Course Code	Course/Subject Name	Credits
CPC703	**Artificial Intelligence**	5

Objectives:

1. To conceptualize the basic ideas and techniques underlying the design of intelligent systems.
2. To make students understand and Explore the mechanism of mind that enable intelligent thought and action.
3. To make students understand advanced representation formalism and search techniques.
4. To make students understand how to deal with uncertain and incomplete information.

Outcomes: Learner will be able to

1. Ability to develop a basic understanding of AI building blocks presented in intelligent agents.
2. Ability to choose an appropriate problem solving method and knowledge representation technique.
3. Ability to analyze the strength and weaknesses of AI approaches to knowledge–intensive problem solving.
4. Ability to design models for reasoning with uncertainty as well as the use of unreliable information.
5. Ability to design and develop the AI applications in real world scenario.

Module	Detailed Contents	Hrs
01	**Introduction to Artificial Intelligence** 1.1 Introduction , History of Artificial Intelligence, Intelligent Systems: Categorization of Intelligent System, Components of AI Program, Foundations of AI, Sub-areas of AI, Applications of AI, Current trends in AI.	04
02	**Intelligent Agents** 2.1 Agents and Environments, The concept of rationality, The nature of environment, The structure of Agents, Types of Agents, Learning Agent.	04
03	**Problem solving** 3.1 Solving problem by Searching : Problem Solving Agent, Formulating Problems, Example Problems. 3.2 Uninformed Search Methods: Breadth First Search (BFS), Depth First Search (DFS) , Depth Limited Search, Depth First Iterative Deepening(DFID), Informed Search Methods: Greedy best first Search ,A* Search , Memory bounded heuristic Search. 3.3 Local Search Algorithms and Optimization Problems: Hill-climbing search Simulated annealing, Local beam search,	14

	Genetic algorithms. 3.4 Adversarial Search: Games, Optimal strategies, The minimax algorithm , Alpha-Beta Pruning.	
04	**Knowledge and Reasoning** 4.1 Knowledge based Agents, The Wumpus World, The Propositional logic, First Order Logic: Syntax and Semantic, Inference in FOL, Forward chaining, backward Chaining. 4.2 Knowledge Engineering in First-Order Logic, Unification, Resolution, Introduction to logic programming (PROLOG). 4.3 **Uncertain Knowledge and Reasoning:** Uncertainty, Representing knowledge in an uncertain domain, The semantics of belief network, Inference in belief network.	12
05	**Planning and Learning** 5.1 The planning problem, Planning with state space search, Partial order planning, Hierarchical planning, Conditional Planning. 5.2 Learning: Forms of Learning, Inductive Learning, Learning Decision Tree. 5.3 Expert System: Introduction, Phases in building Expert Systems, ES Architecture, ES vs Traditional System.	10
06	**Applications** 6.1 Natural Language Processing(NLP), Expert Systems.	04

Term Work:

The distribution of marks for term work shall be as follows:

- Laboratory work (experiments/case studies): …….. (15) Marks.
- Assignment:…….…… (05) Marks.
- Attendance ……. (05) Marks

TOTAL: ……. (25) Marks.

There will be at least two assignments covering the above syllabus.

Practical/Oral examination:

Practical examination based on the above syllabus will be conducted.

List of AI Practical / Experiments

All the programs should be implemented in C/C++/Java/Prolog under Windows or Linux environment. Experiments can also be conducted using available open source tools.

1. One case study on NLP/Expert system based papers published in IEEE/ACM/Springer or any prominent journal.

2. Program on uninformed and informed search methods.

3. Program on Local Search Algorithm.

4. Program on Optimization problem.

5. Program on adversarial search.

6. Program on Wumpus world.

7. Program on unification.

8. Program on Decision Tree.

Any other practical covering the syllabus topics and subtopics can be conducted.

Reference Books (Practicals):

1. Ivan Bratko "PROLOG Programming for Artificial Intelligence", Pearson Education, Third Edition.
2. Elaine Rich and Kevin Knight "Artificial Intelligence "Third Edition
3. Davis E.Goldberg, "Genetic Algorithms: Search, Optimization and Machine Learning", Addison Wesley, N.Y., 1989.
4. Han Kamber, “Data Mining Concepts and Techniques”, Morgann Kaufmann Publishers.

