UNIT I
INTRODUCTION TO AI AND PRODUCTION SYSTEMS

Introduction to AI-Problem formulation, Problem Definition -Production systems, Control strategies, Search strategies. Problem characteristics, Production system characteristics -Specialized production system- Problem solving methods - Problem graphs, Matching, Indexing and Heuristic functions - Hill Climbing-Depth first and Breath first, Constraints satisfaction - Related algorithms, Measure of performance and analysis of search algorithms.

Introduction to AI

Artificial Intelligence:

Artificial Intelligence is the ability of a computer to act like a human being.

- Artificial intelligence systems consist of people, procedures, hardware, software, data, and knowledge needed to develop computer systems and machines that demonstrate the characteristics of intelligence

Concept of Rationality

A system is rational if it does the "right thing". Given what it knows.

- System that think like human
- System that think rationally
- System that act like human
- System that act rationally

Types of AI Tasks

(i) Mundane Tasks

- Perception
- Vision
- Speech
- Natural Language understanding, generation and translation
- Common-sense Reasoning
- Simple reasoning and logical symbol manipulation
- Robot Control

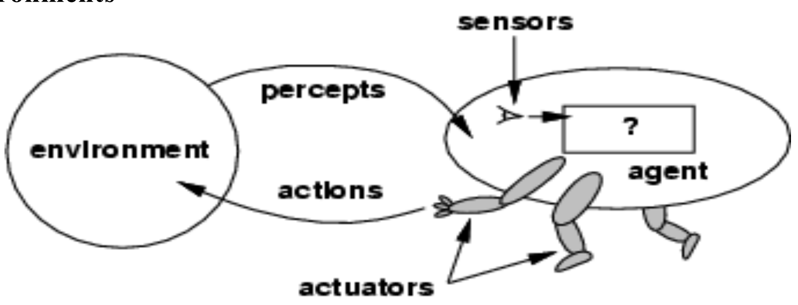
(ii) Formal Tasks

- Games
  - Chess
    Deep Blue recently beat Gary Kasparov
  - Backgammon
  - Draughts
- Mathematics
  - Geometry and Logic
    Logic Theorist: It proved mathematical theorems. It actually proved several theorems from Classical Math Textbooks
  - Integral Calculus
  - Programs such as Mathematical and Mathcad and perform complicated symbolic integration and differentiation.

(iii) Expert Tasks

- Engineering
  - Design
  - Fault finding
  - Manufacturing
- Planning
- Scientific Analysis
- Medical Diagnosis
- Financial Analysis
- Rule based systems - if (conditions) then action

Agents and environments



- An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators
- Human Sensors:
  - eyes, ears, and other organs for sensors.
- Human Actuators:
  - hands, legs, mouth, and other body parts.
- Robotic Sensors:
  - Mic, cameras and infrared range finders for sensors
- Robotic Actuators:
  - Motors, Display, speakers etc

PEAS: Performance, Measure, Environment, Actuators, Sensors
Consider, e.g., the task of designing an automated taxi driver:
- Performance measure - Safe, fast, legal, comfortable trip, maximize profits
- Environment - Roads, other traffic, pedestrians, customers
- Actuators - Steering wheel, accelerator, brake, signal, horn
- Sensors - Cameras, sonar, speedometer, GPS, odometer, engine sensors, keyboard

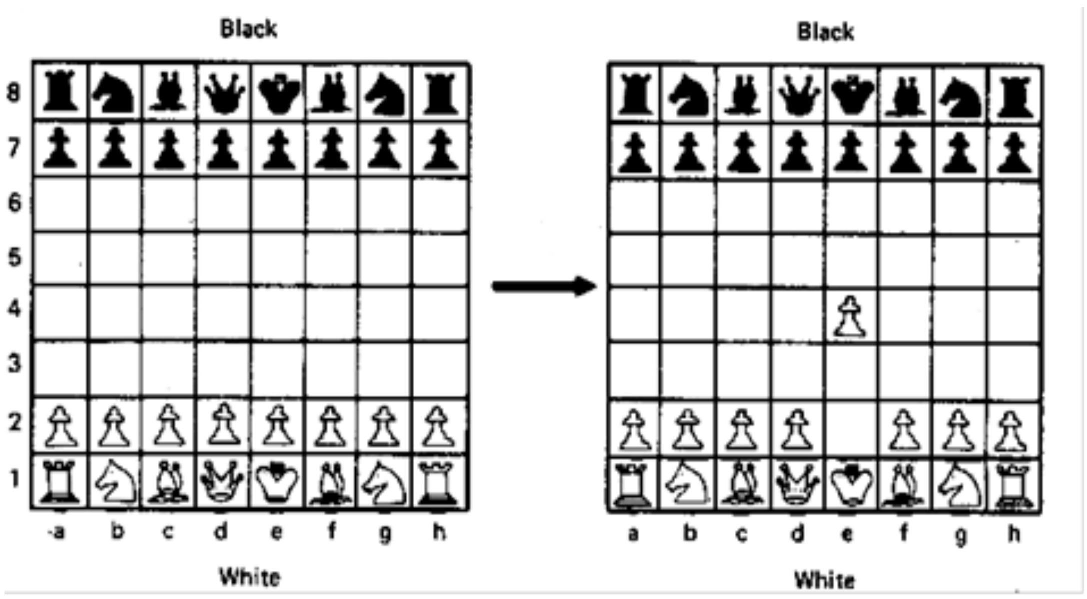
Problem definition

Formal Description of the problem
1. Define a state space that contains all the possible configurations of the relevant objects.
2. Specify one or more states within that space that describe possible situations from which the problem solving process may start (initial state)
3. Specify one or more states that would be acceptable as solutions to the problem. (goal states)
4. Specify a set of rules that describe the actions (operations) available.

Problem Formulation

To build a system to solve a problem
1. Define the problem precisely
2. Analyse the problem
3. Isolate and represent the task knowledge that is necessary to solve the problem
4. Choose the best problem-solving techniques and apply it to the particular problem.

1.Example: Playing Chess
- To build a program that could "play chess", we could first have to specify the starting position of the chess board, the rules that define the legal moves, and the board positions that represent a win for one side or the other.
- In addition, we must make explicit the previously implicit goal of not only playing the legal game of chess but also winning the game, if possible,



white pawn at
square(file e, rank 2)
AND
square(file e, rank 3) is empty
AND
square(file e, rank 4) is empty

Move pawn from
square(file e, rank 2)
to
square(file e, rank 4)

Rules for a single legal move of a white pawn

2.Example: Water Jug Problem

- You are given two jugs, a 4-gallon one and a 3-litre one.
- Neither have any measuring markers 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 4-gallon jug.
- Let x and y be the amounts of water in 4-gallon and 3-gallon Jugs respectively
- (x,y) refers to water available at any time in 4-gallon, 3-gallon jugs.
- (x,y) → (x-d,y+dd) means drop some unknown amount d of water from 4-gallon jug and add dd onto 3-gallon jug.

The state space for this problem can be described as the set of ordered pairs of integers (x,y) such that x = 0, 1,2, 3 or 4 and y = 0,1,2 or 3; x represents the number of gallons of water in the 4-gallon jug and y represents the quantity of water in 3-gallon jug

- The start state is (0,0)
- The goal state is (2,n)

Production rules for Water Jug Problem

Sl No	Current state	Next State	Description
1	(x,y) if x < 4	(4,y)	Fill the 4 gallon jug
2	(x,y) if y <3	(x,3)	Fill the 3 gallon jug
3	(x,y) if x > 0	(x-d, y)	Pour some water out of the 4 gallon jug
4	(x,y) if y > 0	(x, y-d)	Pour some water out of the 3-gallon jug
5	(x,y) if x>0	(0, y)	Empty the 4 gallon jug

6	(x,y) if y >0	(x,0)	Empty the 3 gallon jug on the ground
7	(x,y) if x+y >= 4 and y >0	(4, y-(4-x))	Pour water from the 3 –gallon jug into the 4 –gallon jug until the 4-gallon jug is full
8	(x, y) if x+y >= 3 and x>0	(x-(3-y), 3)	Pour water from the 4-gallon jug into the 3-gallon jug until the 3-gallon jug is full
9	(x, y) if x+y <=4 and y>0	(x+y, 0)	Pour all the water from the 3-gallon jug into the 4-gallon jug
10	(x, y) if x+y <= 3 and x>0	(0, x+y)	Pour all the water from the 4-gallon jug into the 3-gallon jug
11	(0,2)	(2,0)	Pour the 2 gallons from 3-gallon jug into the 4-gallon jug
12	(2,y)	(0,y)	Empty the 2 gallons in the 4-gallon jug on the ground

Gallons in the 4-gallon jug	**Gallons in the 3-gallon jug**	**Rule applied**
0	0	2
0	3	9
3	0	2
3	3	7
4	2	5 or 12
0	2	9 0r 11
2	0	

Required a control structure that loops through a simple cycle in which some rule whose left side matches the current state is chosen, the appropriate change to the state is made as described in the corresponding right side, and the resulting state is checked to see if it corresponds to goal state. One solution to the water jug problem shortest such sequence will have a impact on the choice of appropriate mechanism to guide the search for solution.

