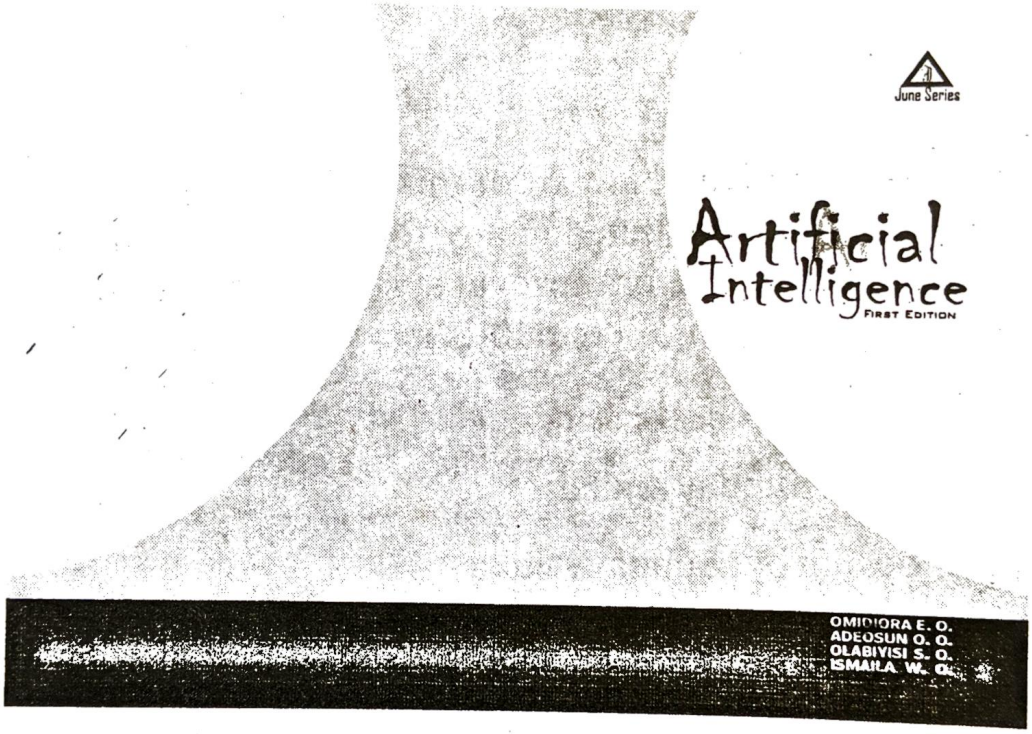
June Series

Artificial Intelligence

First Edition

OMIDIORA E. O.
ADEOSUN O. O.
OLABIYISI S. O.
ISMAILA W. O.



Published in Nigeria by

amstrong

Artificial Intelligence
First edition

ISBN:978 - 38478 - 8 - 0

Settings by: Sunday Ajuka

Cover design by: Sunday Ajuka

Published and printed in Nigeria by
Amstrong Plus Communication,
48, Nitel Rd. Takie, Ogbomoso
08033612510

iii. Production Systems
iv. Natural Language Understanding
v. Planning and Acting
vi. Machine Learning
vii. Robotics
viii. Cognitive modeling.
ix. Neural Networks.

1.1. CLASSIFICATIONS OF A.I.

Human beings specialized in different specific areas though they may be knowledgeable in vast areas. Classification include

i **A1 tools**-Comprises of computer languages, cognitive (act of faculty of knowledge), models, database, management, decision theory, automatic programming.

ii. **Learning and Induction**-Includes heuristics ("tricks" rule of thumb or strategy to solve problems.) reasoning, inference, etc. most techniques for developing A.I. system fall into this strategy.

Iii **Perception and data acquisition**- Comprises of visual and sound recognition.

iv. **Problem solving and model building**- this area includes alternative generation of solutions, goal analysis, searching, decision-making and games.

v **Robotics**-These are machines that imitate human beings. They can recognize objects; manipulate them (gripping, dropping, etc.) and are most of the time mobile.

vi. **Understanding and communication**-This includes knowledge, understanding written and oral language, learning and translation.

