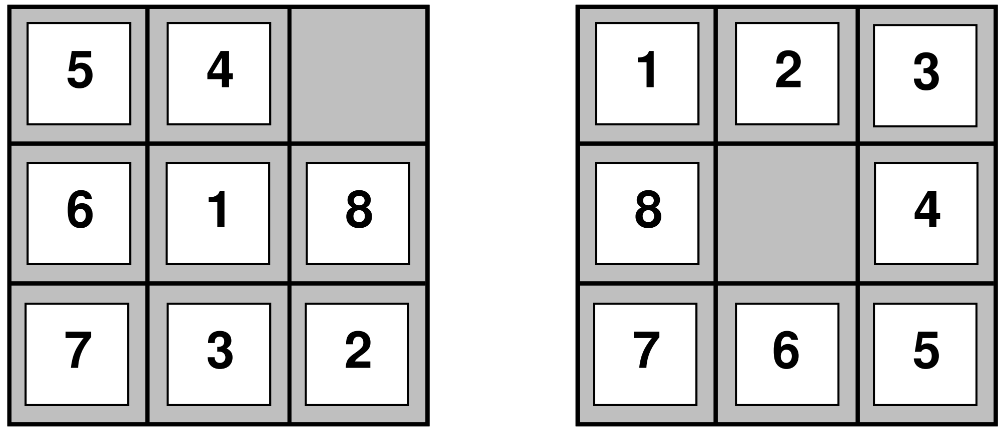


Start State



Goal State

Solving the 8-Puzzle



Start State **Goal State**

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

Solving a Search Problem

Search problems are solved by searching the state space.

Search process builds up a *search tree* over the search space.

Root = the initial state

Leaves = states that do not have successors in the tree (none exist or node has not been expanded yet).

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

Generic Search Algorithm

```
L = make-queue/stack(initial-state)
loop
        node = remove-front(L)
        if goal-test(node) = true return( node )
        S = successors(node, operators)
        insert(S,L)
until L is empty
return failure
```

Search procedure defines a search tree

- **root node — initial state**
- **children of a node — successors of the node**
- **fringe of tree — L: nodes not yet expanded**

stack: Depth-First Search (DFS).

queue: Breadth-First Search (DFS).

Aside: Actual implementation may not use stack/queue.

Evaluating a Search Strategy

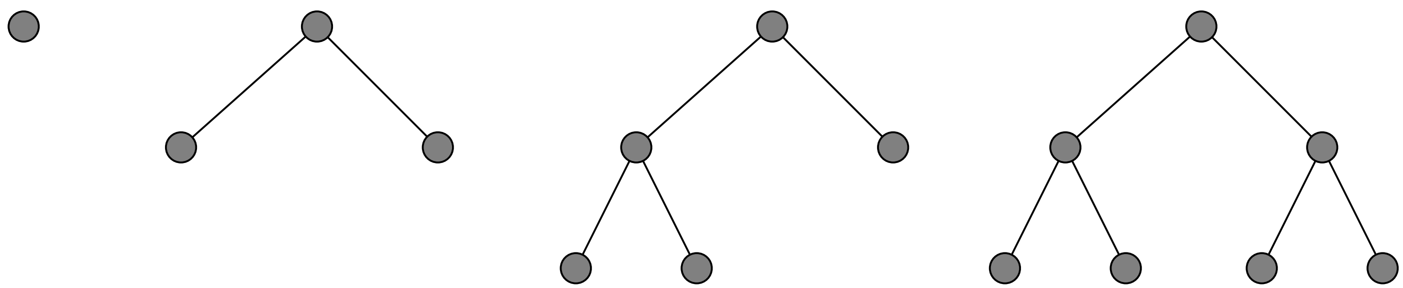
Completeness: is the strategy guaranteed to find a solution when there is one?

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?

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 – $O(b^d)$

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.

