Artificial Intelligence

Search I

Lecture 3

(August 25, 1999)

Tralvex (Rex) Yeap MAAAI MSCS

University of Leeds

Content: Search I

- Quick Review on Lecture 2
- Introduction
- Knowledge, Problem Types and Problem Formulation
- Search Strategies
  - General Search Problem
  - 4 Criteria for Evaluating Search Strategies
  - Blind Search Strategies: BFS, UCS, DFS, DLS, IDS, BDS
  - Comparing Search Strategies
- Students' Mini Research Presentation by Group A
- Class Activity 1: Various Blind Searches workouts for CMU Traffic problem
- Class Activity 2: Research Paper Reading: "All the Needles in a Haystack: Can Exhaustive Search Overcome Combinatorial Chaos? "
- What's in Store for Lecture 4

Quick Review on Lecture 2a & 2b

- Intelligent Agents
  - Three laws of robotics *(iop)*
  - What is an Agent
  - How Agents should Act
  - Rational behaviour depends on knowledge
  - Structure of an I. Agent *(a+p)*
  - Examples of Agents and their PAGE description
  - Five Major Agent Types *(trsgu)*
  - Shopping Example Activities
  - Agent Environments (adesc)
  - An Agent Portfolio

- Class Activity 1: To write the PAGE description for Robocup

- Class Activity 2: To write the characteristics of of the environment of Robocup domain.

- Class Activity 3: A paper on Intelligent Agent - Reading.

Introduction to Search

- Search is one of the most powerful approaches to problem solving in AI
- Search is a universal problem solving mechanism that
  - Systematically explores the **alternatives**
  - Finds the sequence of steps **towards a solution**
- Problem Space Hypothesis (Allen Newell, SOAR: An Architecture for General Intelligence.)
  - All **goal-oriented** symbolic activities occur in a **problem space**
  - Search in a problem space is claimed to be a completely **general model of intelligence**

Introduction to Search:
Popular Classical Problem Domains

- 8-puzzle
- Tower of Hanoi
- Missionaries and Cannibals
- Water Jug
- Vacuum World
- Wumpus World
- Block World
- Travelling Salesperson
- Maze
- Crossword Puzzle
- Crypt-arithmetic
- Wheel of Fortune
- Chess, Bridge, etc

Knowledge & Problem Types

Vacuum World Domain as an illustration

- Let the world be consist of only **2 locations** - Left and Right Box
- Each location may contain **dirt**
- The agent may be in **either box**
- There are **8 possible states**
- The agent can have **3 possible actions** - Left, Right and Suck

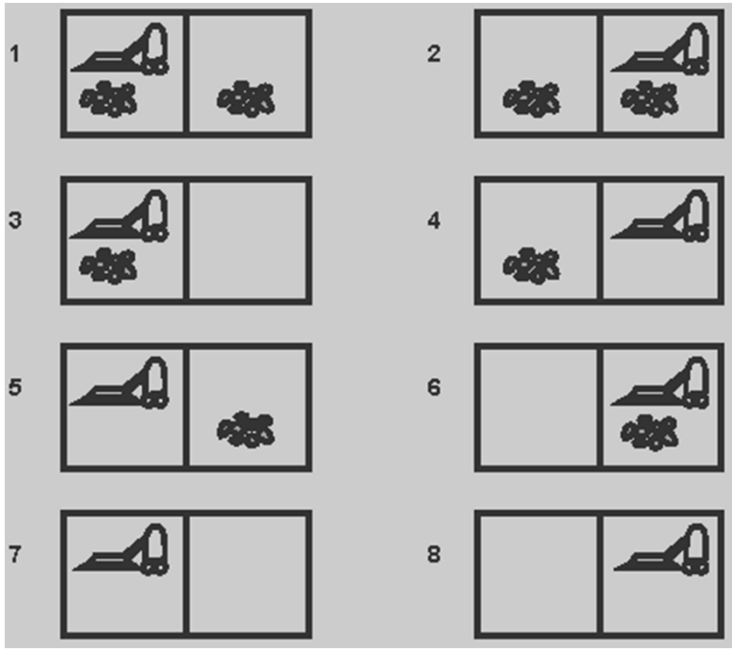


Fig 4.1 The 8 possible states of a Vacuum World

Knowledge & Problem Types

4 Problem Types

Single-state problems

- **exact** state known
- **effects** of actions **known**

eg. In vacuum world, if the initial state is 5, to achieve the goal, do action sequence [*Right*, *Suck*]

Multiple-state problems

- **one** of a set of states
- **effects** of actions **known**

eg. In vacuum world, where there is **no sensors**, the agent knows that there are 8 initial states, it can be calculated that an action of ***Right*** will achieve state **{2, 4, 6, 8}** and the agent can discover that the action sequence **[*Right, Suck, Left, Suck*]** is guaranteed to each the **goal**

Contingency problems

- **limited sensing**
- **conditional effects** of actions
- More complex algorithms involving **planning**

eg 1. In vacuum world, adding a simple Sense_Dirt, to use before the action *Suck*.
eg 2. Most of us keep our eyes open while walking,

Exploration problems

- **execution** 'reveals' states
- needs to **experiment** in order to survive
- **Hardest** task faces by an intel-agent.

