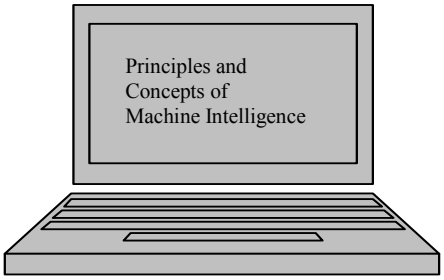
COVENANT UNIVERSITY
COLLEGE OF SCIENCE AND TECHNOLOGY

ARTIFICIAL INTELLIGENCE LECTURE NOTE

Dr. Olawande Daramola

Department of Computer and Information Sciences



Introduction

AI is one of the newest sciences. Work started in earnest soon after World War 11**,** and the name itself was coined in 1956. Along with molecular biology, AI is regularly cited as the "field I would most like to be in" by scientists in other disciplines. A student in physics might reasonably feel that all the good ideas have already been taken by Galileo, Newton, Einstein, and the rest. AI, on the other hand, still has openings for several ground-breaking contributions.

AI currently encompasses a huge variety of subfields, ranging from general-purpose areas, such as learning and perception to such specific tasks as playing chess, proving mathematical theorems, writing poetry, and diagnosing diseases. AI systematizes and automates intellectual tasks and is therefore potentially relevant to any sphere of human intellectual activity. In this sense, it is truly a universal field.

What is A.I.?

Some definitions of artificial intelligence has been organized into four categories in Figure 1.

Systems that think like humans	**Systems that think rationally**
"The exciting new effort to make computers think . . . *machines with minds,* in the full and literal sense." **(Haugeland, 1985)** "[The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning . . ." **(Bellman, 1978)**	"The study of mental faculties through the use of computational models." **(Chamiak and McDermott, 1985)** "The study of the computations that make it possible to perceive, reason, and act." **(Winston, 1992)**
Systems that act like humans	**Systems that act rationally**
"The art of creating machines that perform functions that require intelligence when performed by people." **(Kurzweil, 1990)** "The study of how to make computers do things at which, at the moment, people are better." **(Rich and Knight, 1991)**	"Computational Intelligence is the study of the design of intelligent agents." (Poole *et al.,* 1998) "A1 . . .is concerned with intelligent behavior in artifacts." **(Nilsson, 1998)**

Figure 1: Some Definition of A.I

A.I Problems Domain

- Early work focused on formal tasks like:
    - Game playing e.g. checkers-playing programs, where experience gained through playing with opponents were used to improve performance
    - Theorem proving e.g. Logic Theorist – which is an early attempt to prove mathematical theorem i.e. the first and second chapter of

principia mathematica (a book on mathematics), Gelernter's theorem (Geometry).
    - It appeared that computers could perform well at these formal tasks simply by being fast at exploring a large number of solution paths and then selecting the best one- but this assumption turned out to be false since no computer is fast enough to overcome the combinatorial explosion generated by most problems
    - General Problem Solver (GPS) – (Newell et al., 1963) an attempt to model commonsense reasoning and symbolic manipulation of logical expressions.
- The initial drawback of the early attempts was that the programs were designed to handle large amount of knowledge. But as the AI research progressed and techniques for larger amounts of knowledge were developed new tasks could reasonably be attempted. these include:
- **Some task Domain in A.I.**
- Mundane Tasks:
    - perception (vision, speech)
    - Natural Language processing (understanding, generation, translation)
    - Commonsense reasoning
    - Robot control
- Formal tasks
    - Games (Checkers, Chess, Ayo etc.)
    - Mathematics (Geometry, Logic, Calculus, Proving properties of Program)
- Expert Tasks
    - Engineering (Design, fault finding, manufacturing planning)
    - Scientific analysis
    - Medical analysis
    - Financial analysis

Developments in AI

1. *Problem Solving and Planning*: This deals with systematic refinement of goal hierarchy, plan revision mechanisms and a focused search of important goals.
2. *Expert Systems*: This deals with knowledge processing and complex decision-making problems.
3. *Natural Language Processing*: Areas such as automatic text generation, text processing, machine translation, speech synthesis and analysis, grammar and style analysis of text etc. come under this category.
4. *Robotics*: This deals with the controlling of robots to manipulate or grasp objects and using information from sensors to guide actions etc.
5. *Computer Vision*: This topic deals with intelligent visualisation, scene analysis, image understanding and processing and motion derivation.
6. *Learning*: This topic deals with research and development in different forms of machine learning.

7. *Genetic Algorithms*: These are adaptive algorithms which have inherent learning capability. They are used in search, machine learning and optimisation.
8. *Neural Networks*: This topic deals with simulation of learning in the human brain by combining pattern recognition tasks, deductive reasoning and numerical computations.

