CS464 ARTIFICIAL INTELLIGENCE

MODULE 1

SYLLABUS

- Introduction: What is AI
- The foundations of AI
- History and applications
- Production systems.
- Structures and strategies for state space search.
- Informed and Uninformed searches.

Aritificial Intelligence

Artificial intellegence(AI) is the study of how to make computer do things which at the moment people do better

The AI problems

- Mundane Tasks
  - Perception
    - Vision
    - Speech
  - Natural Language
    - Understanding
    - Generation
    - Translation
  - Commonsense reasoning
  - Robot control
- Formal Taks
  - Games
  - Chess
  - Backgammon
  - Checkers
- Mathematics
  - Geometry
  - Logic
  - Integral calculus
  - Proving properties of programs
- Expert Tasks
  - Engineering
  - Design
  - Fault Finding
  - Manufacturing planning
- Scientific Analysis
- Medical Diagnosis
- Financial Analysis

Cont..

- AI is a system that acts like human beings
  - Natural language processing
    - To enable it to communicate successfully in English.
  - Knowledge representation
    - To store what it knows or hears.
  - Automated reasoning
    - To use the stored information to answer questions and to draw new conclusions.
  - Machine learning
    - To adapt to new circumstances and to detect and extrapolate patterns.
  - Computer vision:To perceive objects.
  - Robotics: To manipulate objects and move about.

Cont..

- AI is a system that thinks like human beings
- AI is a system that thinks rationally
  - For a given set of correct premises, it is possible to yield new conclusions.
- AI is a system that acts rationally

Cont..

- 2 most fundamental concerns of AI researchers
- Knowledge representation
  - It addresses the problem of capturing the full range of knowledge required for intelligent behavior in a formal language,
  - i.e. One suitable for computer manipulation.
  - Eg. predicate calculus, LISP, Prolog
- Search
  - It is a problem solving technique that systematically explores a space of problem states, ie, successive and alternative stages in the problem solving process.

AI Application Areas

- Game Playing
- Heuristics
- Automated Reasoning and Theorem Proving
- Expert Systems
- Natural Language Understanding and Semantic Modeling
- Modeling Human Performance

Languages and Environments for AI

- Programming environments include
  - knowledge structuring techniques such as
  - object oriented programming and
  - expert systems frameworks.
- High level languages such as
  - Lisp, and
  - Prolog support modular development.

Problems, Problem Space And Search

- To build a system to solve a particular problem
  - Define the problem precisely:
    - precise specification of what the initial situation will be as well as what final situation constitute acceptable solution to the problem
  - Analyse the problem
    - a few very important feature can have an immense impact on appropriateness of various possible technique for solving the problem
  - Isolate and represent the task knowledge
    - that is necessary to solve the problem
  - Choose the best problem solving technique and apply it to particular problem.

Defining problem as a state space search

- Build a program to PLAY CHESS

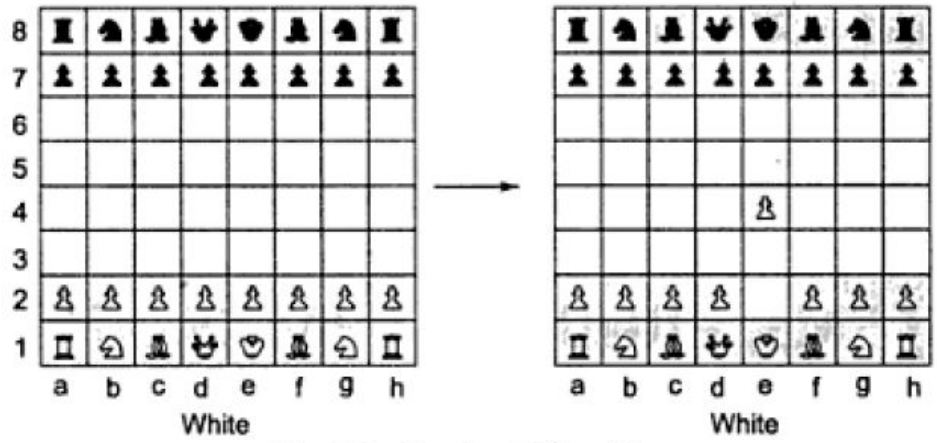


Fig. 2.1 *One Legal Chess Move*

Cont..

- To build a program
  - Specify the starting position of the chess board
  - The rules that define the legal moves
  - The board positions that represent a win for one side or other
- Set of rules consisting of two parts:
  - A left side that serves as a pattern to be matched against the current board position
  - A right side that describe the change to be made to the board position to reflect the move

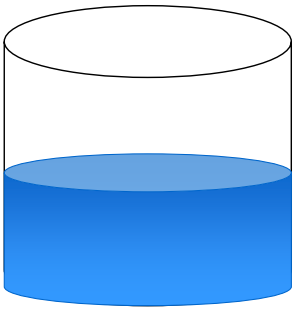
Cont..