Text Books:

1. Stuart J. Russell and Peter Norvig, "Artificial Intelligence A Modern Approach “Second Edition" Pearson Education.
2. Saroj Kaushik “Artificial Intelligence” , Cengage Learning.
3. George F Luger “Artificial Intelligence” Low Price Edition , Pearson Education., Fourth edition.

Reference Books:

1. Ivan Bratko “PROLOG Programming for Artificial Intelligence”, Pearson Education, Third Edition.
2. Elaine Rich and Kevin Knight “Artificial Intelligence” Third Edition
3. Davis E.Goldberg, “Genetic Algorithms: Search, Optimization and Machine Learning”, Addison Wesley, N.Y., 1989.
4. Hagan, Demuth, Beale, “Neural Network Design” CENGAGE Learning, India Edition.
5. Patrick Henry Winston , “Artificial Intelligence”, Addison-Wesley, Third Edition.
6. Han Kamber, “Data Mining Concepts and Techniques”, Morgann Kaufmann Publishers.
7. N.P.Padhy, “Artificial Intelligence and Intelligent Systems”, Oxford University Press.

Ch.1 Introduction to Artificial Intelligence

1.1 Introduction

Definition of AI

Artificial Intelligence is a branch of *Science* which deals with helping machines find solutions to complex problems in a more human-like fashion. This generally involves borrowing characteristics from human intelligence, and applying them as algorithms in a computer friendly way. A more or less flexible or efficient approach can be taken depending on the requirements established, which influences how artificial the intelligent behavior appears.

AI is generally associated with *Computer Science*, but it has many important links with other fields such as *Maths*, *Psychology*, *Cognition*, *Biology* and *Philosophy*, among many others. Our ability to combine knowledge from all these fields will ultimately benefit our progress in the quest of creating an intelligent artificial being.

AI is a branch of computer science which is concerned with the study and creation of computer systems that exhibit

- ❑ some form of intelligence
  OR
- ❑ those characteristics which we associate with intelligence in human behavior

What is intelligence?

Intelligence is a property of mind that encompasses many related mental abilities, such as the capabilities to

- reason
- plan
- solve problems
- think abstractly
- comprehend ideas and language and
- learn

1.2 History of Artificial Intelligence

1943	McCulloch & Pitts: Boolean circuit model of brain
1950	Turing's "Computing Machinery and Intelligence"
1956	Dartmouth meeting: "Artificial Intelligence" adopted
1950s	Early AI programs, including Samuel's checkers program, Newell & Simon's Logic Theorist, Gelernter's Geometry Engine
1965	Robinson's complete algorithm for logical reasoning
1966—73	AI discovers computational complexity Neural network research almost disappears
1969—79	Early development of knowledge-based systems
1980--	AI becomes an industry
1986--	Neural networks return to popularity
1987--	AI becomes a science
1995--	The emergence of intelligent agents

1.3 Intelligent Systems: Categorization of Intelligent System

- Systems that think like humans
- Systems that act like humans
- Systems that think rationally
- Systems that act rationally

Systems that think like humans

- Most of the time it is a black box where we are not clear about our thought process.
- One has to know functioning of brain and its mechanism for possessing information.
- It is an area of cognitive science.
    - The stimuli are converted into mental representation.
    - Cognitive processes manipulate representation to build new representations that are used to generate actions.
- Neural network is a computing model for processing information similar to brain.

Systems that act like humans

- The overall behavior of the system should be human like.
- It could be achieved by observation.

Systems that think rationally

- Such systems rely on logic rather than human to measure correctness.
- For thinking rationally or logically, logic formulas and theories are used for synthesizing outcomes.
- For example,
    - given John is a human and all humans are mortal then one can conclude logically that John is mortal
- Not all intelligent behavior are mediated by logical deliberation.

Systems that act rationally

- Rational behavior means doing right thing.
- Even if method is illogical, the observed behavior must be rational.

1.4 Components of AI Program

AI techniques must be independent of the problem domain as far as possible.