Application Areas of AI
Autonomous Planning and Scheduling, Game Playing, Autonomous Control, Diagnosis, Logistics Planning, Robotics, Language Understanding and Problem Solving

- **Some Pertinent Questions in A.I.**
  - What are our underlying assumptions about intelligence?
  - What kinds of technique will be useful for solving AI problems?
  - At what level of detail, if at all, are we trying to model human intelligence?
  - How will we know when we have succeeded in building an intelligent program?
- **Underlying Assumption**
  - The physical symbol system hypothesis [Newell and Simon , 1976].
  - The physical symbol system has the necessary and sufficient means for general intelligent actions.
  - Physical symbol system consist of :
  - Symbols which are physical patterns which occur as components of another entity called expression or symbol structure.
  - Contains a collection of processes that operate on expressions to produce other expressions. This includes processes of creation, modification, reproduction and destruction.
  - Therefore, a physical symbol system is a machine that produces through time an existing set of symbol structures.
  - Using the computer as a medium for this experimentation has been found to be true
  - The importance of the physical symbols system hypothesis is two fold:
  - It is a significant theory of the nature of human intelligence and so is of great interest to psychologists.
  - It also forms the basis of the belief that it is possible to build programs that can perform intelligent tasks now performed by people.

A.I. Techniques
- An AI technique is a method that exploits knowledge in solving a problem.
- **Nature of Knowledge**
  - It is voluminous
  - It is hard to characterize accurately
  - It is constantly changing
  - It differs from data by being organized in a way that corresponds to the way it will be used.

- **Questions**
  - Are there techniques that are appropriate for the solution of a variety of AI problems?
  - Can these techniques be useful in solving other problems?

- An AI technique must exploit knowledge in such a way that:
  - The knowledge captures generalization
  - It can be understood by people who must provide it
  - It can easily be modified to correct errors and reflect changes in the world and in our world view
  - It can be used in a great many situations even it is not totally accurate or complete
  - It can be used to help overcome its own sheer bulk by helping to narrow the range of possibilities that must be considered.

Note: that it is possible to solve AI problems using non-AI techniques (although the solution are not likely to be very good). It is also possible to apply AI techniques to the solution of non-AI problems.

Characteristics of AI Techniques

- Search – provides a way of solving important problems for which no more direct approach is available as well as a framework into which any direct techniques that are available can be embedded
- Use of Knowledge – Provides a way of solving complex problems by exploiting the structures of the objects that are involved.
- Abstraction – Provides a way of separating important features and variations from the many unimportant ones that would otherwise overwhelm any process.

Why develop AI programs?

Why do we attempt to model human performance:

- To test psychological theories of human performance
- To enable computers to understand human reasoning
- To enable people to understand computer reasoning
- To exploit what knowledge we can glean from people

How do we measure system's Intelligence?

- Turing Test (Alan Turing, 1950): A method to determine whether a system can think.

Characteristics of Problems

- Decomposable problems: problems can be decomposed into smaller or easier components?
- Ignorable problems: solution steps can be ignored when considered not necessary (e.g. theorem proving)
- Recoverable: in which solutions steps can be undone
- Irrecoverable: in which solutions steps cannot be undone
- Certain-outcome: lead to definite outcome
- Uncertain-outcome: produces a probability to lead to a solution

(the hardest problems to solve are those that are irrecoverable, uncertain-outcome) e.g. advising a lawyer who is defending a client who is standing trial for murder

- Problems that require absolutely good solution and those that require relatively good solution e.g. traveling salesman algorithm (Any-path/Best path) problem.
- Problem that require a solution as state or path.

Problem Spaces

Solving AI problems require four things:

- Define the problem precisely: this must include the specification of the initial situation(s) and the goal of the problem solving process
- Analyze the problem: the emergent features after analysis can help determine the appropriateness of various possible techniques for solving the problem.
- Isolate and represent the task knowledge that is necessary to solve the problem.
- Choose the best problem-solving technique(s) and apply it (them) to the particular problem.

State Space Definition of Problems

- AI problems are abstractions of a state space search containing the set of possible states, the operators and the set of rules for transformation.
- State spaces:
  - ✓ forms the basis of most AI problems
  - ✓ It allows for a formal definition of a problem in way that shows the initial state to a goal state using a set of permissible operations
  - ✓ It permits us to define the process of solving a particular problem as a combination of known techniques (each represented as a rule defining a single step in the state space search), the general technique of exploring the space in order to find some path from the current state to a goal state.
- ✓ Most AI problems are NP problems. Hence the search is a very important process in the solution of hard problems for which no more direct techniques are available.

Exercise

1. Find a good state space representation for the following problems:

i) Traveling salesman
ii) Missionaries and cannibals
iii) Tower of Hanoi

State Space for Missionaries and cannibals

3 Missionaries	Movement Across river	3 Cannibals
(0,0) (start state)		(3,3)
(1,1)	←mc	(2,2)
(0,1)	→m	(3,2)
(0,3)	←2c	(3,0)
(0,2)	→c	(3,1)
(2,2)	←2m	(1,1)
(1,1)	→mc	(2,2)
(3,1)	←2m	(0,2)
(3,0)	→c	(0,3)
(3,2)	←2c	(0,1)
(3,1)	→c	(0,2)
(3,3) (goal state)	←2c	(0,0)

Example

Water Jug Problem: You are given two jugs, a 4-gallon one and a 3-gallon one. Neither has any measuring made on it. There is a pump that can be used to fill the jugs with water. How can you get exactly 2 gallons of water into the 4- gallon jug?