White pawn at
Square(file e, rank 2)
AND
Square(file e, rank 3) → move pawn from Square(file e, rank 2) to Square(file e, rank 4)
is empty
AND
Square(file e, rank 4)
is empty

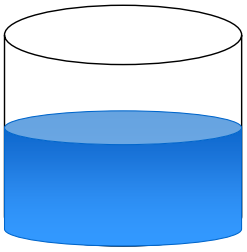
Fig. 2.2 *Another Way to Describe Chess Moves*

- state space representation forms the basics of most of the AI methods Its Structure is:
  - It allows for a formal definition of a problem as the need to convert some given situation into some desired situation using a set of permissible operations
  - It permits us to define the process of solving a particular problem as a combination of known techniques and
  - search the general technique of exploring the space to try to find some path from the current state to goal state.

Water Jug Problem



4 gallon Jug



3 gallon jug

no measuring markers

Cont..

You are given two jugs, a 4-gallon one and a 3 gallonone. Neither has any measuring markers on it.There is a pump that can be used to fill the jugs with water.

How can you get exactly 2gallon of water into 4 gallon jar?

Cont..

- state space
  - set of ordered pair of integers(x,y) such that
  - x=0,1,2,3 or 4 and y=0,1,2 or 3.
  - x represent number of gallon of water in the 4 gallon jug
  - y represent number of gallon of water in the 3 gallon jug
- Start state is (0,0).
- Goal state is (2,n) n can be any value.

- Production rules

1	(x,y) -----> (4,y)If x<4	Fill the 4 gallon jug
2	(x,y) -----> (x,3)If y<3	Fill the 3 gallon jug
3	(x,y) -----> (x-d,y)If x>0	Pour some water out of 4 gallon jug
4	(x,y) -----> (x,y-d)If y>0	Pour some water out of 3 gallon jug
5	(x,y) -----> (0,y)If x>0	Empty the 4 gallon jug on ground
6	(x,y) -----> (x,0)If y>0	Empty the 3 gallon jug on ground

Cont..

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

Solution

Gallon in 4 gallon Jug	Gallon in 3 gallon Jug	Rule Applied

Cont..

- Production System
  - A set of rules each contsisting of a
    - left side that determines the applicability of the rule and
    - right side that describes the operation to be performed if rule is applied.
  - One or more knowledge/database that contain whatever information is appropriate for the particular task.
  - A control stratergy 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.

Problem Charateristics

- Is the problem Decomposable?
- Can solution steps be ignored or undone?
- Is the universe predictable?
- Is a good solution Absolute or relative
- Is the solution a state or path?
- What is the Role of Knowledge
- Does the task require interaction with a person?

Is the problem Decomposable?

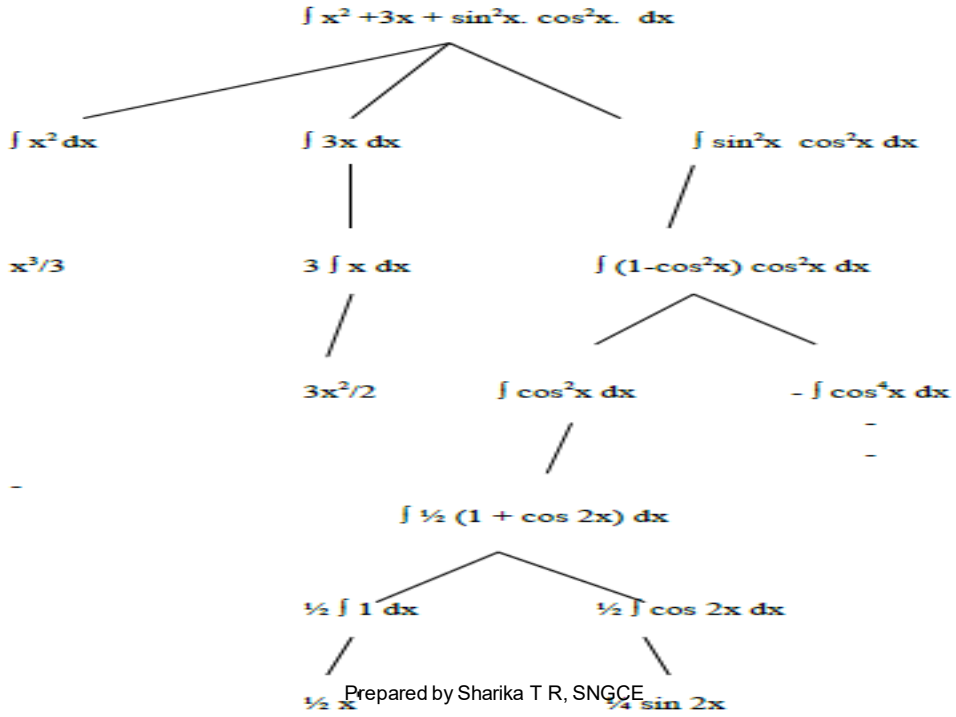
- Decomposable problems
- Non Decomposable problems