1.2. TURNING TEST FOR ARTIFICIAL INTELLIGENCE

The Mathematician, Alan Turing, proposed the only accepted definition of artificial intelligence in 1950. It gets around the problem of our lack of an accepted definition of human intelligence in a very simple way.

The Turing test requires two identical compute terminals in a room. One is connected to a computer in an adjacent room, and the other is connected to another terminal, also in an adjacent room, operated by a person. If anyone

Studies on human thinking and the mind go back nearly 2002 years. Aristotle was one of the first philosophers who attempted to formalize **'right thinking'**. His syllogisms (three-part deductive reasoning) provided patterns for argument structures that always give **'true'** conclusion given **'true'** premises.

A big contribution to A.I. again came from McCarthy in 1958 when he wrote a high level programming language called **'LISP'**. Even today it is one of the most dominant called A.I. programming languages. In the same year, he developed a program called **'Advise Taker'** which was designed to use knowledge to search for solutions to problems. The program was in a published paper titled **'Programs with Common Sense'**. The system used domain knowledge and some simple axioms to generate plans; for example it generated a plan to drive to the airport to catch a plan. The significance of this system in the development of A.I. is that the system embodied knowledge representation and reasoning, and manipulated knowledge representation with deduction.

The early studies on the operations of the mind established the field of logic. Today, the logical approach aim to construct computer programs, with the hope that these programs will be able to create intelligent systems. More specifically in Artificial Intelligent operation must have a learning ability, which is autonomous, goal-directed and highly adaptive.

Artificial intelligence, as described above, demands a number of irreducible features and capabilities. In order to proactively accumulate knowledge form various (and/ or changing) environments, it requires:

i. Sense to obtain features from 'the world' (virtual or actual)
ii. A coherent means for storing knowledge obtained this way, and
iii. Adaptive output/actuation mechanisms (both static and dynamic).

A.I. is the art of creating machine that performs function that requires intelligence when performed by people.

The main research areas in artificial intelligence can be divided into some categories:

i. Game playing
ii. Expert Systems

1. **Learn form experience**: Being able to learn from past situations and events ins a key component of intelligent behavoiur, and is a natural ability for humans, who learn by trial and error. However, learning from experienced is not natural for computer systems. The ability must be carefully programmed into the system. Today researchers developing systems that have this ability. For instance, some A.I. programs, such as computerized chess games, can play human competitors..

2. **Handle Complex and perplexing situations**. Humans are always involved in regarding world difficult, global economic conditions, human and starvation, and natural disasters, in a business setting, top-level mangers and executives are face with a computer mark, difficult and challenging work force. Developing computer system that can handle perplexing situation requires careful planning and elaborate computer programming.

Indeed human intelligence is exhibited in man's actions, responses and choice at his disposals. Therefore when machines are designed to do the same or beyond by setting paths as structure and others then machine can do beyond the reach of man because of the machine's affirmative ability to keep track of events i.e. backward or forward than man. When machine exhibits human intelligence it is know as machine intelligence and machine being man- made thus artificial, machine intelligence is called **artificial intelligence**, while human intelligence is called **natural intelligence.**

A comparison of natural and artificial intelligence

ATTRIBUTES	Natural Intell. (HUMAN)	Artificial Intell. (MACHINE)
i. Acquire a large amount of external information	High	Low
ii. Use Sensors (Eyes, ears etc.)	High	Low
iii. Be creative & imaginative	High	Low
iv. Learn from experience	High	Low
v. Be forgetful	High	Low
vi. Make complex calculation	Low	High
vii. Transfer information	Low	High
viii. The cost of acquiring	Low	High

3. **Apply knowledge acquired from experience**, in addition to learning from experience, people apply what they have learned new settings of

variety of instruments), in many A.I. domains, most of the knowledge a program has must ultimately be provided by people in terms they understand.