eg 1. Mars Pathfinder
eg 2. Robot World Cup - Robocup

Knowledge & Problem Types

State Space for Vacuum World as a SS & MS problem

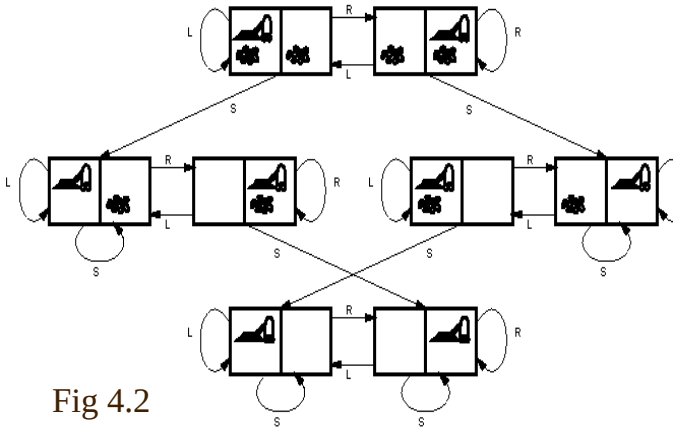


Fig 4.2

Single State Search Space ↖

Multiple-State Search Space ↗

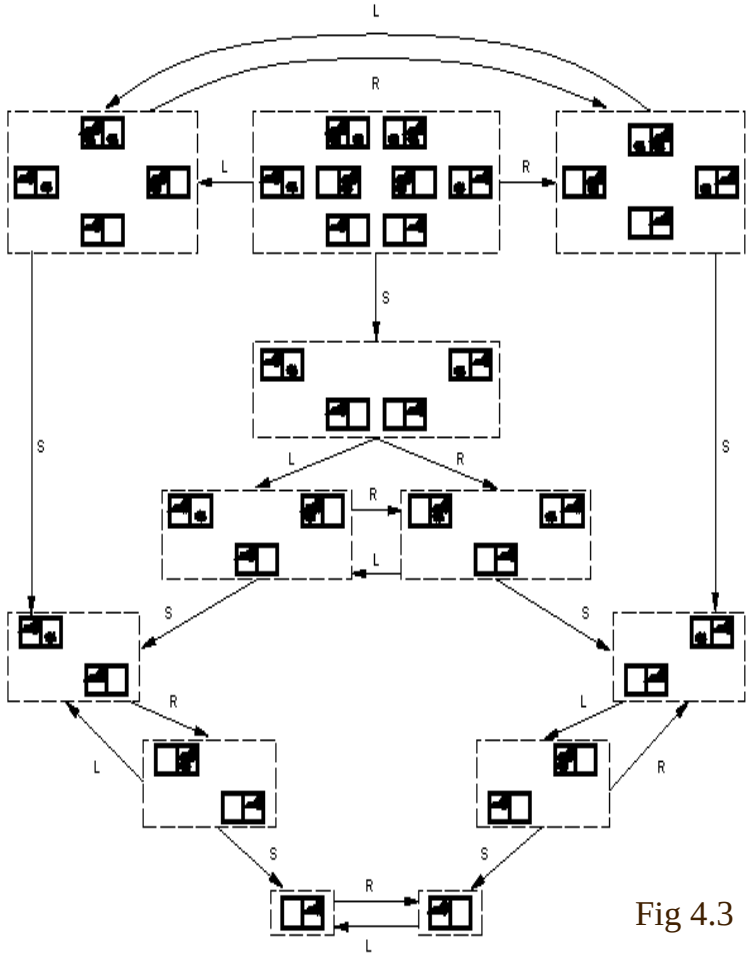


Fig 4.3

Defining a Search Problem

- **State Space**: described by **an initial space** 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 problem 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.

Toy Problems

(1) Vacuum World as a Single-state problem

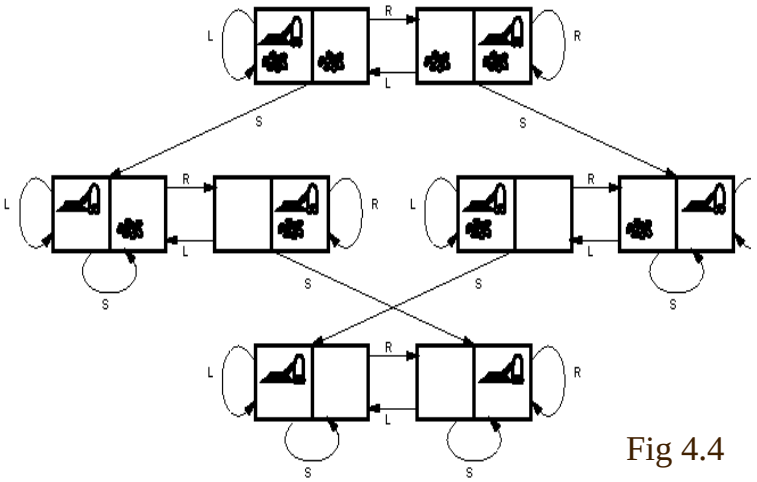


Fig 4.4

- **Initial State**: one of the 8 states shown above.
- **Operators** move Left, move Right, Suck.
- **Goal Test**: no dirt in any square.
- **Path cost**: each action costs 1.

Toy Problems

(2) Vacuum World as a Multiple-state problem

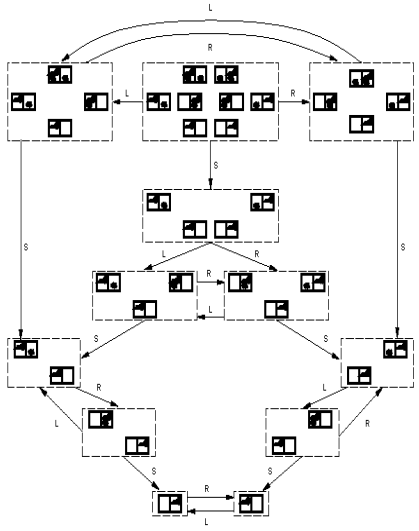


Fig 4.5

- **State sets**: subsets of state 1-8 shown in Fig 4.1 or Fig 4.2
- **Operators** move Left, move Right, Suck.
- **Goal Test**: all states in state set have no dirt.
- **Path cost**: each action costs 1.