Production Systems

A production system consists of:

- A set of rules, each consisting of a left side that determines the applicability of the rule and a right side that describes the operation to be performed if that rule is applied.
- One or more knowledge/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.
- 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.
- A rule applier

To solve a problem:

- We must first reduce it to one for which a precise statement can be given. This can be done by defining the problem's state space (start and goal states) and a set of operators for moving that space.
- The problem can then be solved by searching for a path through the space from an initial state to a goal state.
- The process of solving the problem can usefully be modeled as a production system.

<u>Control Strategies</u>

Control Strategy decides which rule to apply next during the process of searching for a solution to a problem.

- Requirements for a good Control Strategy
  - **It should cause motion**

In water jug problem, if we apply a simple control strategy of starting each time from the top of rule list and choose the first applicable one, then we will never move towards solution.

  - **It should explore the solution space in a systematic manner**

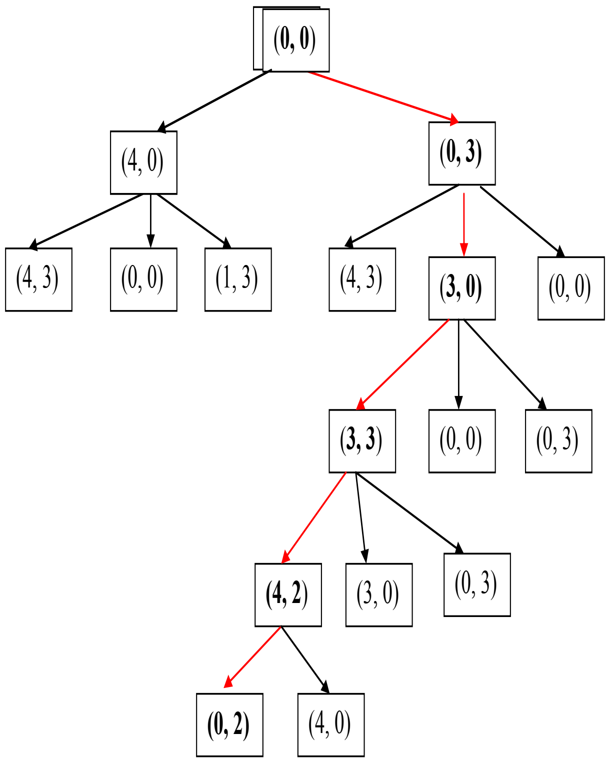
If we choose another control strategy, say, choose a rule randomly from the applicable rules then definitely it causes motion and eventually will lead to a solution. But one may arrive to same state several times. This is because control strategy is not systematic.

<u>Breadth First Search</u>

Algorithm:

1. Create a variable called NODE-LIST and set it to initial state
2. Until a goal state is found or NODE-LIST is empty do
   a. Remove the first element from 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

o



Advantages of BFS

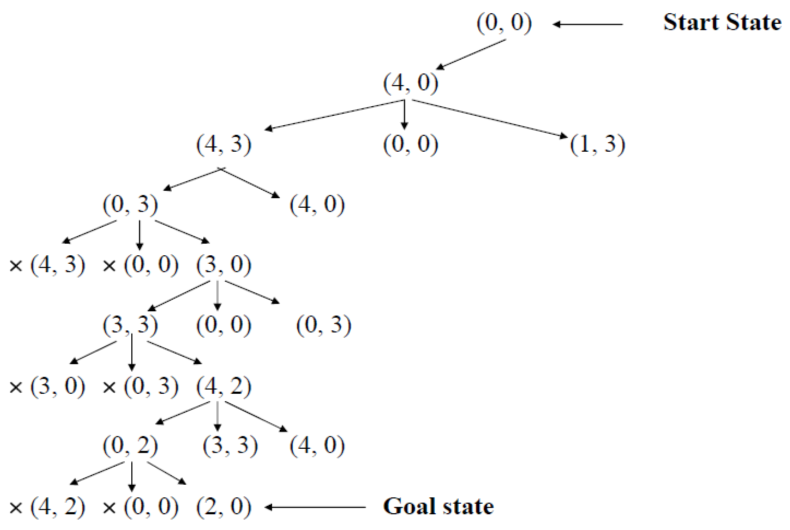
- BFS will not get trapped exploring a blind alley. This contrasts with DFS, which may follow a single unfruitful path for a very long time, perhaps forever, before the path actually terminates in a state that has no successors.
- If there is a solution, BFS is guaranteed to find it.
- If there are multiple solutions, then a minimal solution will be found.

<u>**Depth First Search**</u>

1. If the initial state is a goal state, quit and return success
2. Otherwise, do the following until success or failure is signaled:
   a. Generate a successor, E, of 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 in this loop.

Advantages of Depth-First Search

- DFS requires less memory since only the nodes on the current path are stored.
- By chance, DFS may find a solution without examining much of the search space at all.



Travelling Salesman Problem

A travelling salesman has to visit all the cities atleast once at a minimum cost and return back to the starting position.

- If there are N cities, then number of different paths among them is (N-1)!
- So the total time required to perform this search is proportional to N!
- For 10 cities, 9! = 3,628,800
- This phenomenon is called **combinatorial explosion.**

<u>Problem Characteristics</u>

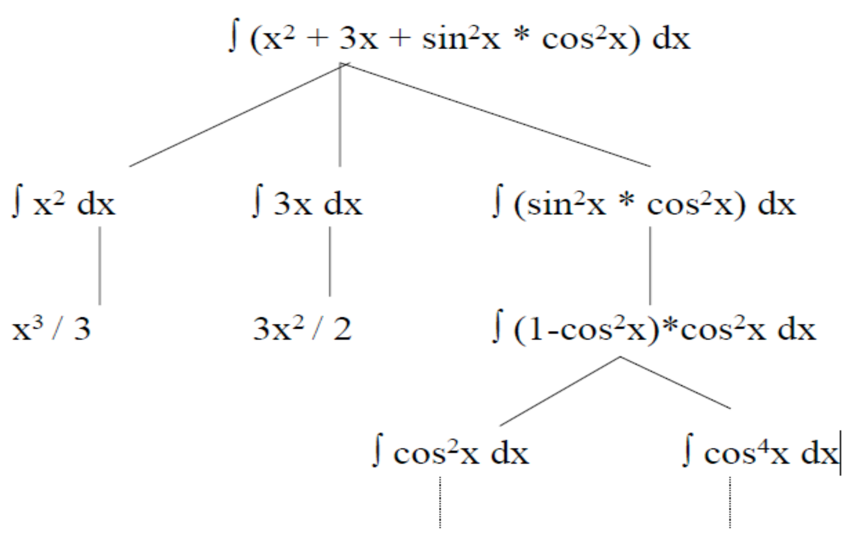
In order to choose the most appropriate method(s) for a particular problem must analyse the problem along several dimensions.