- It can easily be modified to correct errors and to reflect changes in the world and in our world view.
- It can be used in a great many situations even if it is not totally accurate or complete.
- It can be used to help overcome its own sheer bulk by helping to narrow the range or possibilities that must usually be considered.

The three important A.I. techniques are:

i. Search:- Provides a way of solving 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.

ii. Use of knowledge:- Provides a way of solving complex problems by exploiting the structures of the objects that are involved.

iii. Abstraction: - Provides a way of separating important features and variations from the many unimportant ones that will otherwise overwhelm any process.

For the solution of hard problems, programs that exploit these techniques have several advantages over those that do not. They are much less fragile; they will not be thrown off completely by a small perturbation in their input; people can easily understanding what the program's knowledge is. And these techniques can work for large problems where more direct methods break down.

1.6. Knowledge-Based System Tools

The most commercial valuable use of A.I. techniques up-to-date tools that can be used for the development of expert or knowledge based systems are discussed below:

i. **Induction systems**. Induction systems constitute a simple way of creating a limited KBS. The induction technology was derived from work aimed at getting systems to learn from example. Specifically, an induction system takes a set of examples and converts it into decision tree.

ii. **Rule-based systems**. Rule-based systems use facts, rules, and a forward or backward chaining inference engine to infer

iii. Knowledge about situations and make recommendations. **Hybrid system**. Hybrid systems add objects, inheritance, and message-passing to rule-based systems. The combination results in much more powerful tools that can store elaborate Problem descriptions in object hierarchies and use rules to make inferences as needed.

1.7. INTELLIGENT AGENTS

Intelligent agent is anything that can be viewed as perceiving it environment to sensors and acting upon that environment through effectors or actuators. Human agents are eyes, ears, nose while legs, hands, mouth and other body part are effectors. Robotic agents have cameras and infrared range finders for the sensor and various motors for the effectors. Software agent (softbots) requires key strokes, file contents, e.t.c. as sensory input and act on environs.

Structure of intelligent agent: The job of A.I. is to design the agent program, that is, a function that implements the agent mapping from percepts to action. The architecture might be a plain computer, or a special-purpose hardware. It might also include software that provides a degree of insulation between raw computer and the agent program. i.e

Agent = Architecture + program

Before we design an agent program, we must have a pretty good ideal of the possible percept and action, what goals or performance measure the agent is supposed to achieve, and what sort of environment of with operate in.

There exist a variety of basic agent-program designs, reflecting the kind of information made explicit and used in the decision process. The four basic kinds of agent program that embody the principles underlying almost all intelligent systems are: **simple reflex agents**- selects actions on the basis of current percepts, ignoring the rest of the percept history; **model-based reflex program**- maintains internal state to track aspects of the world that are not evident in that current percept; **goal-based agents**- as well as a current state description, the agent needs some sort of goal information that describes situations that are desirable and;

utility-based agents- try to maximize their own expected "happiness".

Lion world: this is a cave consisting of rooms connected by passage ways. Lurking some where in the cave is the Lion world, an animal that eats any one who enters its room. The Lion can be slot by an agent, but the agent has only one arrow. Some rooms contain pits that with trap any one who wonder into these rooms (except Lion).

Stench		breeze	pit
lion	breeze Stench gold	pit	breeze
Stench		breeze	
agent	breeze	pit	breeze

Figure 1.1 A typical Lion world. The agents is in the bottom left corner.

Actuators: The agents can move forward, turn left by 90°, or turn right by 90° The agent dies a miserable death if it enters a square containing a pit or a live Lion. The action grab can be used to pick up an object that is in the same square as the agent. The action shoot can be used to five an arrow in a straight line in the direction the agent is facing

Sensors: the agent has five sensor, each of which gives a single beat of information.

(i) in the square containing the Lion and in the directly (not diagonally) adjacent square the agent with perceive a stench.