1. Decomposable problems

- can solve this problem by breaking it down into three smaller problems

$$\int (x2 + 3x + \sin 2x . \cos 2x) dx$$

- each of which we can then solve by using a small collection of specific rules.
- problem decomposition



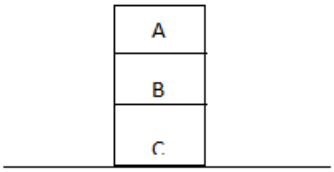
2. Non Decomposable problems

- A simple blocks world problem

Start:

ON (C,A)

Goal:



ON(B,C) and ON(A,B)

Assume that the following operators are availale

1. CLEAR(x) ⟶ ON(x,table)
   [block x has nothing on it] [pick up x and put it on the table]
2. CLEAR(x) and CLEAR(y) ⟶ ON(x,y)

Cont..

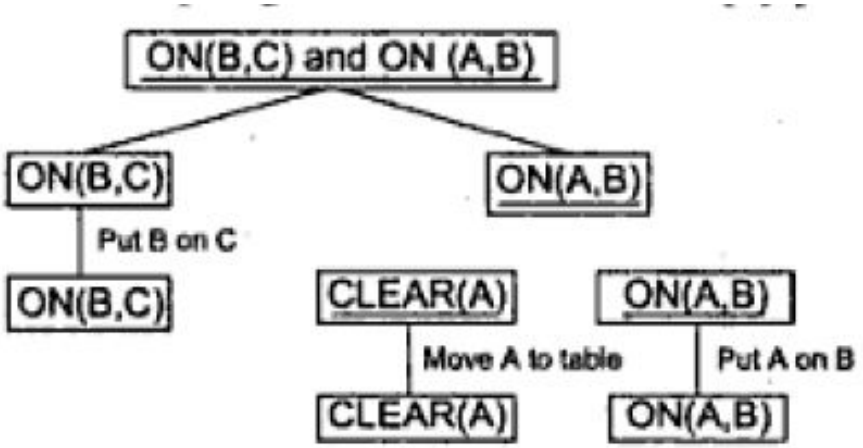


Fig. 2.11 *A Proposed Solution for the Blocks Problem*

Cont..

- Regardless of which one we do first we will not be able to do the second as we had planned.
- In this problem the two sub problems are not independent.
- They interact and those interactions must be considered in order to arrive at a solution for entire problem.

Can solution steps be ignored or undone?

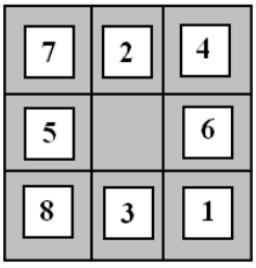
- Here we can divide problems into 3 classes.
  - Ignorable, in which solution steps can be ignored.
  - Recoverable, in which solution steps can be undone.
  - Irrecoverable, in which solution steps cannot be undone.

Ignorable Problem

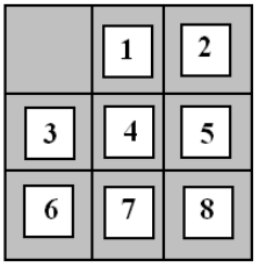
- eg, Theorm Proving
- Suppose we are trying to prove a mathematical theorem.
- We proceed by first proving a lemma that we think will be useful.
- Eventually we realize that the lemma is no help at all.
- Here the different steps in proving the theorem can be ignored.
- Then we can start from another rule.
- The former can be ignored.

Recoverable Problems

- eg, 8 Puzzle
- 8-puzzle solver can keep track of the order in which operations are performed so that the operations can be undone one at a time if necessary.



Start state



Goal state

1	2	5
3	4	
6	7	8

Irrecoverable problems

- eg. Chess
- Suppose a chess playing program makes a stupid move and realizes it a couple of moves later.
- It cannot simply play as though it had never made the stupid move.
- Nor can it simply back up and start the game over from that point.
- All it can do is to try to make the best of the current situation and go from there.

Cont..

- Ignorable problems can be solved using a simple control structure.
- Recoverable problems can be solved by a slightly more complicated control strategy that does sometimes makes mistakes.
- Irrecoverable problems will need to be solved by a system that expends a great deal of effort making each decision since the decision must be final.

Is the universe predictable?

- Certain outcome problems
- Uncertain outcome problems

Certain outcome problems

- 8 puzzle problem.
- 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.