Toy Problems
(3) 8-puzzle problem

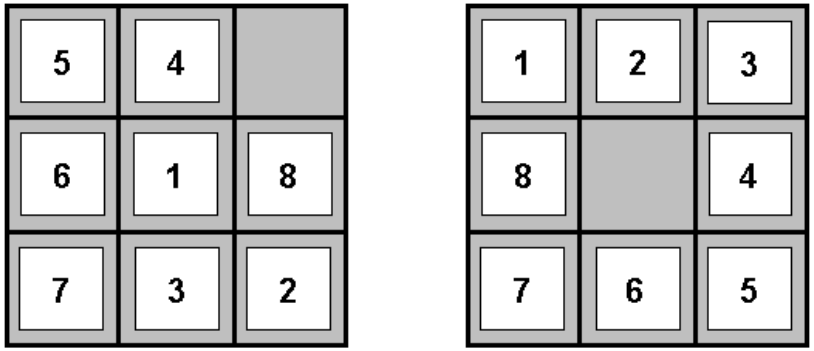


Fig 4.6

- **Initial State**: The location of each of the 8 tiles in one of the nine squares
- **Operators**: blank moves (1) Left (2) Right (3) Up (4) Down
- **Goal Test**: state matches the goal configuration
- **Path cost**: each step costs 1, total path cost = no. of steps

Toy Problems
(4) 8-queens problem

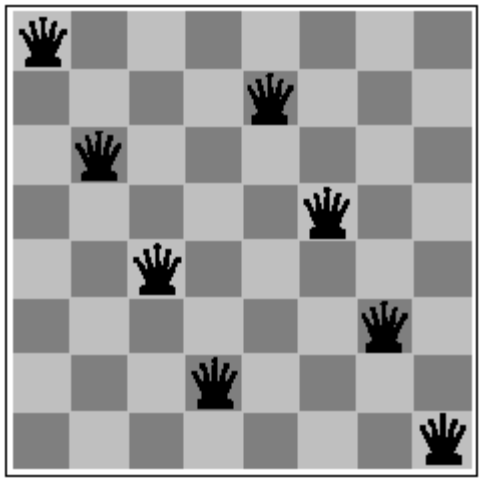


Fig 4.7

- **Initial State**: Any arrangement of 0 to 8 queens on board.
- **Operators**: add a queen to any square.
- **Goal Test**: 8 queens on board, none attacked.
- **Path cost**: not applicable or Zero (because only the final state counts, search cost might be of interest).

Toy Problems
(5) Cryptarithmetic


```
 FORTY          Solution: 29786     F=2, O=9, R=7, etc
+  TEN                     850
+  TEN                     850
 -----                   -----
 SIXTY                   31486
```

Fig 4.8

- **Initial State**: a cryptarithmetic puzzle with some letters replaced by digits.
- **Operators**: replace all occurences of a letter with a non-repeating digit.
- **Goal Test**: puzzle contains only digits, and represents a correct sum.
- **Path cost**: not applicable or 0 (because all solutions equally valid).

Real-world Problems

- **Route Finding** - computer networks, automated travel advisory systems, airline travel planning.
- **VLSI Layout** - A typical VLSI chip can have as many as a million gates, and the positioning and connections of every gate are crucial to the successful operation of the chip.
- **Robot Navigation -** rescue operations
- **Mars Pathfinder** - search for Martians or signs of intelligent lifeforms
- **Time/Exam Tables**

Search Strategies

- General Search Problem
- Criteria for evaluating search strategies
- Blind (un-informed) search strategies
  - Breadth-first search
  - Uniform cost search
  - Depth-first search
  - Depth-limited search
  - Iterative deepening search
  - Bi-directional search
- Comparing search strategies
- Heuristic (informed) search strategies