Steps to formal description of a problem

- Define a state space that contains all possible configurations of the relevant objects (and perhaps some possible ones).

Note: It is possible to define this space without explicitly enumerating all the states it contains.

- Specify one or more states within that space that describe possible situations from which the problem solving may start these states are called the initial states.
- Specify the goal states that would be acceptable as solutions.
- Specify a set of rules that describe the actions (operators) available with respect to the following:
    - What unstated assumptions are present in the informal problem description?
    - How general should the rules be?
    - How much of the work required to solve the problems should be precomputed and represented in the rules?

Open problem

- Writing of programs that can produce formal descriptions from informal ones. This process is called operationalization. “For example consider the task of specifying precisely means to understand an English language sentence”.

Although such a specification must somehow be provided before we can design a program to solve the problem, producing such a specification itself a very hard problem.

- Generally problem can be solved by using the roles in the state space search, in combination with appropriate control strategy, to move goal state is found. Therefore the process is search is fundamental to the problem solving process.

Production Systems

Consists of:

- a set of rules, each consist of a left side (a pattern) that determines the applicability of the rule and a right side that describes the operation to be performed,
- one or more knowledge database (knowledge base) that contains relevant and appropriate information for the task.
- A control strategy that specifies the order in which the rules will be compared to the database and a way of resolving the conflict that arise when several rules match at ones.
- a rule applier

Examples: OPSS [Brownston et. al 1985], ACT * [Anderson 1983]

- Expert System Shells

- General Problem-solving architectures like SOAR [Liard et. al 1989]

Control strategies

Requirements

1. Causes motion: should be able to lead to a solution
2. Systematic: should be based on structure that bring motion e.g. tree or graph that can facilitate effective search process.

Types of Production systems

- Monotonic production systems: the application of a rule never prevents the later application of another rule that could have been applied at the time the first rule was selected.
- Non-Monotonic P.S: lacks the above attribute of Monotonic system
- Partially Commutative P.S.: if the application of a particular sequence of rules transforms state x into state y, then any permutation of those rules that is allowable also transforms state x into state y. (i.e. the order of selection of the set of rules is not important)
- Commutative P.S: is both monotonic and partially commutative
- Partially commutative, monotonic systems are useful for solving ignorable problems. Problems that involve creating new things rather than changing old ones are generally ignorable. e.g. Theorem proving, and making deductions from know facts.
- Non-Monotonic, Partially Commutative P.S. are good for problems where changes occur but can be reversed and in which order of operations is not critical. e.g. Physical manipulation systems like 'robot navigation on flat plane'

- Non-partially commutative P.S. are useful for problems in which irreversible changes occur e.g. declaring the process of chemical compound production (Chemical synthesis).

Characteristics of Production systems

- Production systems are a good way to describe the operations that are performed in a search for a solution to a problem.
- Questions
- Can Production system be described with characteristics that shed some light on how they can easily be implemented?
- If so, what relationship are there between problem types and types of production systems suited to solving the problems.

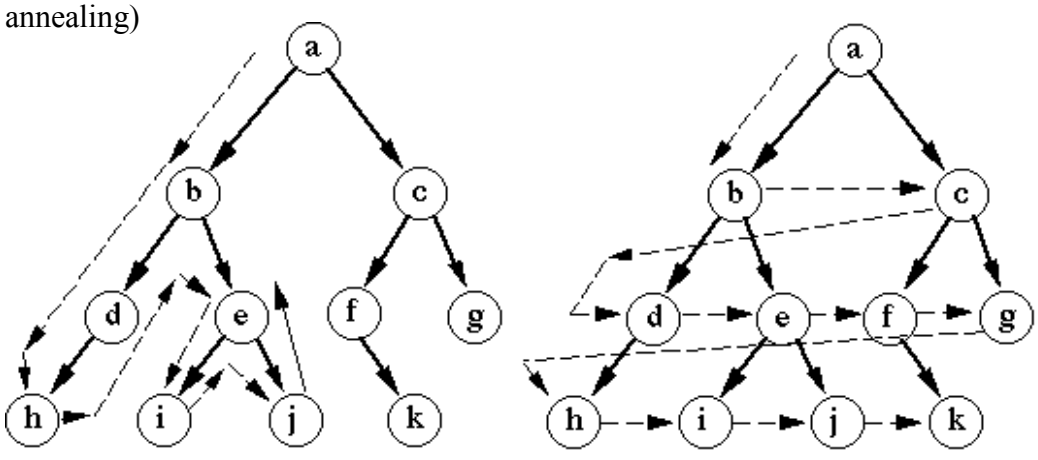
Searching Techniques

Search Strategies

There are two basic forms of search, namely informed search and uninformed search.