BFS

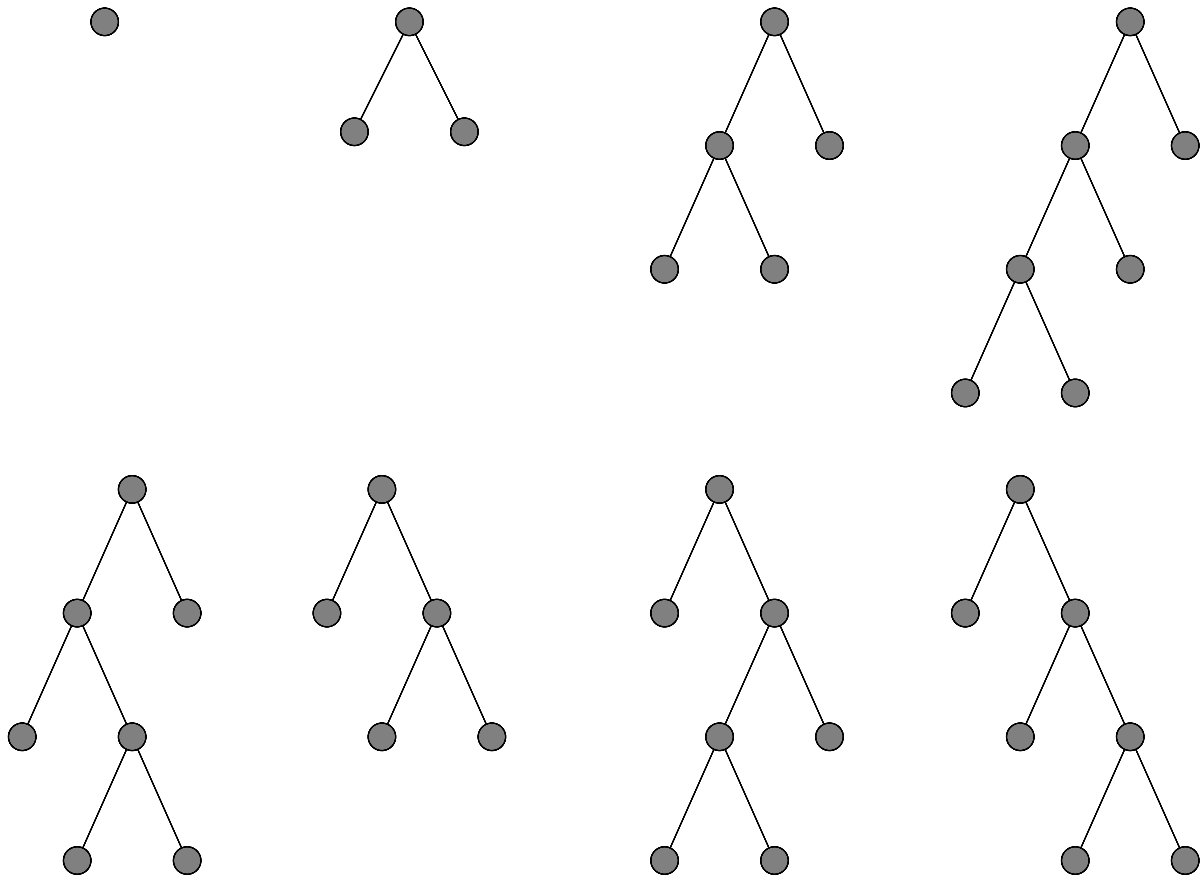
Memory is serious problem!

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}.