General Search Problem

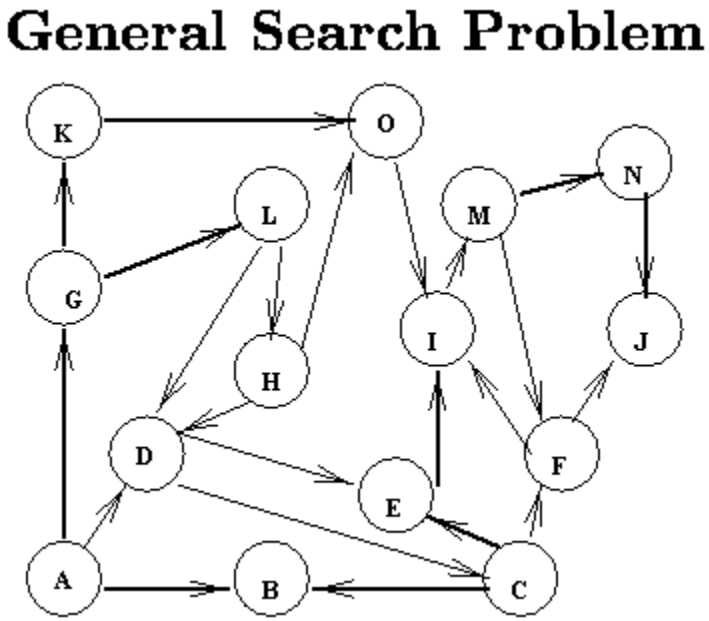


Fig 4.9

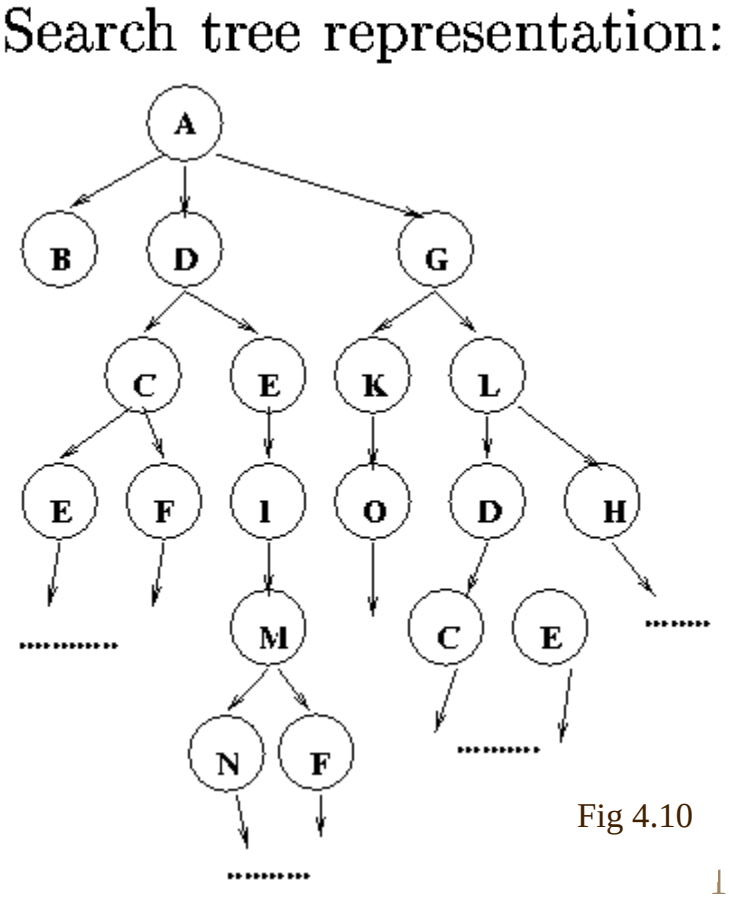


Fig 4.10

Criteria for Evaluating Search Strategies

Each of the search strategies are evaluated based on:

- **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 to perform the search?
- **Optimality**: does the strategy find the **highest-quality solution** when there are **several solutions**?

BS1. Breadth-First Search

- One of the simplest search strategy
- **Time** and **Space complex**
- Cannot be use to solve any but the **smallest problem**, see next page for a simulation.
- **Completeness**: Yes
- **Time complexity**: b^d
- **Space complexity**: b^d
- **Optimality**: Yes

(*b - branching factor, d - depth)*

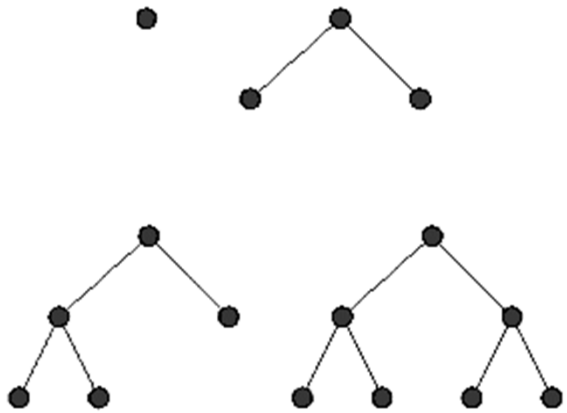


Fig 4.11 Breadth-first search tress after 0, 1, 2, and 3 node expansions (b=2, d=2)

BS1. Breadth-First Search (cont)

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

Fig 4.12

- Time and Memory requirements for a breadth-first search.
- The figures shown assume (1) branching factor b=10; (2) 1000 nodes/second; (3) 100 bytes/node

BS1. Breadth-First Search (cont)

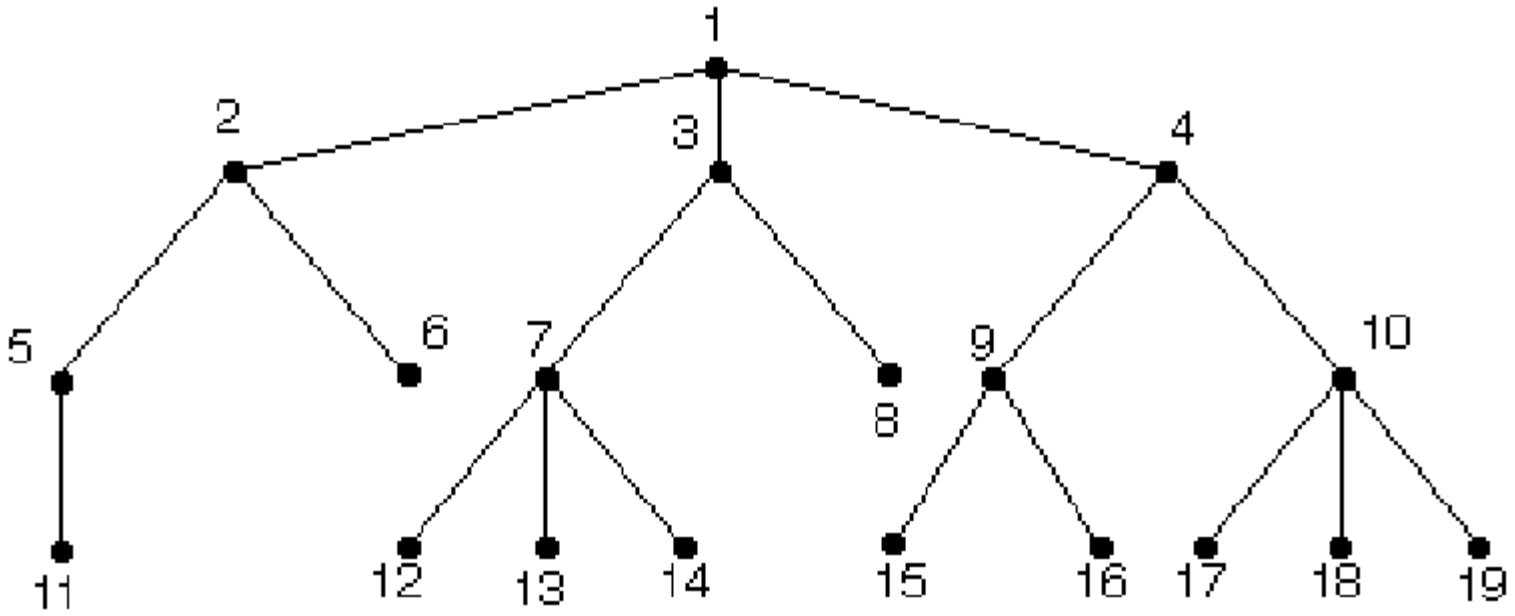


Fig 4.13 Breadth-first Tree Search (Numbers refer to order visited in search)