AI program should have

a. knowledge base
b. navigational capability
c. inferencing

Knowledge Base

- AI programs should be learning in nature and update its knowledge accordingly.
- Knowledge base consists of facts and rules.
- Characteristics of Knowledge:
    - It is voluminous in nature and requires proper structuring
    - It may be incomplete and imprecise
    - It may keep on changing (dynamic)

Navigational Capability

- Navigational capability contains various control strategies
- Control Strategy
    - determines the rule to be applied
    - some heuristics (thump rule) may be applied

Inferencing

- Inferencing requires
    - search through knowledge base
      and
    - derive new knowledge

1.5 Foundations of AI

Foundation of AI is based on

1. Mathematics
2. Neuroscience
3. Control Theory
4. Linguistics

Mathematics

- More formal logical methods
    - Boolean logic
    - Fuzzy logic
- Uncertainty
    - The basis for most modern approaches to handle uncertainty in AI applications can be handled by
        - Probability theory
        - Modal and Temporal logics

Neuroscience

- How do the brain works?
    - Early studies (1824) relied on injured and abnormal people to understand what parts of brain work
    - More recent studies use accurate sensors to correlate brain activity to human thought
        - By monitoring individual neurons, monkeys can now control a computer mouse using thought alone
    - Moore's law states that computers will have as many gates as humans have neurons in 2020
    - How close are we to have a mechanical brain?
        - Parallel computation, remapping, interconnections,….

Control Theory

- Machines can modify their behavior in response to the environment (sense/action loop)
    - Water-flow regulator, steam engine governor, thermostat
- The theory of stable feedback systems (1894)
    - Build systems that transition from initial state to goal state with minimum energy
    - In 1950, control theory could only describe linear systems and AI largely rose as a response to this shortcoming

Linguistics

- Speech demonstrates so much of human intelligence
    - Analysis of human language reveals thought taking place in ways not understood in other settings
        - Children can create sentences they have never heard before
        - Language and thought are believed to be tightly intertwined

1.6 Sub-areas of AI

Sub areas of AI are:

a. Knowledge representation
b. Theorem proving
c. Game playing
d. Common sense reasoning dealing with uncertainty and decision making
e. Learning models, inference techniques, pattern recognition, search and matching etc.
f. Logic (fuzzy, temporal, modal) in AI
g. Planning and scheduling
h. Natural language understanding

i. Computer vision
j. Understanding spoken utterances
k. Intelligent tutoring systems
l. Robotics
m. Machine translation systems
n. Expert problem solving
o. Neural Networks, AI tools etc

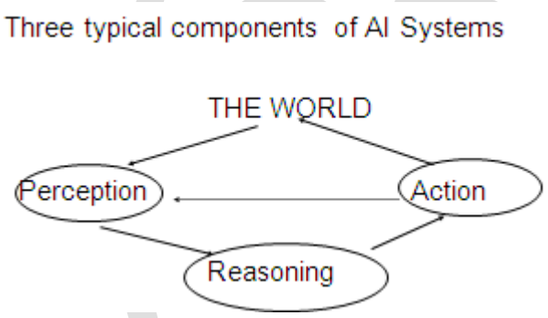
1.7 Applications of AI

Some of the applications are given below:

a. **Business :** Financial strategies, give advice
b. **Engineering:** check design, offer suggestions to create new product
c. **Manufacturing:** Assembly, inspection & maintenance
d. **Mining:** used when conditions are dangerous
e. **Hospital :** monitoring, diagnosing & prescribing
f. **Education :** In teaching
g. **Household:** Advice on cooking, shopping etc.
h. **Farming:** prune trees & selectively harvest mixed crops.

1.8 Current trends in AI

Latest Perception of AI



Heavy use of

a. probability theory
b. decision theory
c. statistics
d. logic (fuzzy, modal, temporal)