Uninformed search: This is a search strategy in which there is no information about the number of steps or the path from the current state to the goal – all they can do is distinguishing a goal state from a non-goal state. It is sometimes called **blind search**. Examples include; Breadth-first search, Uniform path cost search, Depth-first search, Depth-limited search, Iterative deepening search, and Bidirectional search.

Informed search: This is a search strategy in which we see how information about the state space can prevent algorithm from blundering about in the dark. It uses problem specific knowledge to find solution more efficiently. It also shows how optimization problems can be solved. Examples include: best-first search, Greedy search, A^* search, Heuristic search, Iterative improvement search (such as hill-climbing and simulated annealing)



Depth-first search Breadth-first search

Algorithm:

Breath-First Search

1. Create a variable called NODE-LIST and set to initial state
2. Until a goal state is found or NODE-LIST is empty do:
   a) Remove the first element from the NODE-LIST and call it E.
      If NODE-LIST was empty, quit
   b) for each way that each rule can match the state described in E do:
      i) Apply the rule to generate a new state
      ii) If the new state is a goal state, quit and return this state
      iii) Otherwise, add the new state to the end of NODE-LIST

Description

- Construct a tree with the initial state as its roots. Generate all offspring of the root by applying each of the applicable rules to the initial state. Thereafter, for each leaf node, generate all successors by the rules that are appropriate.
- Continue this process until some rule produces a goal state.

Exercise:

Apply the breadth-first search to the water jug problem?

Advantages of Breadth-First Search

- The algorithm therefore will not get trapped after exploring a wrong path
- If there is a solution the breadth first will find it, though it may take time. Also, if there are multiple solutions, the minimal solution will be found. This is because the longer paths are never examined until all the shorter ones have been examined.

Depth-first Search

- Pursues a single branch of the tree until it yields a solution or until a decision to terminate the path is made (i.e. when it reaches a dead end, when the length of the path exceeds the Futility Limit). Thereafter backtracking occurs. It backtracks to the most recently created state from which alternative moves are available.
- Chronological backtracking – because the ruler in which steps are undone depends only on temporal sequence in which the steps were originally made. The most recent steps is always the first to be undone.

Algorithm: Depth-First Search

- First the initial state is a goal state, quit and return success.
- Otherwise, do the following until success or failure is signaled:

(a) Generate a successor, E, of the initial state. If there are no more successors, signal failure.
(b) Call Depth-First Search with E as the initial state.
(c) If success is returned, signal success, otherwise continue this loop.

Advantages of Depth-First Search

- Requires less memory since only the nodes on the current path are stored, this contrast with breadth-first search, where all of the tree that has so far been generated must be stored.
- By chance (if care is taken in ordering the alternative successor states), may find solution without much searching.
- In breadth first all nodes at level n must be examined before any node on level n+1 can be examined. This is particularly significant if many acceptable solutions exist. Depth-first search can stop when one of them is found.

Disadvantage

- A wrong path may be followed, and it can get trapped if there are loops on that path.
- May find a solution on a loop path of a tree not necessarily the nominal path.

Note: the Best-First Search combines the strengths of these two algorithms to achieve a better implementation.

Iterative Deepening Search

When the initial depth cut-off is one, it generates only the root node and examines it. If the root node is not the goal, then depth cut-off is set to two and the tree up to depth 2 is generated using typical depth first search. Similarly, when the depth cut-off is set to m, the tree is constructed up to depth m by depth first search. One may thus wonder that in an iterative happening search, one has to regenerate all the nodes excluding the fringe nodes at the current depth cut-off. Since the number of nodes generated by depth first search up to depth h is:

$(b^h+1-1) / (b-1)$,

the total number of nodes expanded in failing searches by an iterative deepening search will be

$$(d-1) \{1 / (b-1) \} \sum_{h=0} (bh+1 -1)$$

$$\equiv b(bd - d) / (b-1)2.$$

The last pass in the algorithm results in a successful node at depth d, the average time complexity of which by typical depth first search is given by

$b(b^d + d) / 2 (b -1)$.

Thus the total average time complexity is given by

$b(b^d - d) / (b-1)^2 + b(b^d + d) / 2 (b -1)$.