BS2. Uniform Cost Search

- BFS finds the ***shallowest*** goal state.
- Uniform cost search modifies the BFS by **expanding ONLY the lowest cost node** (as measured by the path cost *g(n)*)
- The **cost of a path** must **never decrease** as we traverse the path, ie. no negative cost should in the problem domain
- **Completeness**: Yes
- **Time complexity**: b^d
- **Space complexity**: b^d
- **Optimality**: Yes

BS2. Uniform Cost Search (cont)

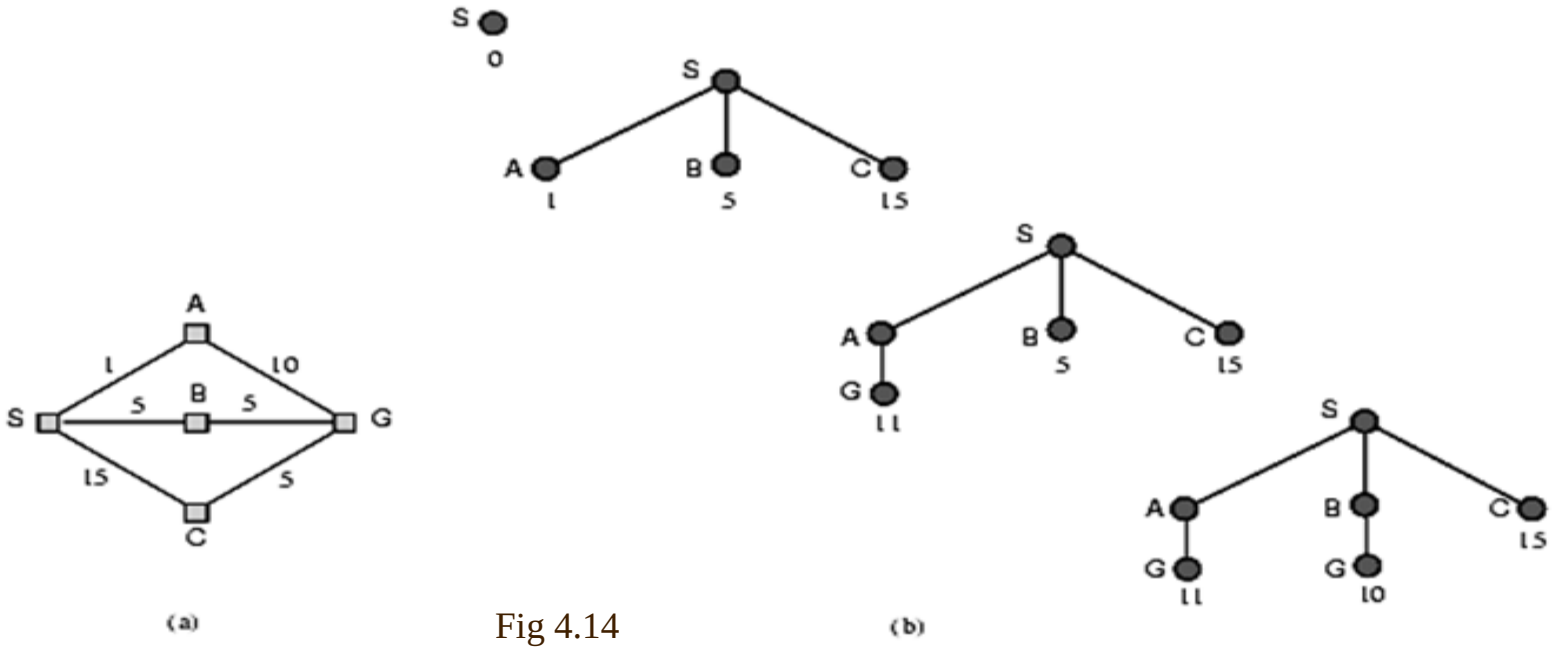


Fig 4.14

- A route finding problem. **(a)** The state space, showing the cost for each operator. **(b)** Progression of the search. Each node is labelled with a numeric path cost *g(n)*. At the final step, the goal node with g=10 is selected

BS3. Depth-First Search

- DFS always **expands one** of the nodes at the deepest level of the tree.
- The search **only go back** once it **hits a dead end** (a nongoal node with no expansion)
- DFS have **modest memory requirements**, it only needs to store a single path from root to a leaf node.
- Using the sample simulation from Fig 4.12, at depth d=12, **DFS** only requires **12 kilobytes** instead of **111 terabytes** for a **BFS** approach.
- For **problems that have many solutions**, **DFS** may actually be faster than BFS, because it has a good chance of finding a solution after exploring only a small portion of the whole space.

BS3. Depth-First Search (cont)

- One problem with DFS is that it can get **stuck** going down the wrong path.
- Many problems have **very deep** or even **infinite** search trees.
- DFS should be **avoided** for search trees with **large** or **infinite maximum depths**.
- It is common to implement a **DFS** with a **recursive function** that calls itself on each of its children in turn.

- **Completeness**: No
- **Time complexity**: b^m
- **Space complexity**: bm
- **Optimality**: No *(b-branching factor, m-max depth of tree)*

BS3. Depth-First Search (cont))