1. Is the problem decomposable into a set of independent smaller sub problems?

Example: Decomposable Problem

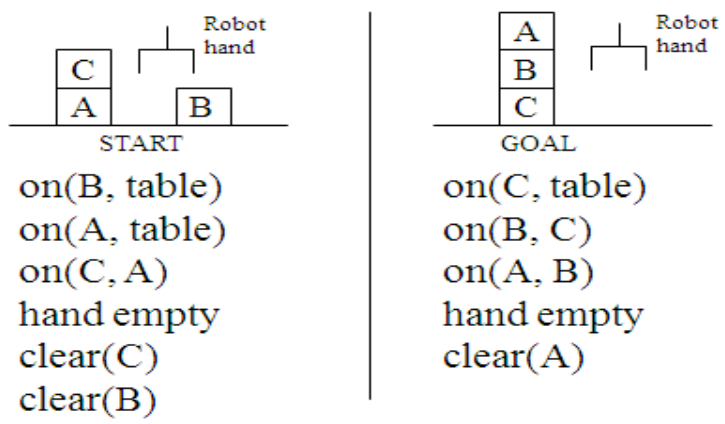
Decomposable problems can be solved by the divide-and-conquer technique. In divide and conquer technique, divide the problem in to sub-problems, find the solutions for the sub-problems. Finally, integrate the solutions for the sub-problems, we will get the solution for the original problem. Suppose we want to solve the problem of computing the integral of the following expression

∫(x2+ 3x + sin2x * cos2x) dx



$\int (x^2 + 3x + \sin^2 x * \cos^2 x)\, dx$
$\int x^2\, dx$
$\int 3x\, dx$
$\int (\sin^2 x * \cos^2 x)\, dx$
$x^3 / 3$
$3x^2 / 2$
$\int (1-\cos^2 x)*\cos^2 x\, dx$
$\int \cos^2 x\, dx$
$\int \cos^4 x\, dx$

Example: Non-Decomposable problem
There are non-decomposable problems. For example, Block world problem is non-decomposable.



on(B, table)
on(A, table)
on(C, A)
hand empty
clear(C)
clear(B)

on(C, table)
on(B, C)
on(A, B)
hand empty
clear(A)

2. Can Solution Steps be ignored or atleast undone if they prove to be unwise?
In real life, there are three important types of problems:
- Ignorable (theorem proving)
- Recoverable (8-puzzle)
- Irrecoverable (Chess)

Ignorable (theorem proving)
Suppose we have proved some lemma in order to prove a theorem and eventually realized that lemma is no help at all, then ignore it and prove another lemma.
- It can be solved by using simple control strategy?

Recoverable (8-puzzle)
Objective of 8 puzzle game is to rearrange a given initial configuration of eight numbered tiles on 3 X 3 board (one place is empty) into a given final configuration (goal state). Rearrangement is done by sliding one of the tiles into empty square.

2	**8**	**3**
1	**6**	**4**
7		**5**

Initial state

1	**2**	**3**
8		**4**
7	**6**	**5**

Goal state

- Solved by backtracking

Irrecoverable (Chess)

- A stupid move cannot be undone.
- Can be solved by planning process.

3. Is the universe predictable?

There are two types of problems. Certain outcome and uncertain outcome.

Certain outcome (8-puzzle problem)

Able to plan the entire sequence of moves.

Uncertain outcome (Bridge game)

It is not possible to plan the entire sequence of actions. We do not know exactly where all the cards are or what the other players will do on their turns. To overcome this, investigate several plans and use probabilities of the various outcomes to choose a plan that has the highest estimated probability of leading to a good score on the hand.

4. Is a good solution absolute or relative?

There are two types of problems

- Any path problem
- Best path problem

Any path problem

- Any path problems can often be solved in a reasonable amount of time by using heuristics that suggest good paths to explore.
- Consider a problem of answering questions based on the database of simple facts.

1. Marcus was a man.
2. Marcus was a Pompeian.
3. Marcus was born in 40AD.
4. All men are mortal.
5. All Pompeian died in 79 AD.
6. No mortal lives longer than 150 years.
7. It is now 1991 AD.

Suppose we ask a question ,**"Is Marcus is alive?".** From the facts given we can easily derive an answer to the question.

1. Marcus was a man.	**1**
4. All men are mortal.	**4**
8. Marcus was a mortal.	**1, 4**
3. Marcus was born in 40AD.	**3**
7. It is now 1991 AD.	**7**
9. Marcus age is 1951 years.	**3, 7**
6. No mortal lives longer than 150 years.	**6**
10. Marcus was dead	**6.8.9**

OR

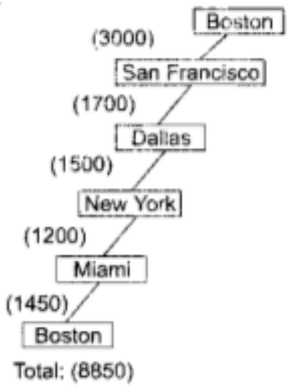
7. It is now 1991 AD.	**7**
5. All Pompeian died in 79 AD.	**5**
11. All Pompeian are dead now.	**7, 5**
2. Marcus was a Pompeian.	**2**
10. Marcus was dead	**11, 2**

Either of two reasoning paths will lead to the answer. We need to find only the answer to the question. No need to go back and see if some other path might also lead to a solution.

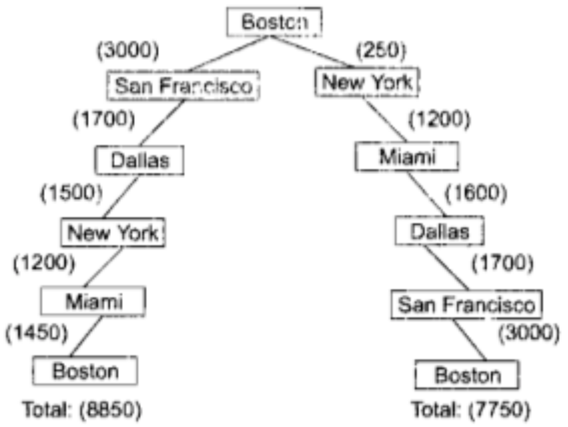
Best path problem

In travelling salesman problem, our goal is to find the shortest route that visits each city exactly once.

	Boston	New York	Miami	Dallas	S.F.
Boston		250	1450	1700	3000
New York	250		1200	1500	2900
Miami	1450	1200		1600	3300
Dallas	1700	1500	1600		1700
S.F.	3000	2900	3300	1700	



This path is not a best solution for the salesman problem.



Best path problems are computationally harder than any path problem.

5. Is the solution a state or path?

State

Finding a consistent interpretation for the sentence "*The bank president ate a dish of pasta salad with the fork*". We need to find the interpretation but not the record of the processing.

Path

In water jug problem, it is not sufficient to report that the problem is solved and the goal is reached, it is (2,0). For this problem, we need the path from the initial state to the goal state.

6. What is the role of knowledge?

For eg, Chess playing

Knowledge about the legal moves of the problem is needed. Without the knowledge, we cannot able to solve the problem.

7. Does the task require Interaction with a person?

Solitary in which the computer is given a problem description and produces an answer with no intermediate communication and no demand for an explanation of the reasoning process.

Conversational in which there is intermediate communication between a person and the computer, either to provide additional assistance to the computer or to provide additional information to the user, or both.

Production Characteristics

1. Can production systems, like problems, be described by a set of characteristics that shed some light on how they can easily be implemented?
2. If so, what relationships are there between problem types and the types of production systems best suited to solving the problems?

- Classes of Production systems:

- **Monotonic Production System:** the application of a rule never prevents the later application of another rule that could also have been applied at the time the first rule was selected.
- **Non-Monotonic Production system**:
- **Partially commutative Production system**: property that if application of a particular sequence of rules transforms state x to state y, then permutation of those rules allowable, also transforms state x into state y.
- **Commutative Production system:** A Commutative production system is a production system that is both monotonic and partially commutative.

	Monotonic	NonMonotonic
Partially Commutative	Theorem proving	Robot Navigation
Not Partially Commutative	Chemical Synthesis	Bridge

Partially commutative non-monotonic Production System
Example: Robot Navigation

The Robot has the following operators: go north (N), go east (E), go south (S), and go west (W). To reach its goal, it does not matter whether the robot executes N-N-E or N-E-N.