Uncertain outcome problems

- Bridge
  - planning may not be possible.
  - One of the decisions we will have to make is which card to play on the first trick.
  - it is not possible to do such planning with certainty since we cannot know exactly where all the cards are or what the other players will do on their turns.

4.Is a good solution Absolute or relative

- Any Path Problem
- Best Path Problem

Any path problems

- Is a good solution Absolute or relative
- Any path problems
  - 1. Marcus was a man.
  - 2. Marcus was a Pompean.
  - 3. Marcus was born in 40 A. D.
  - 4. all men are mortal.
  - 5. All pompeans died when the volcano erupted in 79 A. D.
  - 6. No mortal lives longer than 150 years.
  - 7. It is now 1991 A. D.
- Suppose we ask the question, "Is Marcus alive?".

	Solutions	Axiom
1	Marcus was a man.	1
4	All men are mortal.	4
3	Marcus was born in 40 A.D.	3
7	It is now 2017 A. D.	7
9	Marcus' age is 1977 years.	3,7
6	no mortal lives longer than 150 years.	6
10	Marcus is dead.	8,6,9

Best path problems

- Traveling salesman problem
- Best path problems are computationally harder than any path problems.
- Any path problem can often be solved in a reasonable amount of time by using heuristics that suggest good path to explore.

5.Is the solution a state or path?

- Problems whose solution is a state of the world. eg. Natural language understanding. eg,

  ‘ The bank president ate a dish of pasta salad with the fork’.

  – Since all we are interested in is the answer to the question, it does not matter which path we follow.
- Problems whose solution is a path to a state?
  – Eg. Water jug problem
  – In water jug problem, it is not sufficient to report that we have solved the problem and that the final state is (2,0).
  – For this kind of problem, what we really must report is not the final state, but the path that we found to that state.

6. What is the Role of Knowledge

- Problems for which a lot of knowledge is important only to constrain the search for a solution.
  - Eg. Chess
  - Just the rules for determining the legal moves and some simple control mechanism that implements an appropriate search procedure
- Problems for which a lot of knowledge is required even to be able to recognize a solution.
  - Eg. News paper story understanding

7. Does the task require interaction with a person?

- Solitary problems
  - Here the computer is given a problem description and produces an answer with no intermediate communication and with no demand for an explanation for the reasoning process.
  - Consider the problem of proving mathematical theorems. If
    - All we want is to know that there is a proof.
    - The program is capable of finding a proof by itself.
  - Then it does not matter what strategy the program takes to find the proof.

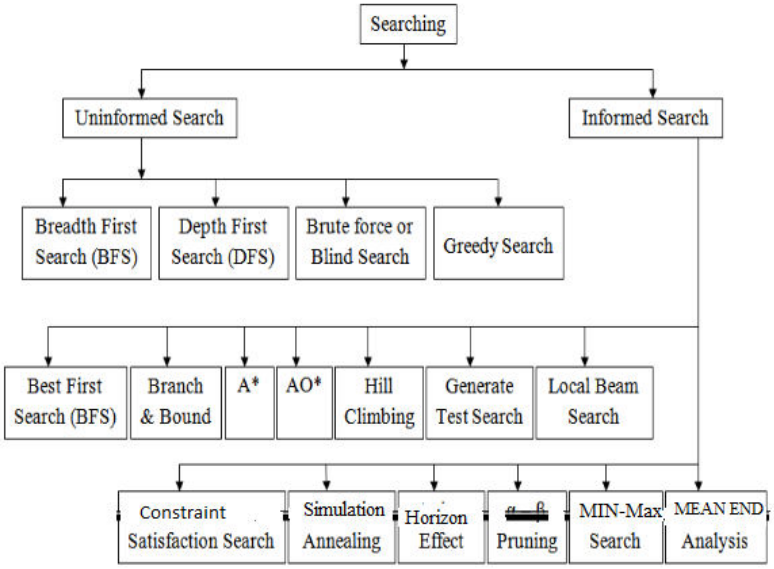
Cont..

- Conversational problems
  - 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.
    - Eg. Suppose we are trying to prove some new, very difficult theorem.
    - Then the program may not know where to start.
    - At the moment, people are still better at doing the high level strategy required for a proof.
    - So the computer might like to be able to ask for advice.
    - To exploit such advice, the computer's reasoning must be analogous to that of its human advisor, at least on a few levels.

SEARCHING

- search algorithm takes a problem as input and returns the solution in the form of an **action sequence**.
- Once the solution is found the execution phase starts.
- After formulating a goal and problem to solve the agent cells a search procedure to solve it.
- A problem can be defined by 5 components.
  a) The initial state: The state from which agent will start.
  b) The goal state: The state to be finally reached.
  c) The current state: The state at which the agent is present after starting from the initial state.
  d) Successor function: It is the description of possible actions and their outcomes.
  e) Path cost: It is a function that assigns a numeric cost to each path.