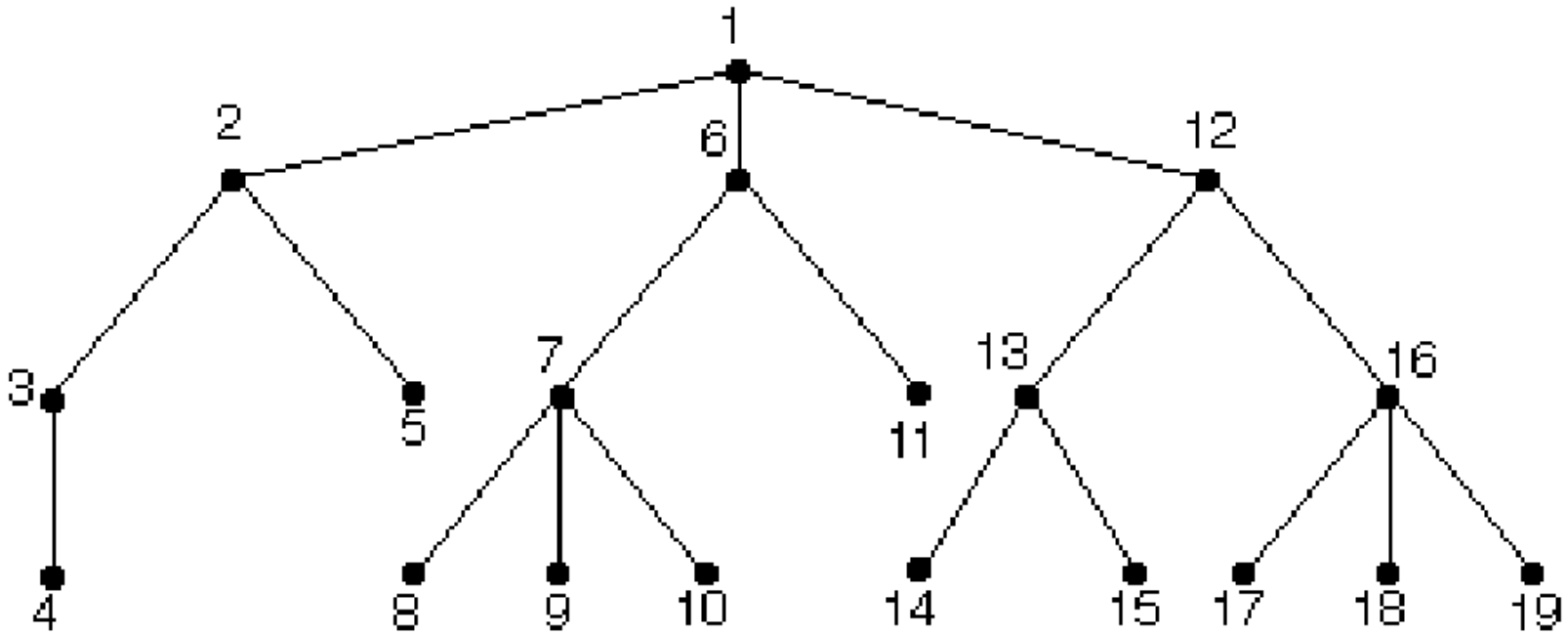


Fig 4.15 Depth-first Tree Search

BS4. Depth-Limited Search

- "**Practical**" DFS
- DLS avoids the pitfalls of DFS by imposing a cutoff on the **maximum depth** of a path.
- However, if we choose a depth limit that is too small, then DLS is not even complete.
- The time and space complexity of DLS is similar to DFS.

- **Completeness**: Yes, if $l >= d$
- **Time complexity**: b^l
- **Space complexity**: bl
- **Optimality**: No *(b-branching factor, l-depth limit)*

BS4. Depth-Limited Search (cont)

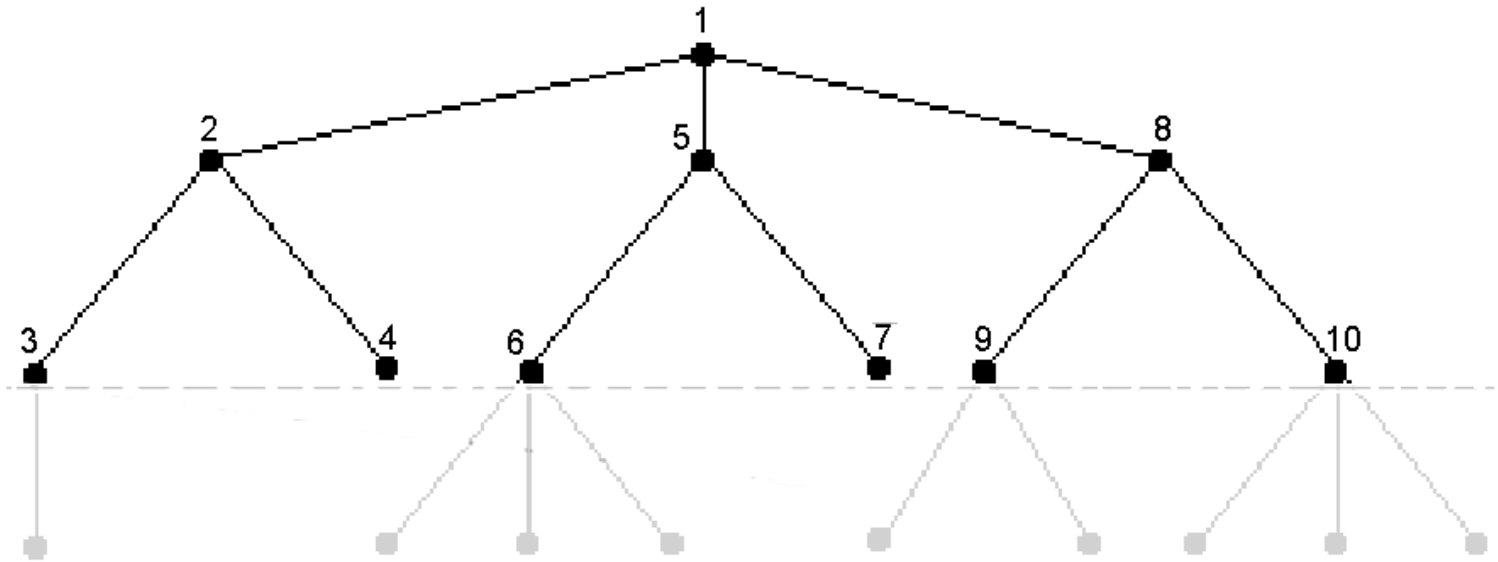


Fig 4.16

- Depth-first search trees for binary search tree. Same problem as Fig 4.15
- Depth limit, *dl = 2*

BS5. Iterative Deepening Search

- The **hard part** about DLS is **picking a good limit**.
- IDS is a strategy that sidesteps the issue of choosing the best depth limit by **trying all possible depth limits**: first depth 0, then depth 1, the depth 2, and so on.
- In effect, it combines the **benefits of DFS and BFS**.
- It is **optimal** and **complete**, like BFS and has **modest memory requirements** of DFS.