(ii) in the square directly adjacent to a pit, the agent with perceive a breeze .

(iii) in the square where the gold is, the agent with perceive a glitter.

(iv) When an agent walks into a wall, it will perceive a bump.

(v) When the Lion is killed, it emits a woeful scream that can be perceived any where in the cave.

The percept will be given to the agent in the form of a five symbols, for examples, if there is a stench and a breeze, but no glitter, bump, or scream, the agent will receive the percept (stench, breeze, none, none, none).

CHAPTER TWO

2.0. PROBLEM AND PROBLEM SPACE

The three steps that is required to build a system to solve a particular problem involved:

i. Define the problem precisely: The definition must include précis specifications of what the initial situation will be as well as hat final situations institute acceptable solutions to the problem.

ii. Analyze the problem: A few very important features can have an immense impact on the appropriateness of various possible techniques for solving the problem.

iii. Choose the best techniques and apply it (them) to the particular problem.

Problem formulation is the process of deciding what actions and states to consider given a goal. Thus, in order to provide a formal description of a problem, it is necessary to do the following things;

i. Define a state space that contains all the possible configurations of the relevant objects (and perhaps some impossible ones). It is of course, possible to define this space without implicitly enumerating all of the states of contains.

ii. Specify one or more states within that space that depend possible structure from which the problem solving process may start, these are called the initial states.

iii. Specify one or more states that would be acceptable as solution to the problem. These states are called goal states.

iv. Specify a set act of rules that suitable the actions (operators) available.

The problem can be solved by using the rules in combination with an appropriate control strategy, to move through the problem space until a path from an initial stage to a goal stage is found. Thus the process of

search is fundamental to the problem solving process. The fact that search provides the basis for the process of problem solving does not, however, mean that other more direct approaches cannot also be exploited. Whatever possible, they can be included in steps in the steps by encoding them into the rules.

2.1. WELL DEFINED PROBLEMS AND SOLUTIONS.

A problem can be defined formally by four components.

i. The initial state that the agent starts in: e.g. In (Osun)

ii. A description of the possible solution available to the agent. For example, from the state In(Osun), the successor function for the Nigeria problem would return {[GO(Ondo), In(Ondo)], [GO(Edo), In(Edo)], [GO(Delta), In(Delta)]}.

The initial state and successor function implicitly define the state space of the problem (state space is the set of all states reachable from the initial state). The state space forms a graph in which the nodes are states and the arcs between nodes are actions. The Nigeria Map can be interpreted as a state space graph if we view each road as standing for two driving actions, one in each direction. A path in the state space is a sequence of state connected by a sequence of actions.

iii. The goal test, which determines whether a given state space is a goal state. The agent's goal in Nigeria is the singleton set [In (Anambra)].

iv. A path cost function that assigns a numeric cost to each path. Hence a solution to a problem is a path from the initial state to a goal state. Solution quality is measured by the path cost function and an optimal solution has lowest path cost among all solution.

EXAMPLES

1. **Vacuum World (toy problem).**

   i. **States**: The agent is in one of two locations each of which might or might not contain dirt. So there are 2^3 possible states.

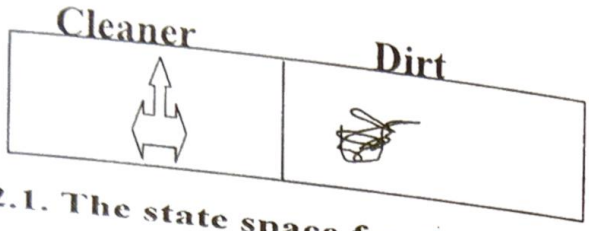


Figure 2.1. The state space for the vacuum world

ii. **Initial state**: Any state can be din stated an the initial state.

iii. **Successor function**: This generated the legal states that result from trying the 3 actions (Left, Right & Suck).

iv. **Goal test**: This chuckles whether all the sequence are churn.

v. **Path cost**: Each step cost 1, so the path cost is the number of steps in the path.