Non-partially commutative monotonic Production System
Example: Chemical synthesis

"Add chemical x to pot" OR "Change temperature to t degrees"

These operators may cause irreversible changes to the potion being brewed. The order in which they are performed can be very important in determining the final output.

<u>Specialized production system</u>

There are two basic types of production System:

- Commutative Production System
- Decomposable Production System

(i) Commutative Production System

A production system is commutative if it has the following properties with respect to a database D:

1. Each member of the set of rules applicable to D is also applicable to any database produced by applying an applicable rule to D.
2. If the goal condition is satisfied by D, then it is also satisfied by any database produced by applying any applicable rule to D.
3. The database that results by applying to D any sequence composed of rules that are applicable to D is invariant under permutations of the sequence.

Commutativity of a system does not mean that the entire sequence of rules used to transform a given database in to one satisfying a certain condition can be reordered. After a rule is applied to a database, additional rules become applicable. Only those rules that are initially applicable to a data base can be organized in to an arbitrary sequence and applied to that database to produce a result independent of order.

Commutative production systems are an important subclass enjoying special properties. For example, an irrecoverable control regime can always be used in a commutative system because the application of a rule never needs to be taken back or undone. Any rule that was applicable to an earlier database is still applicable to the current one.

(ii) Decomposable Production System

Initial database can be decomposed or split into separate components that can be processed independently.

Consider, for example, a system whose initial database is (C,B,Z), whose production rules are based on the following rewrite rules,

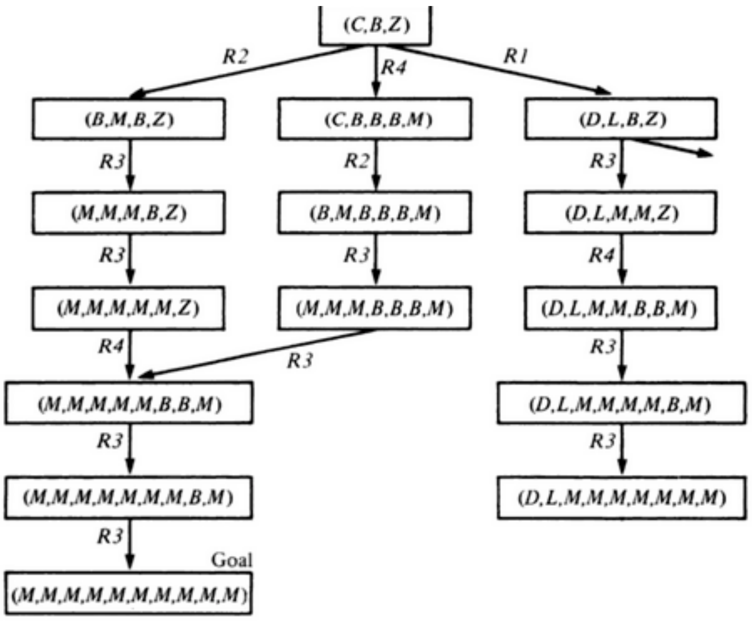
R1: C→ (D,L)

R2: C→ (B,M)

R3: B→ (M,M)

R4: Z→ (B,B,M)

And whose termination condition is that the data base contain only Ms.



A graph search control regime might explore many equivalent paths in producing a database containing only Ms. Redundant paths can lead to inefficiencies because the control strategy might attempt to explore all of them. One way to avoid exploration of these redundant paths is to recognize that the initial database can be decomposed or split in to separate components that can be processed independently.

In our example, the initial database can be split in to the components C,B and Z. production rules can be applied to each of these components independently; the results of this application can also be split and so on, until each components database contains only Ms.

Search Algorithms

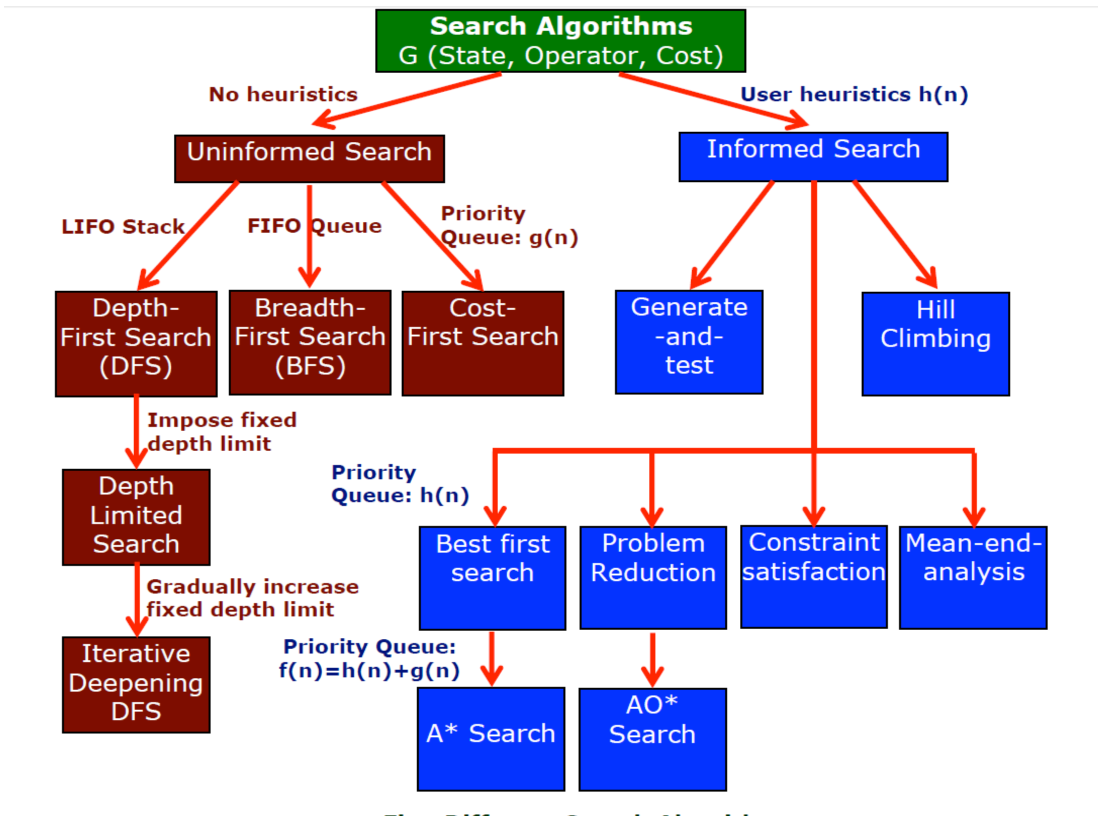


Fig. Different Search Algorithms

- **Uninformed or blind search** strategies uses only the information available in the problem definition
- **Informed or heuristic search** strategies use additional information. Heuristic tells us approximately how far the state is from the goal state. Heuristics might underestimate or overestimate the merit of a state.

Generate and test

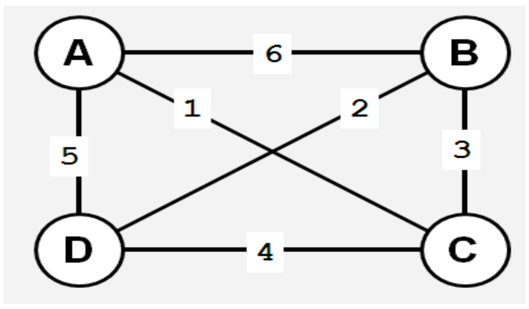
The generate and test is the simplest form of all heuristic search methods

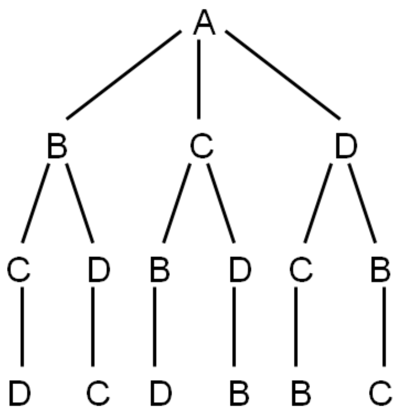
Algorithm

1. Generate a possible solution. For some problems, this means generating a particular point in the problem space. For others, it means generating a path from a start state.
2. Test to see if this is actually a solution by comparing the chosen point or the endpoint of the chosen path to the set of acceptable goal states.
3. If a solution has been found, quit. Otherwise, return to step 1.