BS5. Iterative Deepening Search (cont)

- IDS may seem wasteful, because so many states are expanded multiple times.
- For most problems, however, the **overhead** of this multiple expansion is actually **rather small**.
- **IDS** is the **preferred** search method when there is a **large search space** and the **depth** of the solution is **not known**.

- **Completeness**: Yes
- **Time complexity**: b^d
- **Space complexity**: bd
- **Optimality**: Yes

BS5. Iterative Deepening Search (cont)

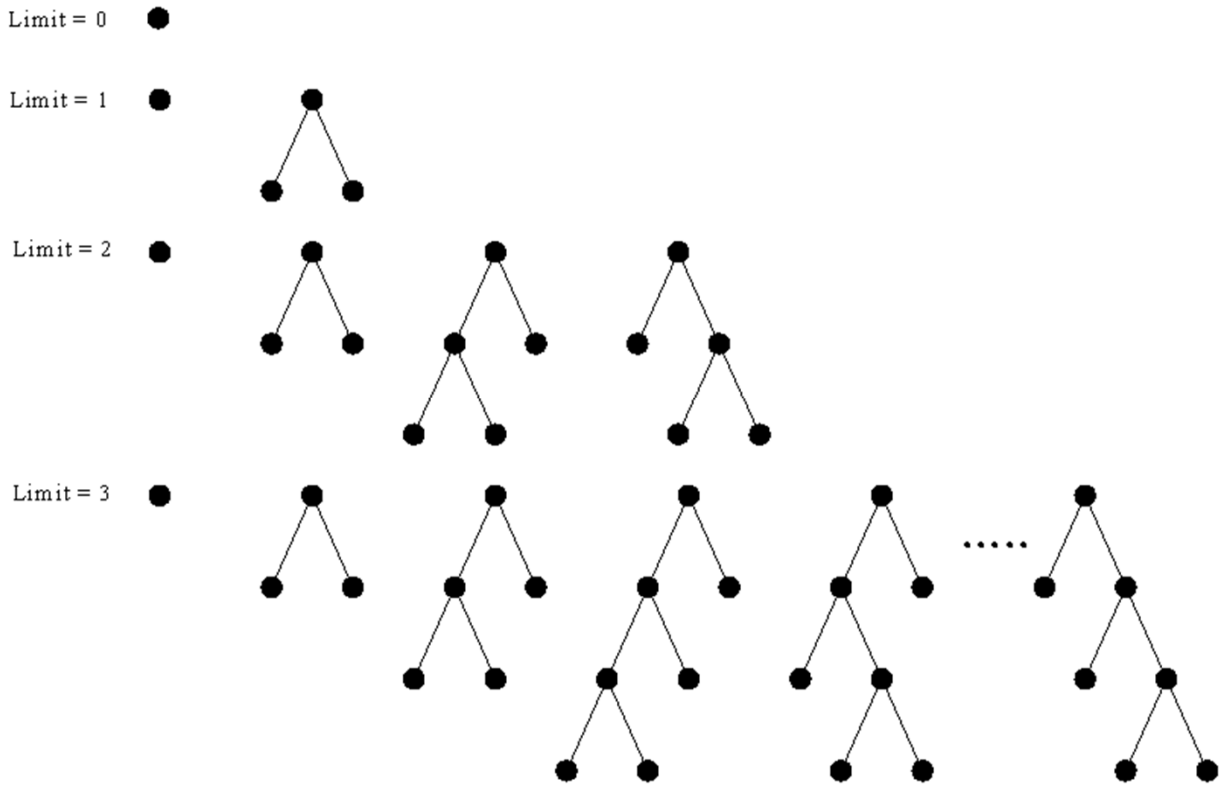


Fig 4.17 Four iterations of iterative deepening search on a binary tree

BS6. Bi-directional Search


- **Search forward** from the Initial state
- And **search backwards** from the Goal state..
- Stop when two meets in the **middle**.

- **Completeness**: Yes
- **Time complexity**: $b^{d/2}$
- **Space complexity**: $b^{d/2}$
- **Optimality**: Yes

BS6. Bi-directional Search (cont)