1. The 8-Puzzle.

It consists of a 3x3 board with 8 numbered tiles and a blank space. The standard formulation is as follows:

i. **States**: A state description specifics the location of each of the 8 tiles and the blank in one of the nine sequences.

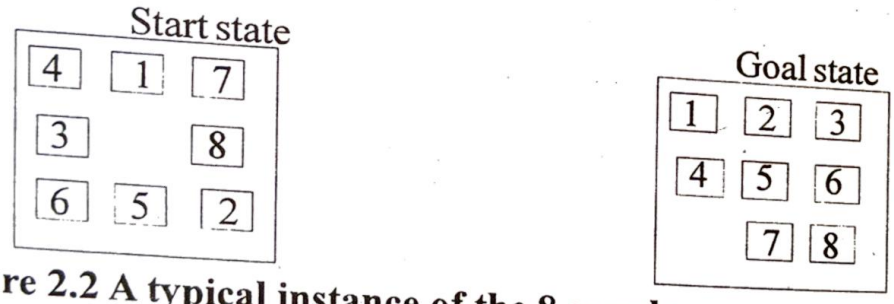


Figure 2.2 A typical instance of the 8-puzzle.

ii. **Initial State**: Any state can be designated as the initial state.

iii. **Successor Function**: this generates the legal status that result from trying the four actions (blankmoves, left, Right, up, or down).

iv. **Goal test**: this checks whether the state matches the goal configurations.

v. **Path cost**: Each step costs 1, so the path cost is the number of steps in the path.

3. Route Finding Problem

Is defined in terms of specified locations and transitions along links between them. Route finding algorithms are used in a variety of applications such as routing in computer networks, military operations planning, and airline travel planning system. Consider an example of an airline travel problem specified as follow:

i **Status**: Each is represented by a location (e.g. an airport) and the current time.

ii. **Initial state**: This is specified by the problem.

iii. **Successor function**: This returns the states resulting from taking any selected flight, learning later then the current time plus the within airport transit time, from the current airport to another.

iv. **Goal test**: Are we at the Destination by some per specified time?

v. **Path cost**: This depends on monetary cost, waiting time, flight time, customs and immigration procedures, seat quality, time of day, type of airplane, frequent flyer mileage awards and so on.

2.1. HOW TO SOLVE A.I. PROBLEMS

Goals help organize behaviors by hinting the objective that the agent is trying to achieve. Goal formation, bases on the correct situation and the agent performance measure, is first in problem solving. Thus, **a goal** is considered to be a set of world state - exactly those states in which the goal is satisfied. The agent tank is to find out which sequence of actions will get it to a goal state.

An agent with several immediate options of unknown value can decide what to do by first examining different possible sequences of actions that lead to states of known value, and than choosing the best sequence. This process of looking for such a sequence is called Search. A search algorithm takes a problem an input and returns a solution in the form of an action sequence. Come a solution is found; the actions of recommends can be carried out. This is called the execution phase.

2.3.PROBLEM CHARACTERISTICS

Heuristic search is a very general method applicable to a large class of problems. It encompasses a variety of specific techniques, each of which is particularly effective for a small class of problem. In order to choose the most appropriate method (or combination of methods) for a particular problem, it is necessary to analyze the problem along several key dimensions.

i. Is the problem decomposable into a set of independent smaller or easier problems?
ii. Can solution steps be ignored or at least undone if they prove unwise?
iii. Is the problem's universe predictable?
iv. Is a good solution to the problem obvious without comparison to all other possible solutions?
v. Is the knowledge based to be used for solving the problem internally consistent?
vi. Is a large amount of knowledge absolutely required to solve the problem or is knowledge important only to constrain search?
vii. Can a computer simply be given the problem and then return with the solution, or will the solution of the problem require interaction between the computer and person?