Example - Traveling Salesman Problem (TSP)

- Traveler needs to visit *n* cities.
- Know the distance between each pair of cities.
- Want to know the shortest route that visits all the cities once.





- TSP - generation of possible solutions is done in lexicographical order of cities:
  1. A - B - C - D
  2. A - B - D - C
  3. A - C - B - D
  4. A - C - D - B
  5. A - D - C - B
  6. A - D - B - C
- *n=80* will take millions of years to solve exhaustively!

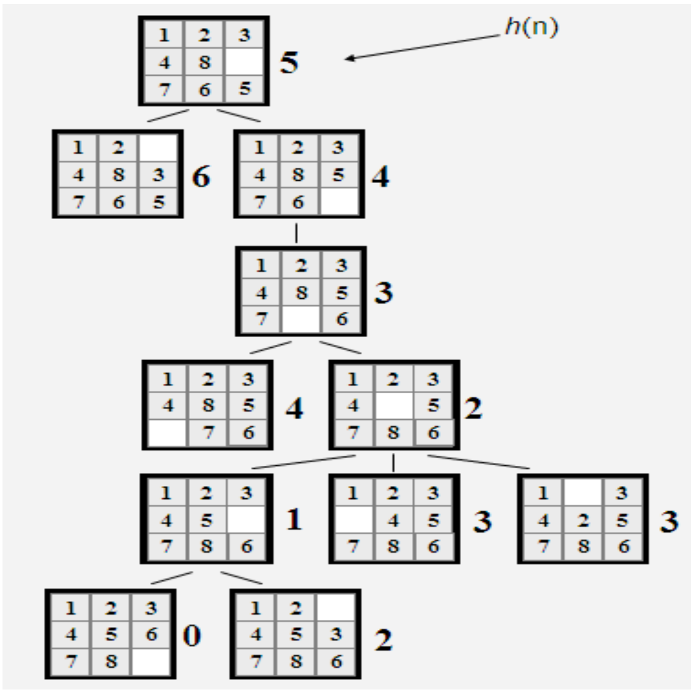
<u>Hill Climbing</u>

Algorithm

1. Evaluate the initial state.
2. Loop until a solution is found or there are no new operators left to be applied:
   - Select and apply a new operator
   - Evaluate the new state:
     - goal → quit
     - better than current state → new current state

Example: 8 puzzle problem

Here, h(n) = the number of **misplaced tiles** (not including the blank), the Manhattan Distance heuristic helps us quickly find a solution to the 8-puzzle.



Advantages of Hill Climbing

- Estimates how far away the goal is.
- Is neither optimal nor complete.
- Can be very fast.

Steepest-Ascent Hill Climbing (Gradient Search)

The steepest ascent hill climbing technique considers all the moves from the current state. It selects the best one as the next state.

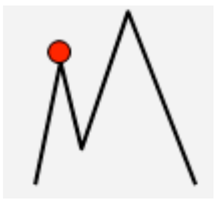
Algorithm

```
1. Evaluate the initial state.
2. Loop until a solution is found or a complete iteration produces no change to current state:
       - SUCC = a state such that any possible successor of the
         current state will be better than SUCC (the worst state).
       - For each operator that applies to the current state, evaluate
         the new state:
              goal → quit
              better than SUCC → set SUCC to this state
       - SUCC is better than the current state → set the current state to SUCC.
```

Disadvantages

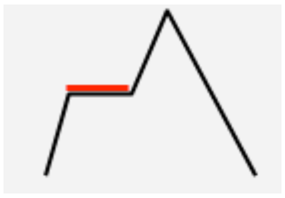
- **Local maximum**

A state that is better than all of its neighbours, but not better than some other states far away.



- **Plateau**

A flat area of the search space in which all neighbouring states have the same value.



- **Ridge**

The orientation of the high region, compared to the set of available moves, makes it impossible to climb up. However, two moves executed serially may increase the height.

There are some ways of dealing with these problems.

- Backtrack to some earlier node and try going in a different direction. This is a good way of dealing with local maxima.
- Make a big jump to try to get in a new section of the search space. This is particularly a good way of dealing with plateaus.
- Apply two or more rules before doing a test. This corresponds to moving in several directions at once. This is particularly a good strategy for dealing with ridges.

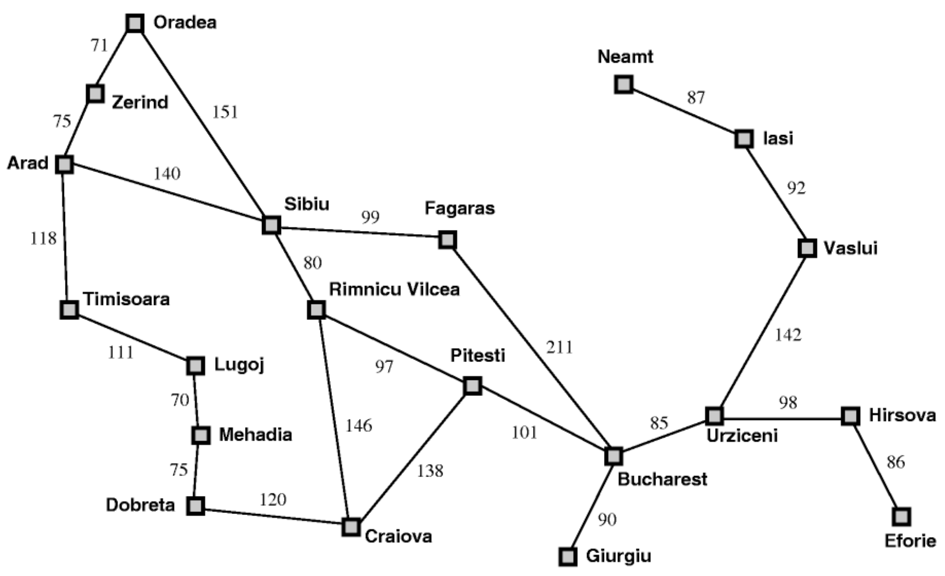
Best-First Search

Best first search combines the advantages of Breadth-First and Depth-First searches.

- DFS: follows a single path, don't need to generate all competing paths.
- BFS: doesn't get caught in loops or dead-end-paths.

• Best First Search: explore the most promising path seen so far. Nodes are ordered and expanded using evaluation function. The best evaluation function is expanded first.

• Two types of evaluation function
  - **Greedy Best First Search**
  - **A* search**



Straight−line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

(i) Greedy Best First Search

Greedy best first search minimize the estimated cost to reach the goal. It always expand the node that appears to be closed to the goal.