DIFFERENT TYPES OF SEARCHING



Uninformed Search Strategies

- A problem determines the graph and the goal but not which path to select from the frontier.
- This is the job of a search strategy.
- A search strategy specifies which paths are selected from the frontier.
- Different strategies are obtained by modifying how the selection of paths in the frontier is implemented.
- they that do not take into account the location of the goal. These algorithms ignore where they are going until they find a goal and report success.
  - Depth-First Search
  - Breadth-First Search

Informed Search

- A search using domain-specific knowledge.
- Suppose that we have a way to estimate how close a state is to the goal, with an evaluation function.
- General strategy:
  - expand the best state in the open list first.
  - It's called a best-first search or ordered state-space search.
- In general the evaluation function is imprecise, which makes the method a heuristic (works well in most cases).
- The evaluation is often based on empirical observations.

Control Stratergies

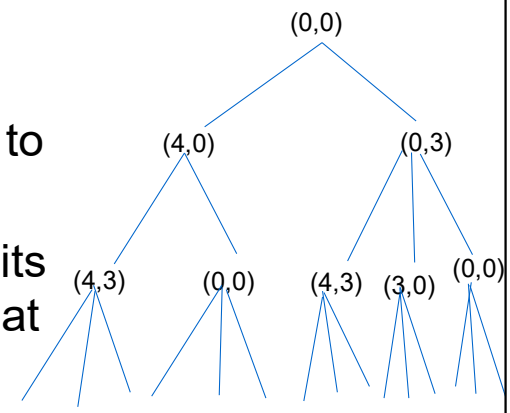
- Requirments
  - cause motion
  - it must be systematic
- Control stratergies that do not cause motion will never lead to a solution.
- The requirement that a control stratergy be systemtic corresponds to the need for global motion as well as for local motion.

UNINFORMED SEARCH

BFS & DFS, Generate and test, Plan Generate and test

Breath First Search

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



Breadth first Search Algorithm

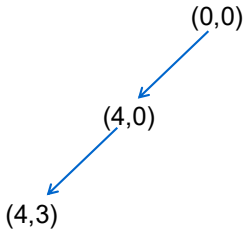
1. Create a variable called NODE_LIST and set it to initial state
2. Until a goal state is found or NODE_LIST is empty
   a) Remove the first element from NODE_LIST and call it E. if NODE_LIST was empty quit
   b) For each way that each rule can match the state described in E do
      i. Apply the rule 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 end of NODE_LIST

Depth first search

- Pursue a single branch of tree until a solution or until a decision to determine the path is made.
- It makes sense to terminate a path if it reaches a dead end. In such a case, backtracking occurs.
- The most recently created state from which alternative moves are available will be revisited and a new state will be created.
- This form of backtracking is called **chronological backtracking.**

DFS Algorithm

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 the initial state. If there are no more successors,signal failure
   b) Call Depth first search with E as the initial state
   c) If the success is returned, signal success otherwise continue in the loop.



Advantages of BFS

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

Generate-and-Test 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 choosen point or the endpoint of the choosen path to the set of acceptable goal states
3. If a solution has been found, quit. Otherwise return to step 1.

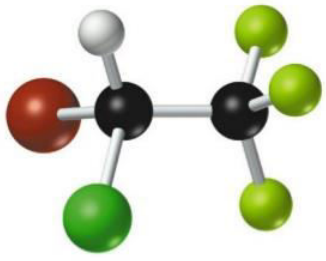
Cont..

- If the generation of possible solution is done systematically, then this procedure will find a solution eventually if one exists.
- If the problem space is very large eventually may be a very long time.
- It is a depth first search procedure since complete solutions must be generated before they can be tested.
- It can also operate by generating solutions randomly but then there is no guarentee that a solution will ever be found.
- It is also known as British Museum Algorithm

Cont..

Plan-Generate-Test

- Dendral uses plan-generate test stratergy in which a planing process that uses constraint-satisfaction techniques creates list of recommended and contraindicated substructures.
- The generate-and-test procedure then uses those lists so that it can explore only a fairly limited set of structures.
- Constrained in this way generate-and-test procedure has proved highly effective.



DENDRAL:
Used to identify the structure of chemical compounds. First used in 1965

Cont..

- A major weakness of planning is that it often produces somewhat inaccurate solutions since there is no feedback from the world.
- But by using it only produce pieces of solution that will then be exploited in the generate-and-test process, the lack of detailed accuracy become unimportant.