2.3.1. Is the Problem Decomposable?

Suppose we want to solve the problem of computing the expression

$$\int(X^2 + 3x + \sin^2 x * \cos^2 x)dx$$

We can solve this problem by breaking it down into three smaller problems, each of which we can then solve using a small collection of specific rules. The figure below shows the problem tree that will be generated by the process of problem decomposition as it can be exploited by a simple recursive integration program that works as follows: at each step, it checks to see whether the problem it is working on is immediately solvable. If so, then the answer is returned directly. If the problem is not solvable, the integrater checks to see whether it can decompose the problem into smaller problems. If it can, it creates those problems and calls itself recursively.

Strictly speaking, this is true. But for a great many problems, there is only one (or small number of essentially equivalent) formulation that naturally describes the problem. This was true for each of the problems used as examples above.

When this is the case, it makes sense to view the recoverability of a problem to be equivalent to the recoverability of a natural formulation of it. The recoverability of a problem plays an important role in determining the complexity of the control structure necessary for its solution. Ignorable problems can be solved using a simple control structure that never backtracks. Irrecoverable problems can be solved by a system that expends a great deal of effort making each decision, since the decision must be final.

2.3.2. Is the Universe Predictable?

Suppose that we are playing with the 8-puzzle. Every time we make a move, we know exactly what will happen. This means that it is possible to plan an entire sequence of moves and be confident that we know what the resulting state will be. We can use planning to avoid having to undo actual moves, although it will still be necessary to backtrack past those moves one at a time during the planning process.

One-way of describing planning is that it is problem solving without feedback from the environment. For solving certain-outcome problem, this open approach will work fine since the result of an action can be predicted.

Thus, planning can be used to generate a sequence of operators that is guaranteed to lead to a solution. It is often very expensive since the number of solution paths that need to be explored in creases exponentially with the number of points at which the outcome cannot be predicted.

For uncertain-outcome (playing bridge) problems, however, planning can at best generate a sequence of operators that has a good probability of leading to a solution. To solve such problems, it is also necessary to allow for a process of plan revision to take place as the plan is carried out and the necessary feedback is provided.

2.4. SEARCHING FOR SOLUTION

Having formulated some problems, we now need to solve them. This is done by a search through the state space. The search techniques that use an explicit search tree that is generated by the initial state and the successor

function that together define the state space will be discussed. The root of the search is a search node corresponding to the initial state the initial sate is being expanded to generate new set of states that are further expended (if necessary) to reach the goal state. The choice of which state to expand is determined by the search strategy.

There are many ways to represent nodes, but we will assume that a node is a data structure with five components:

i **State** -The state in the state space to which the node corresponds;

ii **Parent- Node-** The node in the search tree that generated this node

iii **Action-** The action that was applied to the parent to generate the node.

iv. **Path Cost-** The cost, traditionally denoted by g(n), of the path from the initial state to the node, as indicated by the parent pointer;

v. **Depth-** The number of steps along the path from the initial state.

Function TREE-SEARCH (*problem, strategy*) **returns** a solution, or failure / Initialize the search tree using the initial state of *problem*

Loop do

if there are no candidates for expansion **then** **return** failure choose a leaf node for expansion according to *strategy* **if** the node contains a goal state then return the corresponding solution **Else** expand the node and add the resulting nodes To the search tree

Figure 2.4. An informal description of the general tree -search algorithm

Since search forms the core of many intelligent processes, it is useful to structure A.I. programs in a way that facilitates describing the search process. Production systems provide such structures. A definition of a production system is given below

A production system consists of:

i. A set of rules, each consisting of a left side (a pattern) that determines the applicability of the rule, and a right side that

describes the action to be performed if the rule is applied.

ii. One or more databases that contain whatever information is appropriate for the particular task. Some parts of the database may be permanent, while other parts of it may pertain only to the solution of the current problem.

iii. A control strategy that specifies the order in which the rules will be compared to the database and a way of resolving the conflicts that arise when several rules match at once.