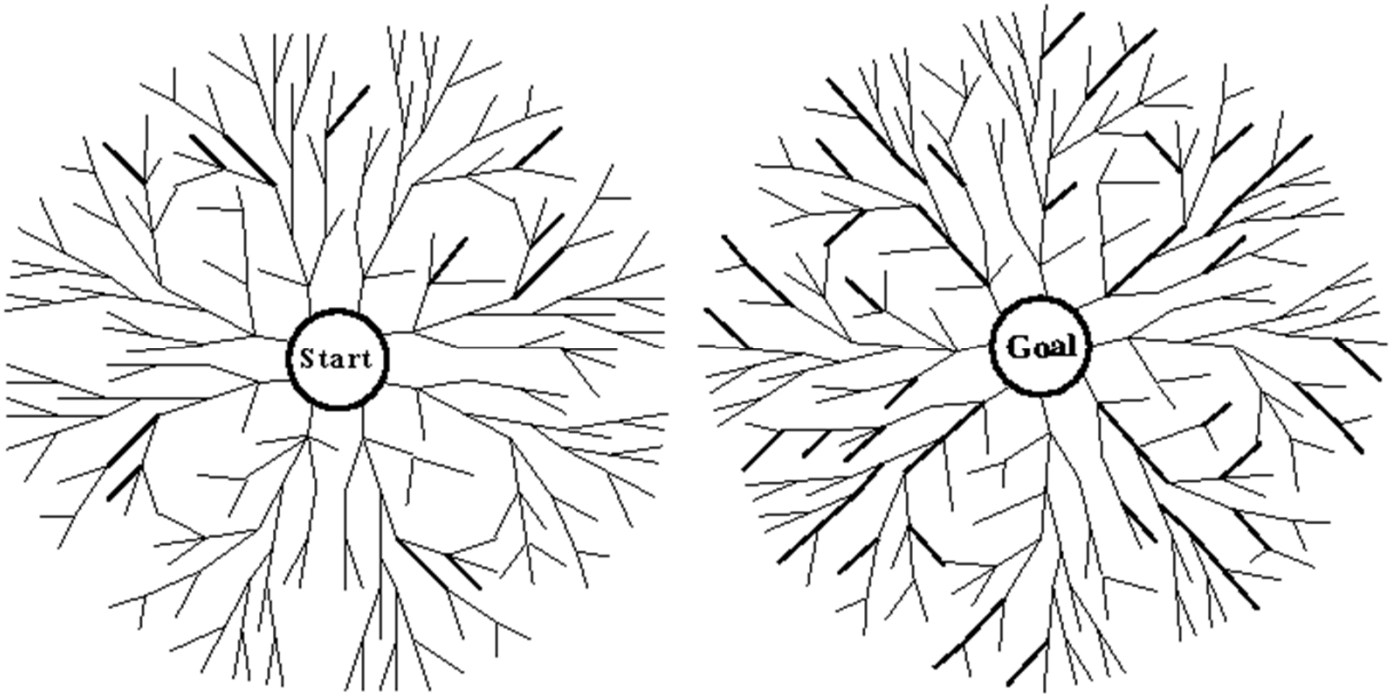


Fig 4.18 A schematics view of a bi-directional BFS that is about to succeed, when a branch from the start node meets a branch from the goal node

Comparing Blind 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

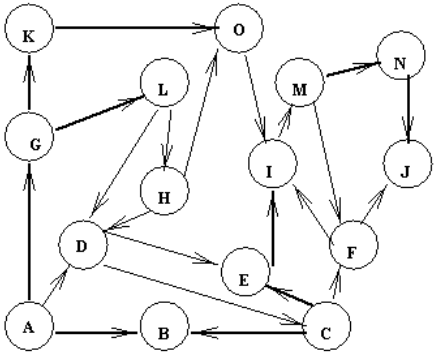
Fig 4.19

- Comparison of 6 search strategies in terms of the 4 evaluation criteria set forth in "Criteria for Evaluating Search Strategies"
- *b* - branching factor; *d* is the depth of the solution; *m* is the maximum depth of the search tree; *l* is the depth limit

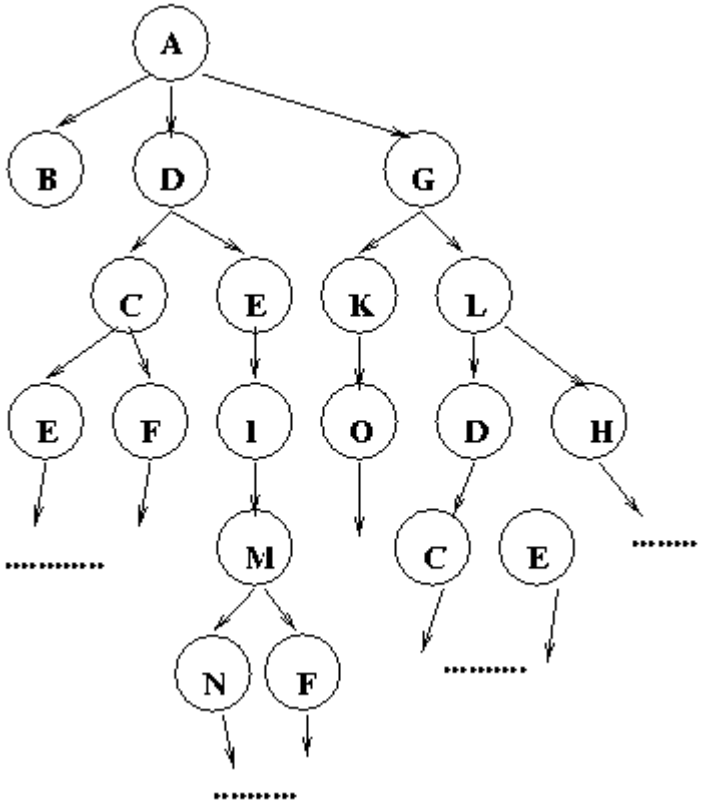
Class Activity 1:

Various Blind Searches workouts for CMU Traffic problem

Traffic Graph (raw)



Tree Representation



- Group A, B, C, D to use BFS approach
- Group D, E, F, G to use DFS approach

Class Activity 2: Research Paper Reading

"All the Needles in a Haystack: Can Exhaustive Search Overcome Combinatorial Chaos?"

1. The Scope of Search Techniques
2. Emerging Achievements of Exhaustive Search
3. The Role of Exhaustive Search: Speculation
4. State Spaces, their Size and Structure
5. Exhaustive Search Techniques
6. Case Study: Merrils and its Verification
7. Projects and Outlook
    - 7.1 Using Heuristics to Trade Accuracy for Space and Time: Pawn endings in chess
    - 7.2 Enumeration of Maximally Elastic Graphs
    - 7.3 Primes in Intervals of Fixed Length
    - 7.4 Quo Vadis Exhaustive Search?

References

What's in Store for Lecture 4

- Constrained Satisfaction Search
- Heuristic Search Methods
- Definition of a Heuristic Function
- Best First Search
  - An example - Sliding Tiles
  - Greedy Search
  - A* Search
- Class Activity 1: One paper on Heuristic Search Strategies - Reading
- Class Activity 1: DFS, BFS, BDS, UCS and GS workout for Romania Map problem
- Heuristics for an 8-puzzle problem
- Iterative Improvement Algorithms
  - Hill-climbing
  - Simulated Annealing
- Class Activity 2: Real-world Paper Reading - "Smart Moves - Intelligent Pathfinding"

End of Lecture 3

Good Night.