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

Consequently, the ratio of average time complexity of the iterative deepening search to depth first search is given by

$\{(b+1) b^{d+1} / 2 (b -1)^2 \} : \{ b^{d+1} / 2 (b-1)\}$

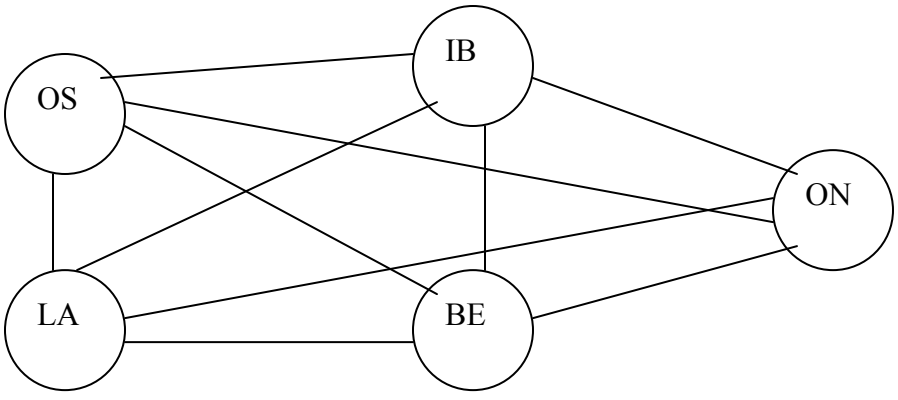
$= (b+1): (b -1)$.

The iterative deepening search thus does not take much extra time, when compared to the typical depth first search. The unnecessary expansion of the entire tree by depth first search, thus, can be avoided by iterative deepening. A formal algorithm of iterative deepening is presented below.

Procedure Iterative-deepening
Begin
1. Set current depth cutoff =1;
2. Put the initial node into a stack, pointed to by stack-top;
3. **While** the stack is not empty and the depth is within the
   given depth cut-off do
Begin
Pop stack to get the stack-top element;
if stack-top element = goal, return it and stop
else push the children of the stack-top in any order
      into the stack;
End While;
4. Increment the depth cut-off by 1 and repeat
   through step 2;
End.

The breadth first, depth first and the iterative deepening search can be equally used for Generate and Test type algorithms. However, while the breadth first search requires an exponential amount of memory, the depth first search calls for memory proportional to the largest depth of the tree. The iterative deepening, on the other hand, has the advantage of searching in a depth first manner in an environment of controlled depth of the tree.

Exercise
Discuss the traveling salesman problem?
A salesman has a list of cities, each of which he must visit exactly once. There are direct roads between each pair of cities on the list. Find the route the salesman should follow for the shortest possible trip that both starts and finishes at any one of the cities.



- Combinatorial explosion
- Branch and bound strategy

- NP problem.
No of path among cities = (N-1)!
Total time to perform the search = N!

Branch and bound Technique: Begin generating complete paths, keeping track of the shortest path found so far. Give up exploring any path as soon as its partial length becomes greater than the shortest path found so far.

Why can't we just check all possible tours using a computer?

Fact

If there are n cities, the number of possible tours is $(n-1)!/2$.

No. cities	No. tours	Time
5	12	12 microsecs
8	2520	2.5 millisecs
10	181,440	0.18 secs
12	19,958,400	20 secs
15	87,178,291,200	12.1 hours
18	177,843,714,048,000	5.64 years
20	60,822,550,204,416,000	1927 years

Heuristic Search

- A heuristic is a technique that improves the efficiency of a search process, possibly by sacrificing claims of completeness.
- It is a control structure that is not guaranteed to find the best answer but will always find a good answer.
- Heuristics are rules of the thumbs that can guide for correctness unlike algorithms.
- Heuristics help to find good though non-optimal solutions to NP problems.
- There are general-purpose heuristics and also domain-specific heuristics. e.g. Nearest neighbour heuristic, which works by selecting the locally superior alternative at each step.
- Error bounds (applicable to general purpose heuristics)
- Heuristics solves the problem of combinational explosion.
- Heuristic search is generally employed for two distinct types of problems: i) forward reasoning and ii) backward reasoning. In a forward reasoning problem we move towards the goal state from a pre-defined starting state, while in a backward reasoning problem, we move towards the starting state from the given goal. The former class of search algorithms, when realized with heuristic functions, is generally called heuristic Search for OR-graphs or the *Best First search Algorithms*. It may be noted that the best first search is a class of algorithms, and depending on the variation of the performance measuring

function it is differently named. One typical member of this class is the algorithm A*. On the other hand, the heuristic backward reasoning algorithms are generally called AND-OR graph search algorithms and one ideal member of this class of algorithms is the AO* algorithm. We will start this section with the best first search algorithm.

Arguments in favour of Heuristics

- Rarely do we actually need the optimum solution in the real world, a good approximation will usually serve very well
- Although Approximation produced by heuristics may not be very good in the worst case, worst cases rarely arise in the real world.
- Understanding why heuristics works or why it doesn't work, often leads to a deeper understanding of the problem.
- Domain specific heuristic knowledge can be incorporated into a rule-based search procedure by:
- Putting them in rules
- As heuristic function that evaluates individual problem states and determines how desirable they are.

- Heuristic Function is a function that maps from problem state description to measures of desirability, usually represented as numbers. The value of the heuristic function at any given node is meant to be as good an estimate as possible whether that node is on the derived path to a solution. It specifies the level of importance of that node to the path of solution. It is designed to efficiently guide a search process toward a solution. The purpose of the heuristic function is to guide the search process in the most profitable direction by suggesting which path to follow first when more than one is available.