Other Uninformed Search Algorithms

- Depth-Limited Search Algorithm
- Uniform-cost Search Algorithm
- Iterative deepening depth-first Search
- Bidirectional Search Algorithm

Informed Search Technique

Travelling Salesman Problem Hill Climbing, Simulated Annealing

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 round trip that both starts and finishes at any one of the cities.

Cont..

- Strategy 1
  - It would simply explore all possible path in the tree and return the one with the shortest length.
  - This approach will even work in practice for every short list of cities.
  - If there are N cities then the number of different path among them is 1,2, . . . (N-1) or (N-1)!
  - Not a feasible solution

Cont..

- Strategy
  - branch and bound.
  - Being generating complete paths keeping track of shortest path found so far
  - give up exploring any path as soon as its partial length become greater than the shortest path found so far .
  - Using this technique we are still guarenteed to find the shortest path.

Heuristic Search

Heuristics are rule for choosing branches in a state space that are most likely to lead to an acceptable problem solution.

- Two basic solution:
  - a problem may not have an exact solution
    - eg, Medical diagnosis: A given set of symptoms may have several possible causes, doctors use heuristics to choose the most likely diagnosis and formulate a plan of treatment.
  - A problem may have an exact solution, but the computational cost of finding it may be prohibitive.
    - A heuristic algorithm can defeat this combinational explosion and find an acceptable solution.

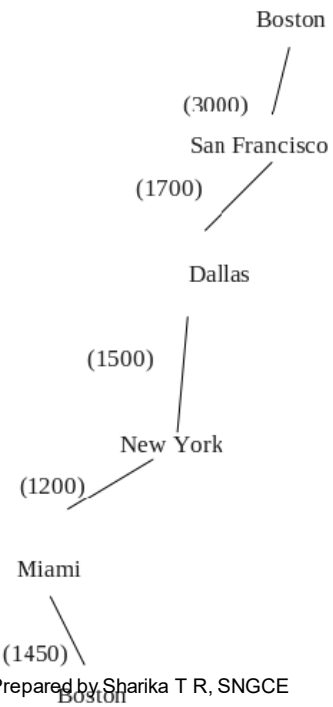
Cont..

- Heuristic approach for travelling salesman problem
  - arbitarily select a starting city
  - To select the next city, look at all cities not yet visited and select the one closest to the current city go to it next.
  - Repeat step 2 until all cities have been visited
- executes in N^2 time which is better than N!

Cont..

	Boston	NY	Miami	Dallas	SF
Boston		250	1450	1700	3000
NY	250		1200	1500	2900
Miami	1450	1200		1600	3300
Dallas	1700	1500	1600		1700
SF	3000	2900	3300	1700	

Cont..



Hill Climbing

- Generate-and-test + direction to move.
- Heuristic function to estimate how close a given state is to a goal state.
- Hill climbing is a variant of generate and test in which feedback from the test procedure is used to help the generator decide which direction to move in the search space.

Simple hill Climbing

1. Evaluate the initial state. If it is also a goal state, then return it 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 left to be applied in current 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 it is not a goal state but it is better than current state then make it the current state
      iii. If it is not better than current state then continue in the loop

Cont..

- The key difference between this algorithm and generate-and-test is the use of an evaluation function as a way to inject task specific knowledge into control proceed.
- In this algorithm we asked the relatively vague question, 'Is one state better than another".

Steepest Ascent Hill Climbing(Gradient Search)

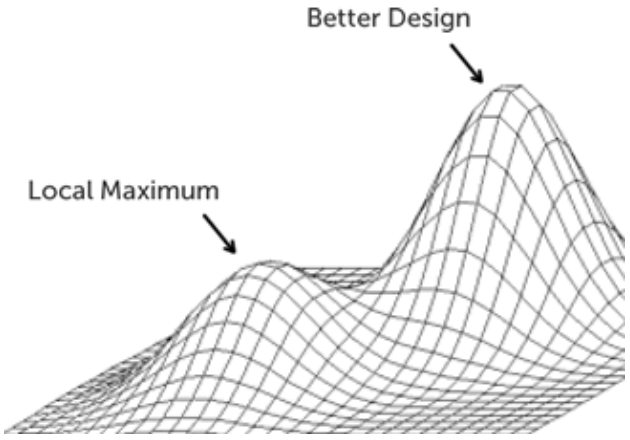
1. Evaluate the initial state. If it is also goal state, then return it and quit otherwise continue with the initial state as the current state.
2. Loop until a solution is found or until a complete iteration produces no change to current state:
   a. Let SUCC be a state such that any possible successor of the current state will be better than SUCC
   b. For each operator that applies to the current state do
      i. Apply the operator or generate a new state
      ii. Evaluate the new state. If it is a goal state, then return it and quit. If not compare it to SUCC. If it is better, then set SUCC to this state. If it is not better, leave SUCC alone.
      iii. If the SUCC is better than current state, then set current state to SUCC.