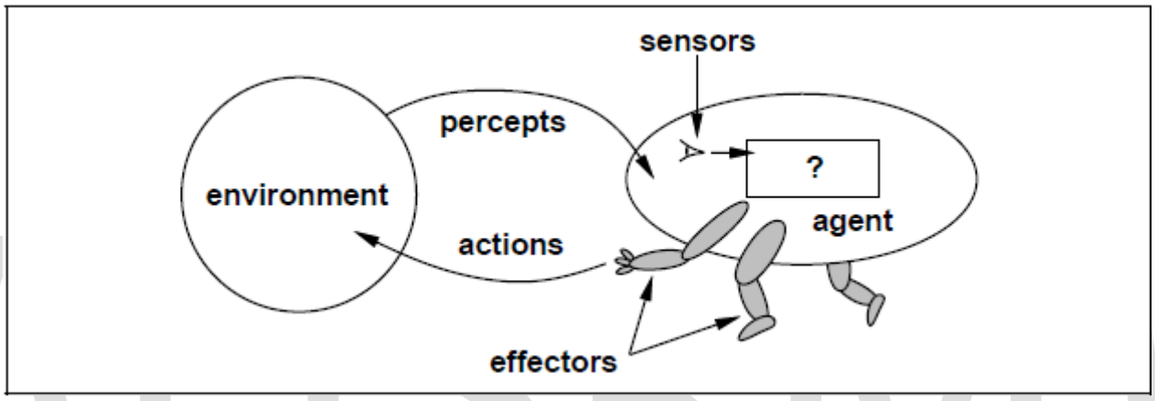
Ch.2 Intelligent Agent

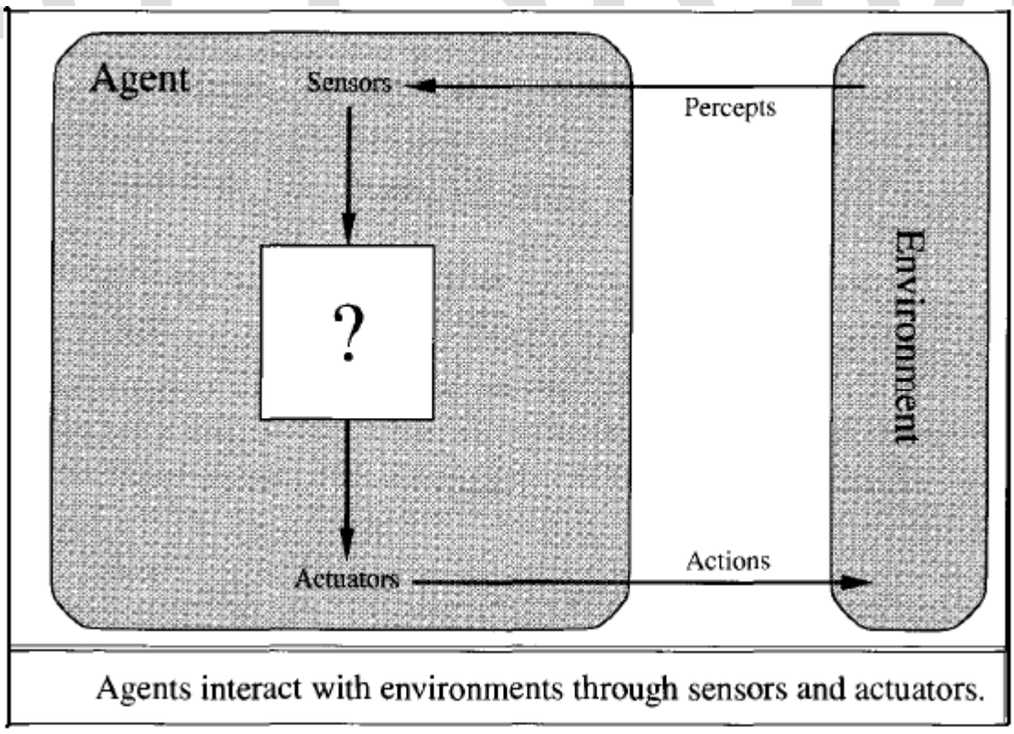
Artificial Intelligent and Agent

The branch of computer science concerned with making computers behave like humans.
"Artificial Intelligence is the study of human intelligence such that it can be replicated artificially."

An agent is anything that can be viewed as **perceiving** its environment through **sensors and acting** upon that environment through **effectors.**

- A human agent has eyes, ears, and other organs for sensors, and hands, legs, mouth, and other body parts for effectors.
- A robotic agent substitutes cameras and infrared range finders for the sensors and various motors for the effectors.
- A software agent has encoded bit strings as its percepts and actions.