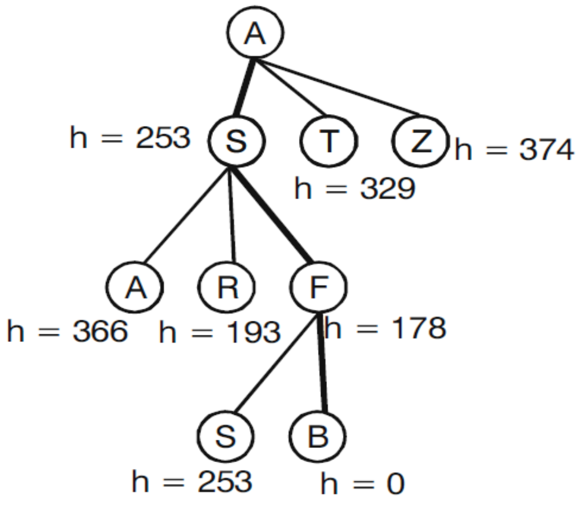
Evaluation function

f(n)=h(n)

• *h(n) = estimated cost of the cheapest path from the state at node n to a goal state*

Algorithm

```
1. Start with OPEN containing just the initial state.
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 and to any
        successors that this node may already have.
```



A-S-F-B→ 140+99+211=450

Performance Analysis

- Time and space complexity – O(b^m)
- Optimality – no
- Completeness - no

<u>(ii) A* search Algorithm</u>

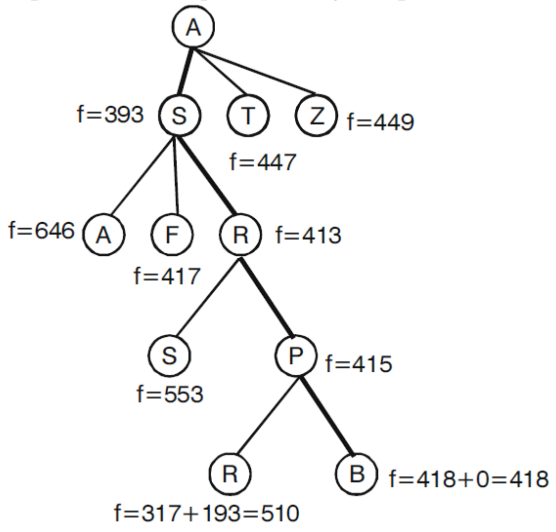
Evaluation function

f(n) = h(n)+g(n)

- f(n) = cost of the cheapest solution through n
- g(n) = actual path cost from the start node to node n

Algorithm

1. Create a priority queue of search nodes (initially the start state). Priority is determined by the function f)
2. While queue not empty and goal not found:
   - (a) Get best state x from the queue.
   - (b) If x is not goal state:
   - (i) generate all possible children of x (and save path information with each node).
   - (ii) Apply f to each new node and add to queue.
   - (iii) Remove duplicates from queue (using f to pick the best).



A-S-R-P-B→ 140+80+97+101=418

Performance Analysis

- Time complexity – depends on heuristic function and admissible heuristic value
- space complexity – O(b^m)
- Optimality – yes (locally finite graphs)
- Completeness – yes (locally finite graphs)

<u>Constraint Satisfaction Problem (CSP) – search</u>

- Search procedure that operates in a space of constraint sets.
- Initial state - contains the constraints that are originally given in the problem description.
- Goal State - any state that has been constrained "enough," where "enough" must be defined for each problem
- Example: Crypt arithmetic, graph coloring

CSP – Example - Crypt arithmetic

Statement: the aim is to find a substitution of digits for letters such that the resulting sum is arithmetically correct, each letter stand for a different digit.

- Given : FORTY+ TEN+ TEN=SIXTY
- 29786 + 850+ 850 = 31486
- F=2, O=9, R=7, T=8, Y=6, E=5, N=0

Constraint satisfaction is a two-step process

- Constraints are discovered and propagated as far as possible throughout the system. Then, if there is still not a solution, search begins.
- A guess about something is made and added as a new constraint. Propagation can then occur with this new constraint, and so forth

Algorithm

1. Propagate available constraints. To do this, first set OPEN to the set of all objects that must have values assigned to them in complete solution. Then do until an inconsistency is detected or until OPEN is empty:

   (a) Select an object OB from OPEN. Strengthen as much as possible the set of constraints that apply to OB.

   (b) If this set is different from the set that was assigned the last time OB was examined or if this is the first time OB has been examined, then add to OPEN all objects that share any constraints with OB.

   (c) Remove OB from OPEN.

2. If the union of the constraints discovered above defines a solution , then quit and report the solution.

3. If the union of the constraints discovered above defines a contradiction, then return failure.

4. If neither of the above occurs, then is necessary to make a guess at something inorder to proceed. To do this, loop until a solution is found or all possible solutions have been eliminated.

   (a) Select an object whose value is not yet determined and select a way of strengthening the constraints on the object.

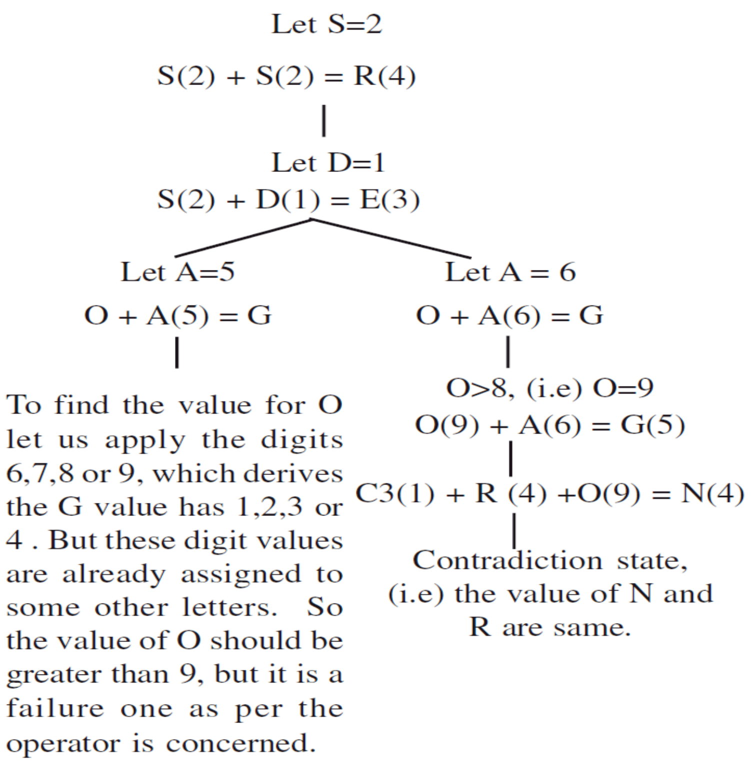
   (b) Recursively invoke constraint satisfaction with the current set of constraints augmented by the strengthening constraint just selected.

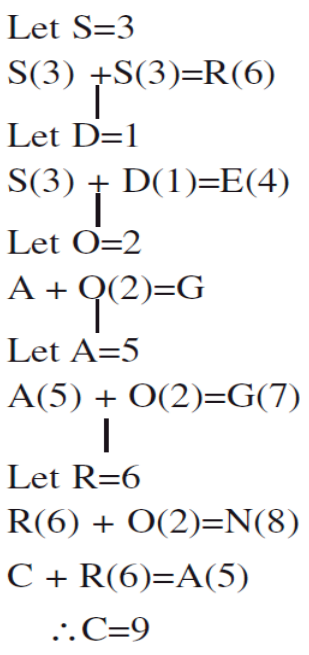
Example

Given : CROSS + ROADS = DANGER

Initial state : C R O S A D N G E C1 C2 C3 C4 = ?

Goal state: The digits to the letter should be assigned in such a manner that the sum is satisfied.


Let S=2
S(2) + S(2) = R(4)
Let D=1
S(2) + D(1) = E(3)
Let A=5
O + A(5) = G
To find the value for O let us apply the digits 6,7,8 or 9, which derives the G value has 1,2,3 or 4 . But these digit values are already assigned to some other letters. So the value of O should be greater than 9, but it is a failure one as per the operator is concerned.
Let A = 6
O + A(6) = G
O>8, (i.e) O=9
O(9) + A(6) = G(5)
C3(1) + R (4) +O(9) = N(4)
Contradiction state, (i.e) the value of N and R are same.



	C4(0)	C3(0)	C2(0)	C1(0)	
	C(9)	R(6)	O(2)	S(3)	S(3)
	R(6)	O(2)	A(5)	D(1)	S(3)
D(1)	A(5)	N(8)	G(7)	E(4)	R(6)

<u>MEANS-ENDS ANALYSIS</u>

So far we have seen many search strategies that can move either in the forward direction or backward direction. But, means end analysis allows both backward and forward searching
Search process reduces the difference between the current state and the goal state until the required goal is achieved

- Solve major parts of a problem first and then return to smaller problems when assembling the final solution – operator sub-goaling
- Example :
  - GPS was the first AI program to exploit means-ends analysis.
  - STRIPS (A robot Planner)

Procedure

1. Until the goal is reached or no more process are available:
   (a) Describe the current state, the goal state and the differences between the two.
   (b) Use the difference between the current state and goal state, possibly with the description of the current state or goal state, select a promising procedure.
   (c) Use the promising procedure and update current state.
2. If goal is reached then **success** otherwise **failure**.