Issues in the Design of Search Programs

- Every search process can be viewed as the traversal of a tree structure
- The node represents a problem state
- Each arc represents a relationship between the state nodes its connects
- In most programs the rules are not represented explicitly, rather the tree is represented implicitly in the rules and generates explicitly only those paths that they decide to explore.
- (Therefore, keep in mind the distinction between implicit search trees and the explicit partial search tree that are actually constructed by the search tree that are actually constructed by search program.)
- The direction of search (forward / backward reasoning.)
- Selection of applicable rules (matching)
- How to represent each node of the search process (Knowledge representation problem and the Frame problem)

Search Tree/Graph

- The concept of search graph eliminates the repeated traversal of previously expanded and process nodes that may reappear. This makes the search space to

be an arbitrary directed graph rather than a tree. The graph differs from a tree in that several paths may come together as a node. (See Figure 1.3 below)

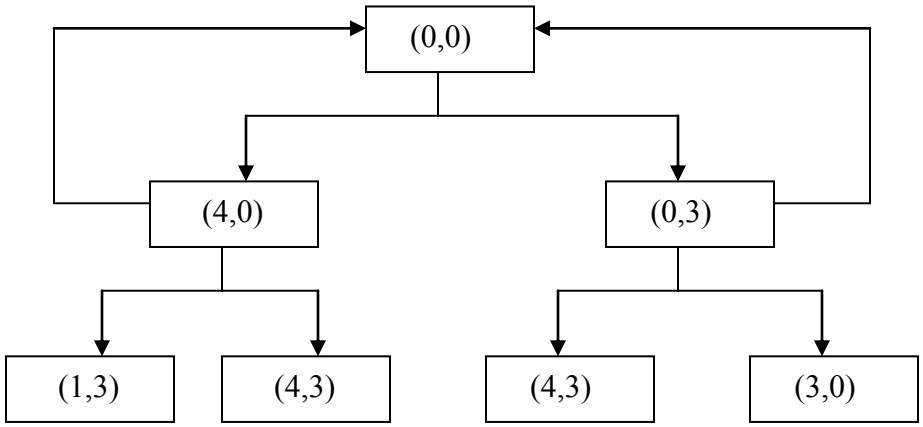


Figure 1.3 Search Graph for the Water-Jug Problem

Heuristic Search Techniques

Weak methods: are varieties of heuristic search techniques whose efficacy is dependent on the way they exploit domain-specific knowledge, since in themselves they are unable to overcome the problems of combinatorial explosion to which search processes are so vulnerable.

General Problem Solving Approaches

There exist quite a large number of problem solving techniques in AI that rely on search. The simplest among them is *the generate and test* method. The algorithm for the generate and test method can be formally stated as follows:

Procedure Generate & Test
Begin
Repeat
Generate a new state and call it current-state;
Until current-state = Goal;
End.

It is clear from the above algorithm that the algorithm continues the possibility of exploring a new state in each iteration of the repeat-until loop and exits only when the current state is equal to the goal. Most important part in the algorithm is to generate a new state. This is not an easy task. If generation of new states is not feasible, the algorithm should be terminated.

In our simple algorithm, we, however, did not include this intentionally to keep it simplified. But how does one generate the states of a problem? To formalize this, we define a four tuple, called state space, denoted by

{ nodes, arc, goal, current },

where

nodes represent the set of existing states in the search space;

an **arc** denotes an operator applied to an existing state to cause transition to another state;
goal denotes the desired state to be identified in the nodes; and
current represents the state, now generated for matching with the goal.

Procedure Hill-Climbing
The 'generate and test' type of search algorithms presented above only expands the search space and examines the existence of the goal in that space. An alternative approach to solve the search problems is to employ a function f(x) that would give an estimate of the measure of distance of the goal from node x. After f(x) is evaluated at the possible initial nodes x, the nodes are sorted in ascending order of their functional values and pushed into a stack in the ascending order of their 'f' values. So, the stack-top element has the least f value. It is now popped out and compared with the goal. If the stack-top element is not the goal, then it is expanded and f is measured for each of its children. They are now sorted according to their ascending order of the functional values and then pushed into the stack. If the stack-top element is the goal, the algorithm exits; otherwise the process is continued until the stack becomes empty. Pushing the sorted nodes into the stack adds a depth first flavor to the present algorithm. The hill climbing algorithm is formally presented below.

The hill climbing algorithm too is not free from shortcomings. One common problem *is trapping at local maxima at a foothill.* When trapped at local maxima, the measure of f at all possible next legal states yield less promising values than the current state. A second drawback of the hill climbing is reaching a *plateau*. Once a state on a plateau is reached, all legal next states will also lie on this surface, making the search ineffective. A new algorithm, called simulated annealing, discussed below could easily solve the first two problems. Besides the above, another problem that too gives us trouble is the *traversal along the ridge.* A ridge on many occasions leads to a local maxima. However, moving along the ridge is not possible by a single step due to non-availability of appropriate operators. A multiple step of movement is required to solve this problem.