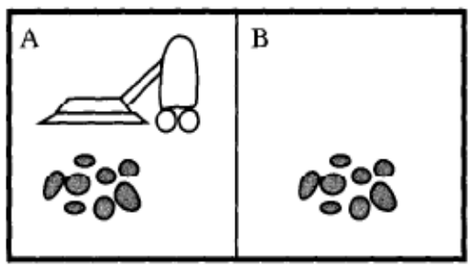
Simple Terms

- Percept
  - Agent's perceptual inputs at any given instant
- Percept sequence: Complete history of everything that the agent has ever perceived
- Agent's behavior is *mathematically* described by
  - **Agent function**
  - A function mapping any given percept sequence to an action
- *Practically* it is described by

- An **agent program**
- The real implementation

Example: Vacuum-cleaner world

- Perception: Clean or Dirty? Where it is in?
- Actions: Move left, Move right, suck(clean), do nothing(NoOp)



A vacuum-cleaner world with just two locations.

Percept sequence	Action
[A, Clean]	Right
[A, Dirty]	Suck
[B, Clean]	Left
[B, Dirty]	Suck
[A, Clean], [A, Clean]	Right
[A, Clean], [A, Dirty]	Suck
⋮	⋮
[A, Clean], [A, Clean], [A, Clean]	Right
[A, Clean], [A, Clean], [A, Dirty]	Suck
⋮	⋮

Figure 2.3 Partial tabulation of a simple agent function for the vacuum-cleaner world shown in above Figure

Program implements the **agent function** tabulated in above figure;

Function Reflex-Vacuum-Agent([*location,status*]) return an action
If *status = Dirty* **then return** *Suck*
else if *location = A* **then return** *Right*
else if *location = B* **then return** *left*

Concept of Rationality

- A **rational agent** is one that does the right thing. As a first approximation, we will say that the right action is the one that will cause the agent to be most successful.
- That leaves us with the problem of deciding *how* and *when* to evaluate the agent's success.
- We use the term **performance measure** for the *how*—the criteria that determine how successful an agent is.

- In summary, what is rational at any given time depends on four things:
  - ✓ The performance measure that defines degree of success.
  - ✓ Everything that the agent has perceived so far. We will call this complete perceptual history the **percept sequence.**
  - ✓ What the agent knows about the environment.
  - ✓ The actions that the agent can perform.

This leads to a definition of an **ideal rational agent**

For each possible percept sequence, an ideal rational agent should do whatever action is expected to maximize its performance measure, on the basis of the evidence provided by the percept sequence and whatever built-in knowledge the agent has.

Example of a rational agent

- Performance measure
  - Awards one point for each clean square
    - at each time step, over 10000 time steps
- Prior knowledge about the environment
  - The geography of the environment
  - Only two squares
  - The *effect* of the actions
- Actions that can perform
  - Left, Right, Suck and NoOp (No Operation)
- Percept sequences
  - Where is the agent?
  - Whether the location contains dirt?
  - Under this circumstance, the agent is rational.

The Nature of Environment

PEAS: To design a rational agent, we must specify the task environment

Consider, e.g., the task of designing an automated taxi:

Performance measure?

The performance given by taxi should make it most successful agent that is flawless performance.
e.g. Safety, destination, profits, legality, comfort, . . .

Environment?

It is a first step in designing an agent. We should specify the environment which suitable for agent action. If swimming is the task for an agent then environment must be water not air.
e.g. Streets/freeways, traffic, pedestrians, weather . . .

Actuators?

These are one of the important details of agent through which agent performs actions in related and specified environment.
e.g. Steering, accelerator, brake, horn, speaker/display, . . .

Sensors?

It is the way to receive different attributes from environment.
e.g. Cameras, accelerometers, gauges, engine sensors, keyboard, GPS . . .

(In designing an agent, the first step must always be to specify the task environment as fully as possible)

Agent Type	Performance Measure	Environment	Actuators	Sensors
Taxi driver	Safe, fast, legal, comfortable trip, maximize profits	Roads, other traffic, pedestrians, customers	Steering, accelerator, brake, signal, horn, display	Cameras, sonar, speedometer, GPS, odometer, accelerometer, engine sensors, keyboard

PEAS description of the task environment for an automated taxi.

Properties of task environments

Fully observable vs. partially observable

If an agent's sensors give it access to the complete state of the environment at each point in time then the environment is effectively and fully observable i.e. If the sensors detect all aspects and that are relevant to the choice of action.