Household robot domain

- Problem: Move desk with two things on it from one location S to another G. Find a sequence of actions robot performs to complete the given task.
- Operators are: PUSH, CARRY, WALK, PICKUP, PUTDOWN and PLACE given with preconditions and results.

S **B________C** **G**

Start **PUSH** **Goal**

S(Start)→ walk(start_desk_loc) → pickup(obj1) → putdown(obj1) → pickup(obj2)→ putdown(obj2) →push(desk, goal_loc) → walk(start_desk_loc) → pickup(obj1) → carry(obj1,goal_loc) → place(obj1,desk) → walk(start_desk_loc) → pickup(obj2) → carry(obj2,goal_loc) → place(obj2,desk) →G(Goal).

Algorithm

1. Compare CURRENT and GOAL. If there are no differences between them then return.
2Otherwise, select the most important difference and reduce it by doing the following until success or failure is signaled.

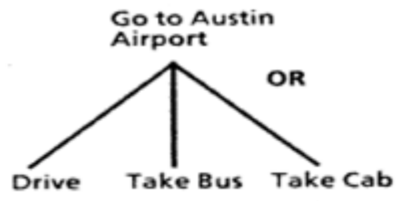
(a) Select an as yet untried operator O that is applicable to the current difference. If there are no such operators, then signal failure.

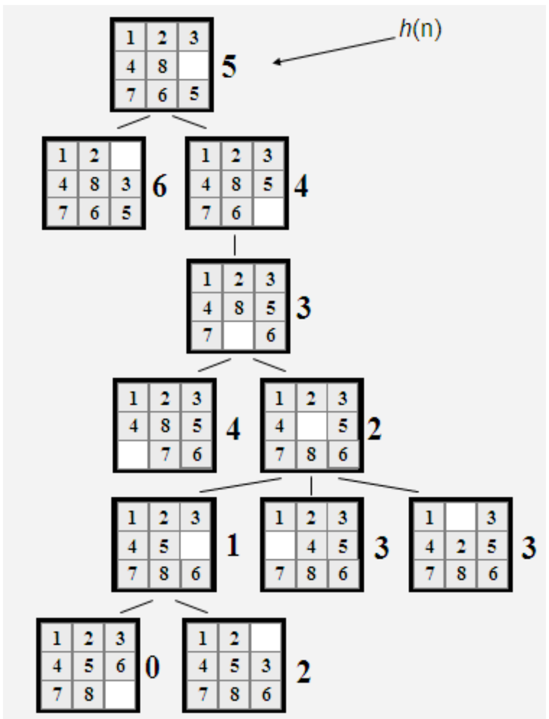
(b) Attempt to apply O to CURRENT. Generate descriptions of two states:
O-START- a state in which O's preconditions are specified.
O-RESULT- the state that would result if O were applied in O-START.

(c) If
(FIRST-PART←MEA(CURRENT, O-START))
and
(LAST-PART←MEA(O-RESULT, GOAL))
are successful, then signal success and return the result of concatenating FIRST-PART, O and LAST-PART.

OR graph

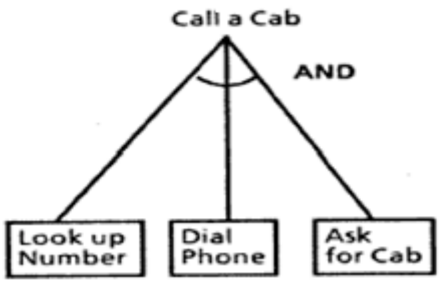
An OR graph consists entirely of OR nodes, and in order to solve the problem represented by it, you only need to solve the problem represented by one of his children
(Eight Puzzle Tree example).

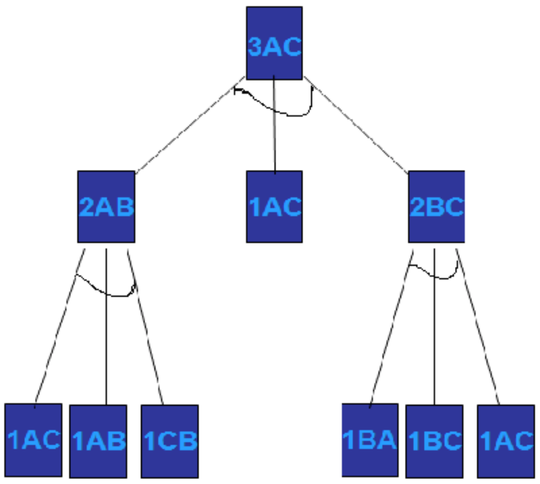




AND GRAPH

An AND graph consists entirely of AND nodes, and in order to solve a problem represented by it, you need to solve the problems represented by all of his children *(Hanoi towers example)*.

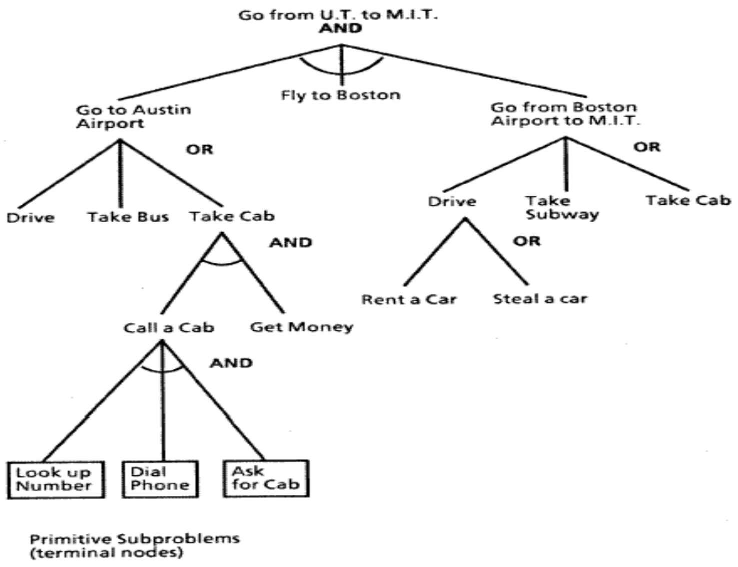




AND/OR Graphs

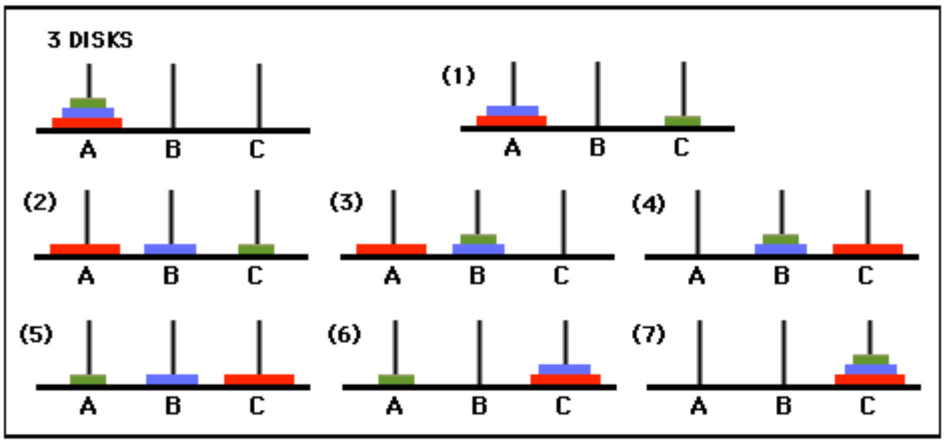
AND-OR graph is useful for certain problems where

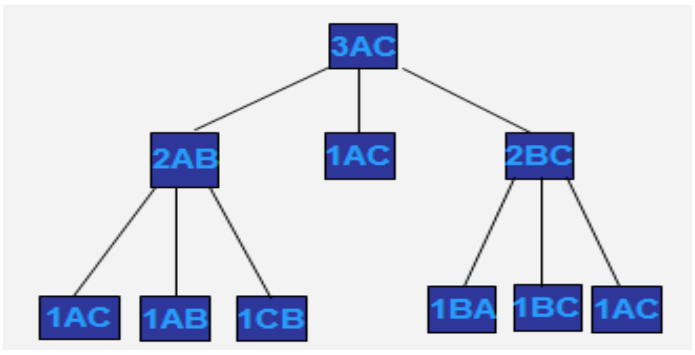
- The solution involves decomposing the problem into smaller problems. We then solve these smaller problems
- An AND/OR graph consists of both AND nodes and OR nodes.