Uninformed search: DFS



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

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]

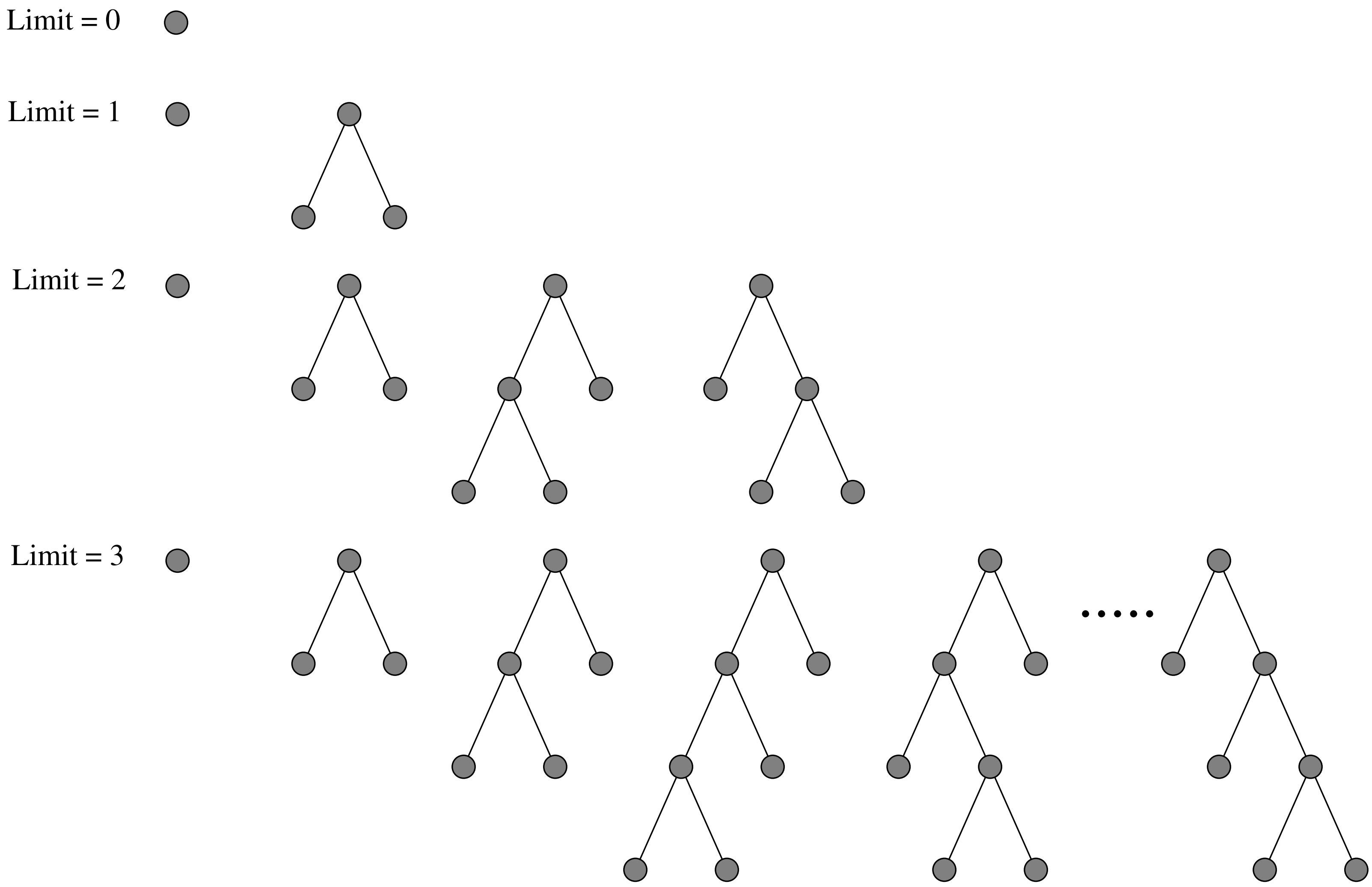
Idea:

Use an *artificial* depth cutoff, c.