2.4. CONTROL STRATEGIES

The first requirement of a good strategy is that it causes motion; the second requirement of a good control strategy is that it must be systematic. However, in search for solution to a problem using search tree, the root of the search trees is a search-node (INITIAL state). The leaf node corresponds to state that do not have successors in the tree.

2.4.1. Breadth-first Search

In this strategy, the root is expanded first, then all the nodes generated by the root node are expanded and their successors and so on. It is a very systematic strategy because it considers all the paths of length one the first, then all those of length two and so on. If there is a solution, BFS is guaranteed to find it, and if there are several solutions, BFS will always find the shallowest goal state first. The memory requirements are enormous. Thus the node that is visited first, will be expanded first in FIFO queue.

Figure 2.5. Breadth-first search on a simple binary tree.

The three major problems with breadth-first search are:

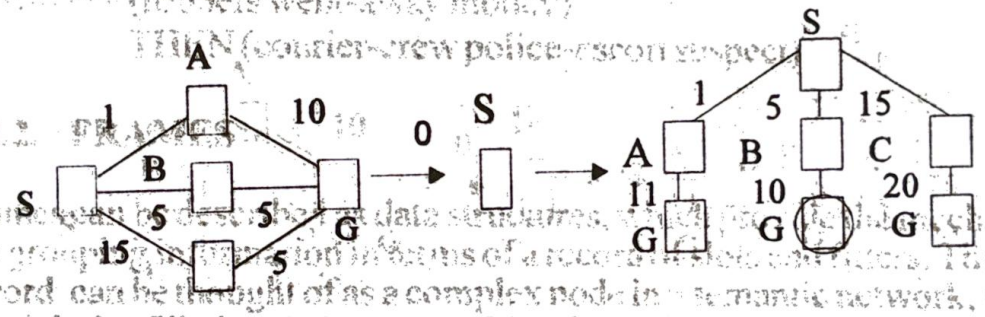
i. It requires a lot of memory. The number of nodes at each level of the tree increases exponentially with the level number, and they must all be stored at once.

ii. It requires a lot of work, particularly if the shortest solution path is quite long, since the number of nodes that need to be examined increases exponentially with the length of the path.

iii. Irrelevant or redundant operators will greatly increase the number of nodes that must be expanded.

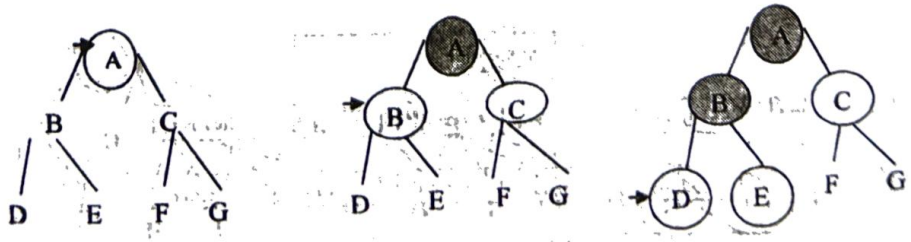
2.4.2. Uniform Cost Search

It modifies the BFS by always expanding the lowest cost node on the fringe rather than the lowest depth node. It is optimal when all step costs are equal, because it always expands the shallowest unexpanded node. When certain conditions are met, the first solution that is found is generated to be the cheapest solution because if there were a cheaper path that was a solution, it would have been expanded earlier and then would have been found first.



2.4.3. Depth First Search

It always expands the cheapest node in the current fringe of the search tree. The search proceeds immediately to the deepest level of he search tree where the nodes have a Successors. As those nodes are expanded, they are dropped from the fringe, so then the search "backs up" to the shallowest node that still has unexplored successors. Only when the search hits a dead end does the search goes back and expanded nodes at shallower levels.