- **Algorithm**
  1. Evaluate the initial state. If it is also a goal state, then return and quit. Otherwise continue with the initial state as the current state.
  2. Loop until a solution is found or until there are no new operators to be applied in the currents state.
     (a) Select an operator that has not yet been applied to the current state and apply it to produce a new state.
     (b) Evaluate the new state
        i. If it is a goal state, then return it and quit
        ii. If is not a goal state but it is better than the current state, then make it the current state
        iii. If it is not better then the current state, then continue in the loop
- The Hill climbing is a variant of the generate-and-test search (another search technique based on depth-first search) in which feedback from the test

procedure is used to help the generator decide which direction to move in the search space.

- Used mostly when a good heuristic function is available for evaluating states but no other useful knowledge is available.
- The key difference between this algorithm and the generate-and-test is the use of evaluation function as a way to inject task-specific knowledge into the control process.

Some Phenomena in Heuristic Search

- A *local maximum* is state that is better than all its neighbors but is not better than some other states further away. At a local maximum, all moves appear to make things worse. Local maxima are particularly frustrating because they often occur almost within sight of solution.
- A *plateau* is a flat area of the search space in which a whole set of neighboring states has the same value. On a plateau, it is not possible to determine the best direction in which to move by making local comparisons.
- A *ridge* is a special kind of local maximum. It is an area of the search space that that is higher than surrounding area and that itself has slope. (which one would like to climb). But the orientation of the high region, compared to the set of available moves and the directions in which they move, makes it impossible to traverse a ridge by single moves.

Simulated Annealing Search (Kirkpatrick et al. 1983)

"Annealing" is a process of metal casting, where the metal is first melted at a high temperature beyond its melting point and then is allowed to cool down, until it returns to the solid form. Thus in the physical process of annealing, the hot material gradually loses energy and finally at one point of time reaches a state of minimum energy. A common observation reveals that most physical processes have transitions from higher to lower energy states, but there still remains a small probability that it may cross the valley of energy states and move up to a energy state, higher than the energy state of the valley. The concept can be verified with a rolling ball. For instance, consider a rolling ball that falls from a higher (potential) energy state to a valley and then moves up to a little higher energy state. The probability of such:
transition to a higher energy state, however, is very small and is given by
$p = \exp(-\Delta E / KT)$

where p is the probability of transition from a lower to a higher energy state, ΔE denotes a positive change in energy, K is the Boltzman constant and T is the temperature at the current thermal state. For small ΔE, p is higher than the value of p, for large ΔE. This follows intuitively, as w.r.t the example of ball movement, the probability of transition to a slightly higher state is more than the probability of transition to a very high state.
An obvious question naturally arises: how to realize annealing in search?
Readers, at this stage, would remember that the need for simulated annealing is to identify the direction of search, when the function f yields no better next states than the

current state. Under this circumstance, ΔE is computed for all possible legal next states and p' is also evaluated for each such next state by the following formula:
p' = = exp (-ΔE / T)
A random number in the closed interval of [0,1] is then computed and p' is compared with the value of the random number. If p' is more, then it is selected for the next transition. The parameter T, also called temperature, is gradually decreased in the search program. The logic behind this is that as T decreases, p' too decreases, thereby allowing the algorithm to terminate at a stable state. The algorithm for simulated annealing is formally presented below.

Procedure Simulated Annealing
Begin
1. Identify possible starting states and measure the distance (f) of their closeness with the goal; Push them in a stack according to the ascending order of their f;
2. **Repeat**
Pop stack to get stack-top element;
If the stack-top element is the goal,
announce it and exit;
Else do
Begin
a) generate children of the stack-top element N and
compute f for each of them;
b) If measure of f for at least one child of N is improving
Then push those children into stack in ascending order of
their f;
c) If none of the children of N is better in f
Then do
Begin
a) select any one of them randomly, compute its p' and test whether p' exceeds a randomly generated number in the interval [0,1]; If yes, select that state as the next state; If no, generate
another alternative legal next state and test in this way until one move can be selected; Replace stack-top element by the selected move (state);
b) Reduce T slightly; If the reduced value is negative, set it to zero;
End;
Until the stack is empty;
End.

The algorithm is similar to hill climbing, if there always exists at least one better next state than the state, pointed to by the stack-top. If it fails, then the last begin-end bracketed part of the algorithm is invoked. This part corresponds to simulated annealing. It examines each legal next state one by one, whether the probability of occurrence of the state is higher than the random value in [0,1]. If the answer is yes, the state is selected, else the next possible state is examined. Hopefully, at least one state will be found whose probability of occurrence is larger than the randomly generated probability.

Another important point that we did not include in the algorithm is the process of computation of ΔE. It is computed by taking the difference of the value of f of the next state and that of the current (stack-top) state.
The third point to note is that T should be decreased once a new state with less promising value is selected. T is always kept non-negative. When T becomes zero, p' will be zero and thus the probability of transition to any other state will be zero.