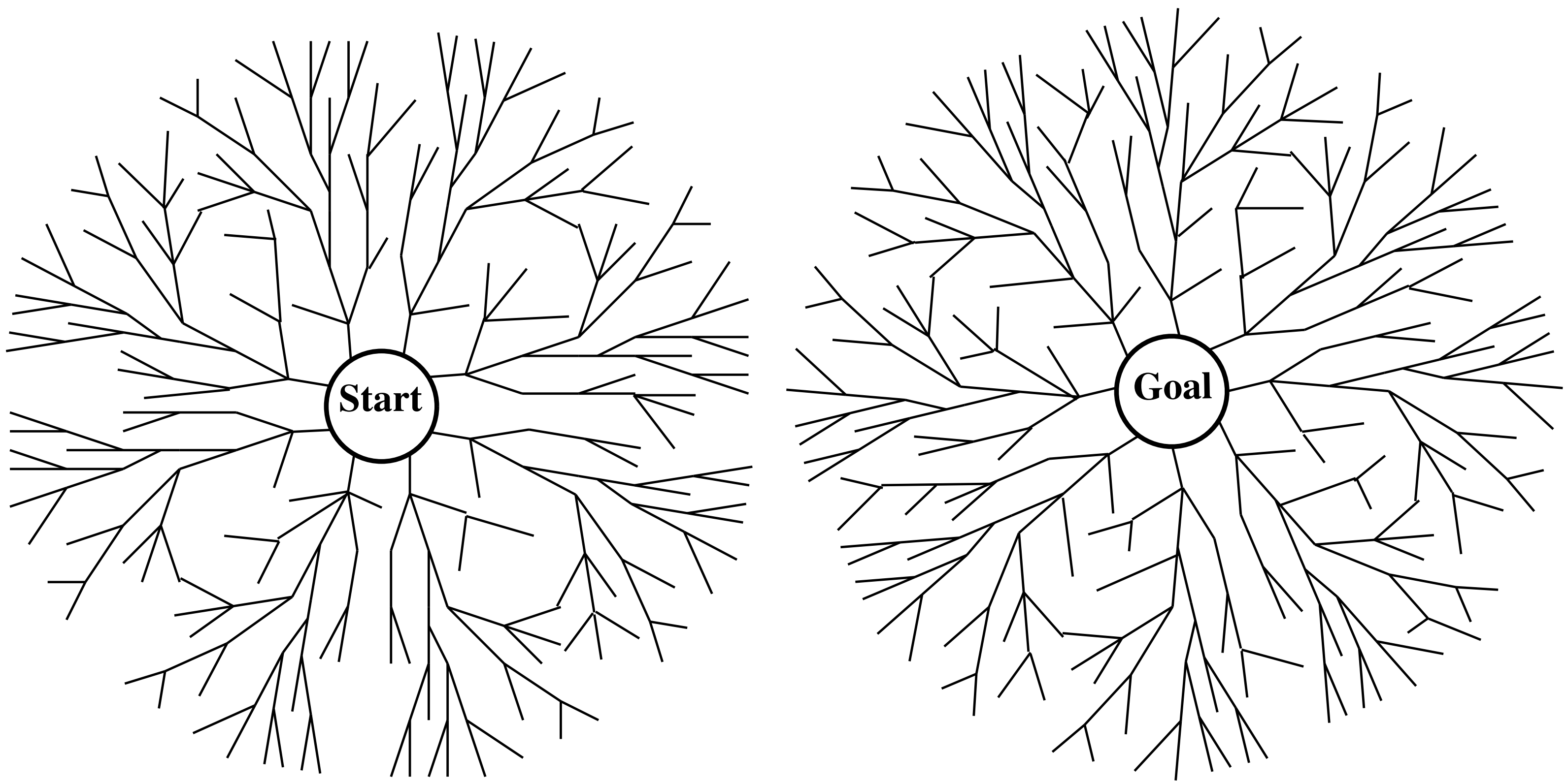
If search to depth c succeeds, we're done. If not, increase c by 1 and start over.

Each iteration searches using DFS.

Iterative Deepening



Bidirectional Search



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