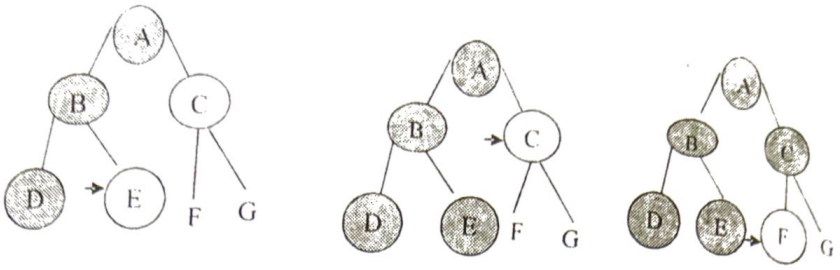


Figure 2.5 Depth-first search on a binary tree. Nodes that have been expanded and have no descendants in the fringe can be removed from the memory; these are shown in black. And f is the only goal node.

2.4.4. Depth Limited Search

It avoids the pitfall of depth first search by imposing a cut off on the maximum depth (l) of a path which can be implemented with a special depth limited search algorithm or by using general search algorithm with operations that keep track of the depth.

Unfortunately, it also introduces an additional source of incompleteness if we choose $l < d$, that is, the shallowest goal is beyond the depth limit. Depth-limited search can be implemented as a simple modification to the general tree-search algorithm or to the recursive depth-first search algorithm. Note that depth-limited search can terminate with two kinds of failure: the standard failure value indicates no solution; the cutoff value indicates no solution within the depth limit.

2.4.5. Iterative Deepening Search

Iterative deepening search is a strategy that sidesteps the issue of choosing the best depth limit by trying all possible depth limits. It is often used in combination with the first search that finds the best depth limit. It does this by gradually increasing the limit -- first 0 then 1, then 2, and so on until a goal is found. The algorithm is shown below. It is optimal and complete like BFS but has only modest memory requirement.

function ITERATIVE-DEEPENING-SEARCH (*problem*) **returns** a solution, or failure

inputs: *problem*, a problem

for *depth* 0 **to** **do**

result DEPTH-LIMITED-SEARCH (*problem. depth*)

if *result* cutoff **then return** *result*

Figure 2.6 The iterative deepening search algorithm, which repeatedly applies depth-limited search with increasing limits.

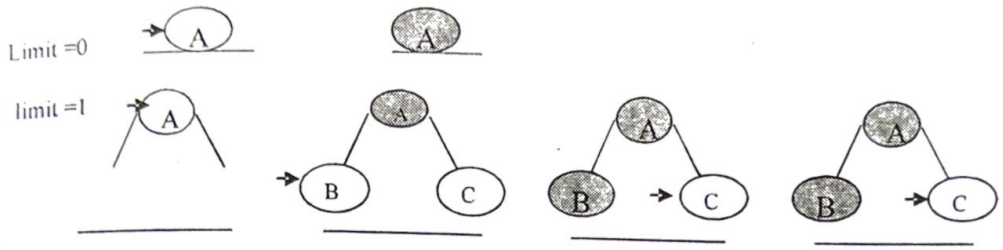


Figure 2.7 Two iterations of iterative deepening search on a binary tree.

Interactive deepening search is analogous to breadth-first search in that it explores a complete layer of new nodes at each interactive before going on to the next layer. The idea is to use necessary path-cost limits instead of increasing depth limits.

2.4.6. Bidirectional Search

The idea behind bi-directional search is to run two simultaneous searchesone forward from the initial state and the other backward from the goal, stopping when the two searches meet each node before it is expanded to see if so, a solution has been found.

Bi-directional search is implemented by having one or both of the searches check each node before it is expanded to see if it is in the fringe of the other search tree; if so, a solution has been found. For example, if a problem has solution depth d=6, and each direction runs breadth-first search one node at a time, then in the worst case the two searches meet when each has expanded all but one of the nodes at depth 3.

The most significant weakness of bi-directional search is the space requirement. The algorithm is complete and optimal (for uniform step cost) if both searches are breadth-first, other combinations may sacrifice completeness, optimal, or both.