Cont..

- In steepest-ascent hill climbing we must consider all permuatations of the initial state and choose the best.
- Basic and steepest hill climbing may terminate not by finding a goal state but by getting to a state from which no better states can be generated.
- This will happen if the program has reached either a local maximum, plateau or ridge.

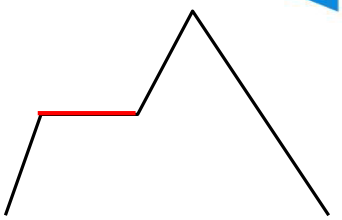
Local Maximum

- It is a state that is better than all its neighbors but is not better than sum other states farther away.



Plateau

- flat area of the search space in which a whole set of neighboring states have the same value.
- On a plateau it is not possible to determine the best direction in which to move by making local comparison



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

Ways for dealing with these problems

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

Local and Global Heuristics

- Hill Climbing is a local method
  - it decide what to do next by looking only the immediate consequence of its choice rather than by exhustively exploring all the consequences
  - it lacks guarentee that it will be effective
  - looks only one move ahead not any further

Example

- Local Heuristic function:
  - Add one point for every block that is resting on the thing it is supposed to be resting on.
  - Substract one point for every block that is sitting on wrong thing
  - Goal Score=8
  - Initial Score=4 (6-2)

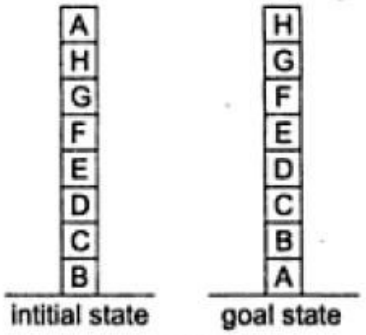


Fig. 3.1 *A Hill-Climbing Problem*

Cont..

- Move A to table makes State Score=6
- State Scores after moving H
  - (a) = 4
  - (b) =4
  - (c)=4
- Local Maximum reached
- to solve this it is necessary to disassemble a good local structure because it is wrong global structure

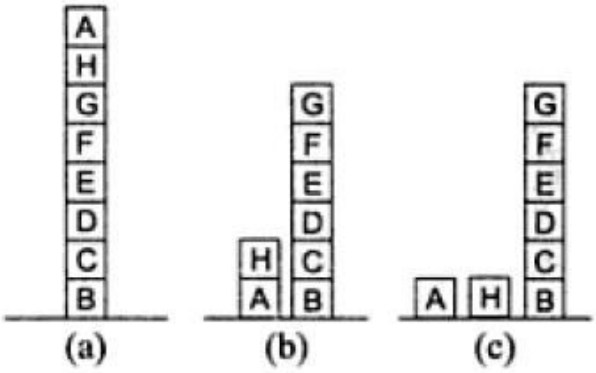


Fig. 3.2 *Three Possible Moves*

- We could modify the heuristic function to make it better.
- Global Heuristic Function:
  - For each block that has correct structure add one point for every block in support structure
  - For each block that has an incorrect support structure substract one point for every block in existing support structure
- That makes
  - INITIAL SCORE= -28
  - FINAL SCORE= 28 (A=0,B=1,C=2...)

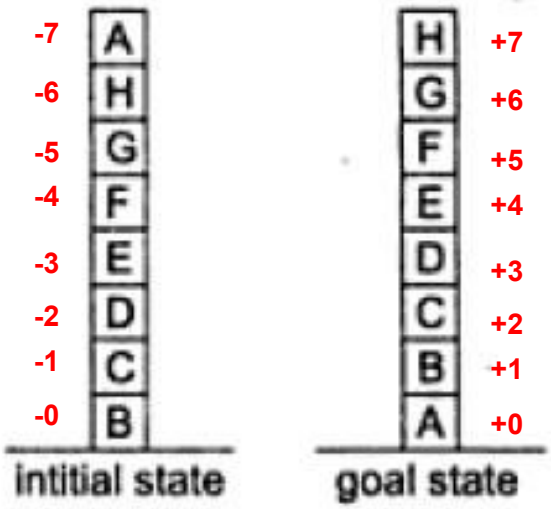


Fig. 3.1 *A Hill-Climbing Problem*

Cont..

- Moving A to table make score= -21 (-22+1)
  - since A do not have 7 wrong blocks under it
  - the same 3 states produce following scores
    - (a) = -28
    - (b) = -16
    - (c) = -15
  - Now we can choose (c)

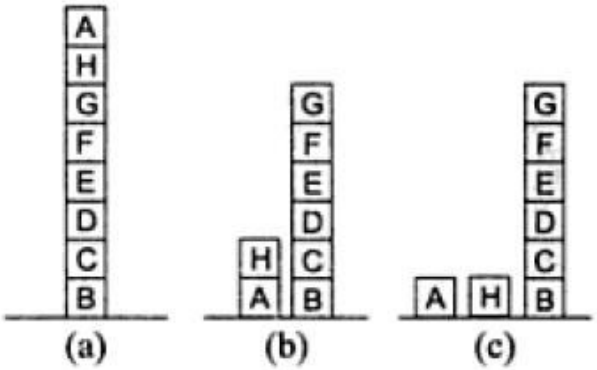


Fig. 3.2 *Three Possible Moves*

Cont..