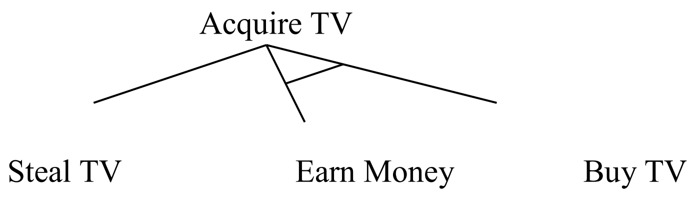
Problem Reduction

- Each sub-problem is solved and final solution is obtained by combining solutions of each sub-problem.
- Decomposition generates arcs that we will call AND arc.





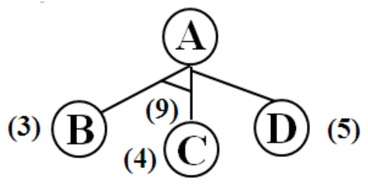
- One AND arc may point to any number of successors, all of which must be solved.
- Such structure is called AND–OR graph rather than simply AND graph.

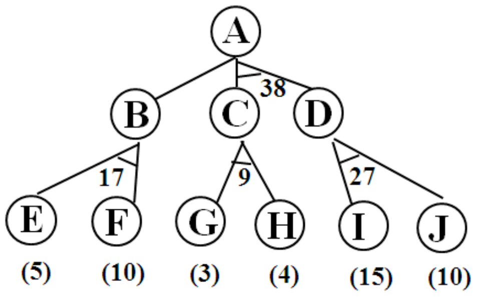


To find a solution in AND–OR graph, we need an algorithm similar to A* with the ability to handle AND arc appropriately.
In search for AND-OR graph, we will also use the value of heuristic function **f** for each node.

i) AND–OR Graph Search:

- Traverse AND-OR graph, starting from the initial node and follow the current best path.
- Accumulate the set of nodes that are on the best path which have not yet been expanded.
- Pick up one of these unexpanded nodes and expand it.
  Add its successors to the graph and compute f (using only h) for each of them.
- Change the f estimate of newly expanded node to reflect the new information provided by its successors.
  Propagate this change backward through the graph to the start.
- Mark the best path which could be different from the current best path.
- Propagation of revised cost in AND-OR graph was not there in A*.





Algorithm: Production reduction

1. Initialize the graph to the starting node.
2. Loop until the starting node is labeled SOLVED or until its cost goes above FUTILITY:

   a) Traverse the graph, starting at the initial node and following the current best path and accumulate the set of nodes that are on the path and have not yet been expanded or labeled as solved.

   b) Pick one of the unexpanded nodes and expand it. If there are no successors, assign FUTILITY as the value of this node. Otherwise, add its successors to the graph and for each of them complete f' (use only h' and ignore g) .if any node is 0,mark that node as SOLVED.

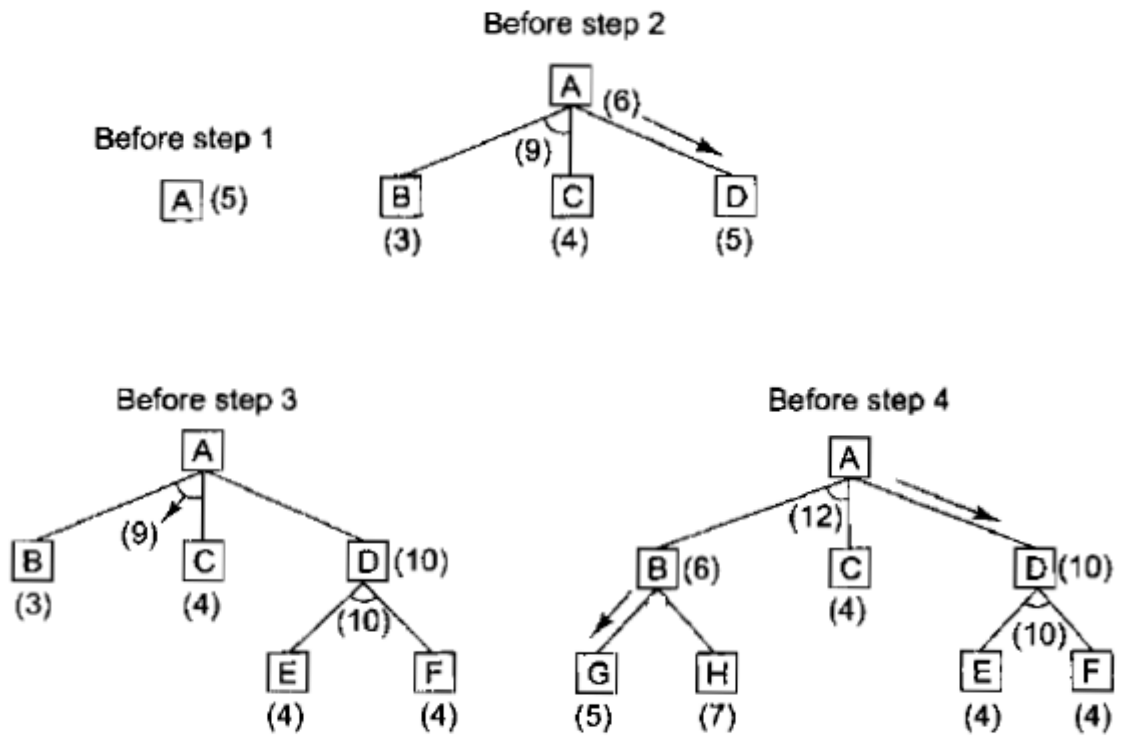
   c) Change the f' estimate of the newly expanded need to reflect the new information provided by its successors. Propagate this change backward through the graph. If any node contains the successor are whose descendants are all solved, label the node itself as SOLVED. At each that us visited while going up the graph, decide which of its successor arcs is the most promising and mark it as part of the current best path. This may cause the current best path to change. This prorogation of revised cost estimates back up the tree was not necessary in the best first search algorithm.

ii) AO* algorithm

AO* algorithm uses a single structure GRAPH, representing the part of the search graph that has been explicitly generated so far. Each node in the graph will point both down to its immediate successors and up to its immediate predecessors.

1. Let G be a graph with only starting node INIT.
2. Repeat the followings until INIT is labelled SOLVED or h(INIT) > FUTILITY
   a) Select an unexpanded node from the most promising path from INIT (call it NODE)
   b) Generate successors of NODE. If there are none, set h(NODE) = FUTILITY (i.e., NODE is unsolvable); otherwise for each SUCCESSOR that is not an ancestor of NODE do the following:
      i. Add SUCCESSSOR to G.
      ii. If SUCCESSOR is a terminal node, label it SOLVED and set h(SUCCESSOR) = 0.
      iii. If SUCCESSPR is not a terminal node, compute its h.
   c) Propagate the newly discovered information up the graph by doing the following: let S be set of SOLVED nodes or nodes whose h values have been changed and need to have values propagated back to their parents. Initialize S to Node. Until S is empty repeat the followings:
      i. Remove a node from S and call it CURRENT.
      ii. Compute the cost of each of the arcs emerging from CURRENT. Assign minimum cost of its successors as its h.
      iii. Mark the best path out of CURRENT by marking the arc that had the minimum cost in step ii
      iv. Mark CURRENT as SOLVED if all of the nodes connected to it through new labelled arc have been labelled SOLVED

v. If CURRENT has been labelled SOLVED or its cost was just changed, propagate its new cost back up through the graph. So add all of the ancestors of CURRENT to S.



The operation of problem reduction