Best- First Search

- Combines the good attributes of both Depth-First Search and Breadth-First Search.
- Depth first search is good because it allows a solution to be found without all computing branches having to be expanded.
- Breadth-first search is good because it does not get trapped on dead-end paths. The Best-first combines these two attributes to follow a single path at a time, but switch paths whenever some competing path looks more promising than the current one does.
- At each step of the best-first search process, we select the most promising of the nodes generated so far. This is done by applying an appropriate heuristic function to each of them. We then expand the chosen node by using the rules to generate its successors. If one of them is a solution, we can quit. If not, all those new nodes are added to a set of nodes generated so far. Again the most promising node is selected and the process continues. Usually what happens is that a bit of depth-first searching occurs as the most promising branch is explored. But eventually, if a solution is not found, that branch will start to look less promising than on e of the top-level branches that had been ignored. At that point, the now more promising, previously ignored branch will be explored. But the old branch is not forgotten. Its last node remains in the set generated but unexpanded nodes. The search return to it when all the others get bad enough that it is again the most promising path.

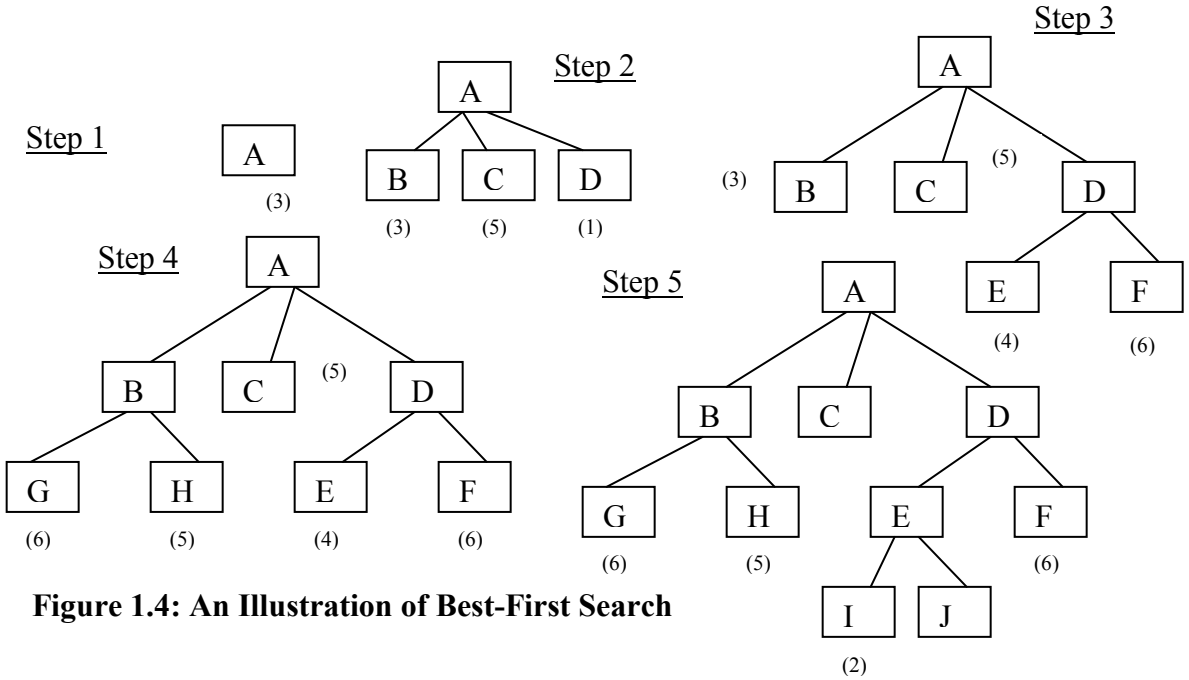


Figure 1.4: An Illustration of Best-First Search

Algorithm: Best –First Search OR Graph

Although best-first search applies to a tree structure, it is better to search a graph so that duplicate path will not be pursued. To implement such a graph-search procedure, we will need to use two list nodes.

- OPEN - nodes that have been generated and have had the heuristic function applied to them but which have not yet been examined (i.e. had their successors generated)
- OPEN is actually a priority queue in which the elements with the highest priority are those with the most promising value of the heuristic function. (Revise: Standard technique for manipulating queues.)
- CLOSED- nodes that have already been examined. We need to keep the nodes in memory if we want to search a graph rather than a tree, since whenever a new node is generated, we need to check whether it has been generated before.

Algorithm

1. Start with OPEN containing just the initial start
2. until a goal is found or there are no nodes left on OPEN do:
    (a) Pick the best node on OPEN
    (b) Generate its successors
    (c) For each successor do:
        i. If it has not been generated before, evaluate it, add it to OPEN, and record its parent.
        ii. If it has been generated before, change the parent if this new path is better than the previous one. In that case, update the cost of getting to this node any successors that this node may already have.