- It is not always possible to construct such a perfect heuristic function
- hill climbing can be very ineffective in a large, rough problem space.
- it is useful when combined with other methods that get it started in right general neighborhood

- Assignment : Simulated Annealing

Simulated Annealing

- Simulated annealing is a variation of hill climbing in which at the begining of the process, some downhill moves may be made.

  do enough exploration of whole space early on so that the final solution is relatively insensitive to starting state

- This should lower the chances of getting caught at a local maximum, a plateau, or a ridge.

Cont..

- Two notational changes
  1. We use the term objective function in the place of the term heuristic function
  2. We attempt to minimize rather than maximize the value of the objective function

     valley descending rather than hill climbing
- Goal of this process is to produce a minimal-energy final state.
- objective function= energy level

Physical Anealing

- Metals melted and then gradually cooled until some solid state is reached
- But there is some probability that a transition to higher energy state will occur,

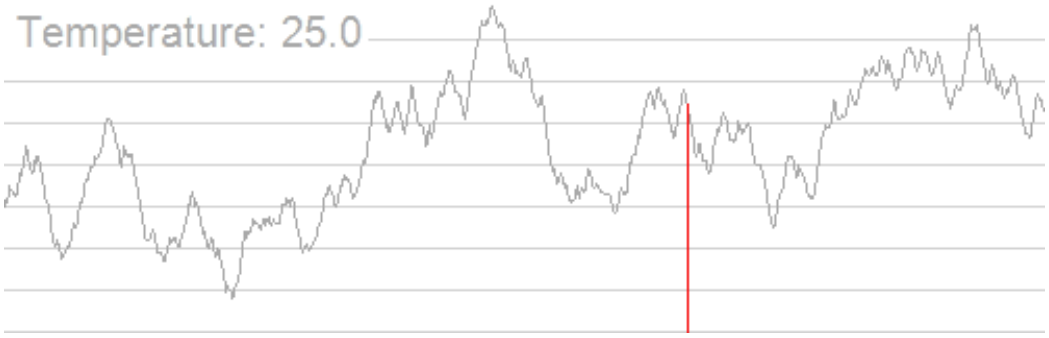
$$p = e^{-\Delta E / kT}$$

$\Delta E \rightarrow$ positive change in energy

$T \rightarrow Temperature$

$k \rightarrow$ Boltzmann's constant

- k describes the correspondence between the units of temprature and the units of energy



Cont..

- probability of a large uphill move is lower than probability of a small one

probability that an uphill move will be made decreases as temprature decreases.

- Rate at which system is cooled is called the annealing schedule
  - if cooling occurs too rapidly stable regions of high energy will form ie, local minimum will be reached
  - if cooling occur slowly uniform crystelline structure ie, global minimum will be reached
  - if too slow time is wasted

Cont..

- Explore successors wildly randomly==>High Temperature
- As time goes by explore less wildly ==> Cool Down
- Until her's a time where things settle ==> Cold

Simulated Annealing from Physical

1. Evaluate the initial state. If it is also a goal state, then return it and quit. Otherwise continue with the initial state as the current state.
2. Initialize BEST_SO_FAR to the current state
3. Initialize T according to the annealing schedule

Cont..

4. Loop until a solution is found or until there are no new operators left to be applied in the current 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 compute (Value of current) - (value of new)
      - If the new state is a goal state, then return it and quit
      - If it is not a goal state but it is better then make it the current state. Also set BEST_SO_FAR to this new state
      - If it is not better than the current state, then make it the current state with probability p' as defined above. This step is usually implemented by invoking a random number generator to produce a number in the range [0,1]. if that number is less than p', then the move is accepted. Otherwise do nothing.
   c) revise T as necessary according to the annealing schedule.
5. Return BEST_SO_FAR as the answer.

Cont..

- The annealing schedule has three components.
  1. the initial value to be used for temprature.
  2. the criteria that will be used to decide when the temprature of the system should be reduced.
  3. the amount by which the temprature will be reduced each time it is changed.
  4. There may also be a fourth component of the schedule, when to quit.
- T starts out high and gradually decreases towards 0.
- The higher the temprature the more likely it is that a bad move can be made.
- As T tends to zero, this probability tends to zero and simulated annealing become more like hill climbing.