An environment might be Partially observable because of noisy and inaccurate sensors or because parts of the state are simply missing from the sensor data.

Deterministic vs. **nondeterministic** (**stochastic**)

If the next state of the environment is completely determined by the current state and the actions selected by the agents, then we say the environment is deterministic. In principle, an agent need not worry about uncertainty in an accessible, deterministic environment. If the environment is inaccessible, however, then it may *appear* to be nondeterministic. This is particularly true if the environment is complex, making it hard to keep track of all the inaccessible aspects. Thus, it is often better to think of an environment as deterministic or nondeterministic *from the point of view of the agent*.

E.g. Taxi driving (non deterministic), humid environment (deterministic)

Episodic vs. **no episodic (Sequential)**.

In an episodic environment, the agent's experience is divided into "episodes." Each episode consists of the agent perceiving and then acting. The quality of its action depends just on the episode itself, because subsequent episodes do not depend on what actions occur in previous episodes. Episodic environments are much simpler because the agent does not need to think ahead.

E.g. chess (sequential)

Static vs. **dynamic**.

If the environment can change while an agent is deliberating, then we say the environment is dynamic for that agent; otherwise it is static. Static environments are easy to deal with because the agent need not keep looking at the world while it is deciding on an action, nor need it worry about the passage of time. If the environment does not change with the passage of time but the agent's performance score does, then we say the environment is semi dynamic.

Discrete vs. **continuous**.

If there are a limited number of distinct, clearly defined percepts and actions we say that the environment is discrete. Chess is discrete—there are a fixed number of possible moves on each turn. Taxi driving is continuous—the speed and location of the taxi and the other vehicles sweep through a range of continuous values.

Single agent VS. Multiagent

Playing a crossword puzzle – single agent and Chess playing – two agents

- Competitive multiagent environment-Chess playing
- Cooperative multiagent environment-Automated taxi driver for avoiding collision

Examples of task environments

Task Environment	Observable	Deterministic	Episodic	Static	Discrete	Agents
Crossword puzzle	Fully	Deterministic	Sequential	Static	Discrete	Single
Chess with a clock	Fully	Strategic	Sequential	Semi	Discrete	Multi
Poker	Partially	Stochastic	Sequential	Static	Discrete	Multi
Backgammon	Fully	Stochastic	Sequential	Static	Discrete	Multi
Taxi driving	Partially	Stochastic	Sequential	Dynamic	Continuous	Multi
Medical diagnosis	Partially	Stochastic	Sequential	Dynamic	Continuous	Single
Image-analysis	Fully	Deterministic	Episodic	Semi	Continuous	Single
Part-picking robot	Partially	Stochastic	Episodic	Dynamic	Continuous	Single
Refinery controller	Partially	Stochastic	Sequential	Dynamic	Continuous	Single
Interactive English tutor	Partially	Stochastic	Sequential	Dynamic	Discrete	Multi

Examples of task environments and their characteristics.

Structure of Intelligent agents

- The job of AI is to design the **agent program:** a function that implements the agent mapping from percepts to actions.
- We assume this program will run on some sort of computing device, which we will call the **architecture.** Obviously, the program we choose has to be one that the architecture will accept and run.
- The architecture might be a plain computer, or it might include special-purpose hardware for certain tasks, such as processing camera images or filtering audio input. It might also include software that provides a degree of insulation between the raw computer and the agent program, so that we can program at a higher level.
- In general, the architecture makes the percepts from the sensors available to the program, runs the program, and feeds the program's action choices to the effectors as they are generated.
- The relationship among agents, architectures, and programs can be summed up as follows:

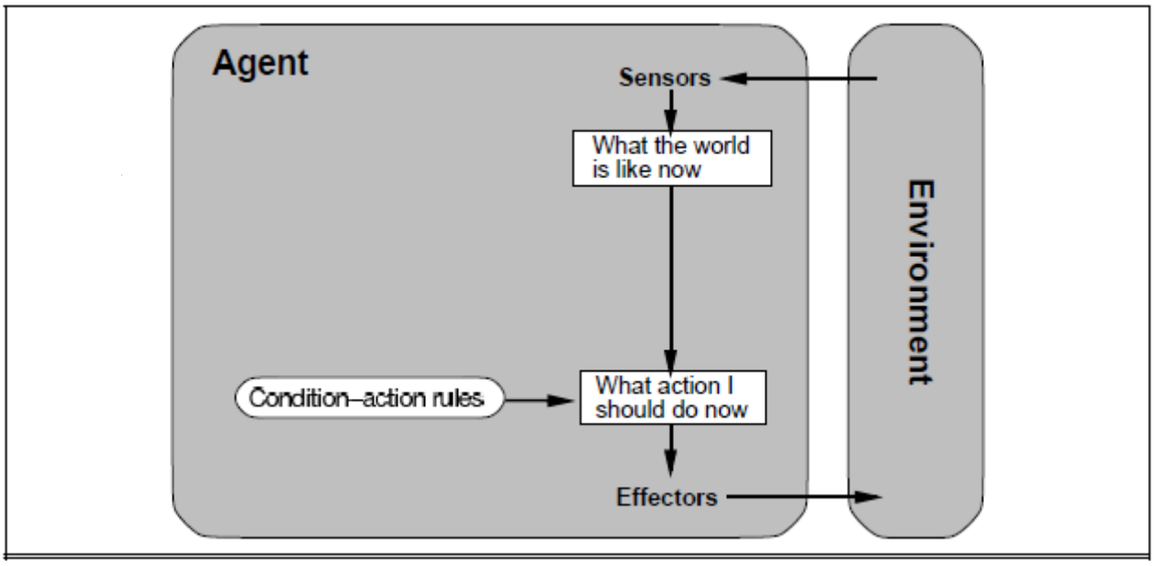
  Agent = architecture + program

- **Software agents** (or software robots or **softbot)** exist in rich, unlimited domains. Imagine a softbot designed to fly a flight simulator for a 747.
- The simulator is a very detailed, complex environment, and the software agent must choose from a wide variety of actions in real time.
- Now we have to decide how to build a real program to implement the mapping from percepts to action.
- We will find that different aspects of driving suggest different types of agent program.

Intelligent agents categories into five classes based on their degree of perceived intelligence and capability

1. Simple reflex agents
2. model-based reflex agents
3. goal-based agents
4. utility-based agents
5. Learning agents

Simple reflex agents

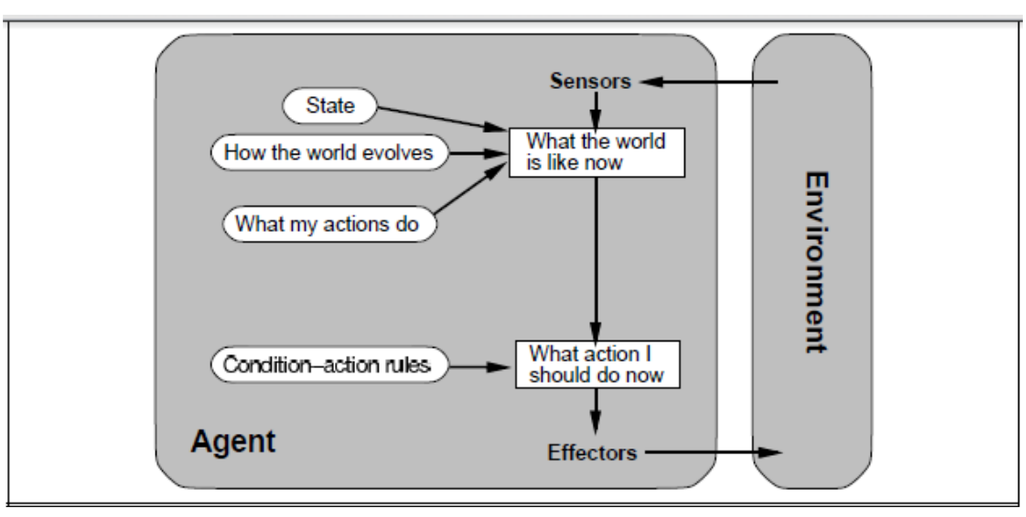



```
function SIMPLE-REFLEX-AGENT(percept) returns action
  static: rules, a set of condition-action rules

  state ← INTERPRET-INPUT(percept)
  rule ← RULE-MATCH(state, rules)
  action ← RULE-ACTION[rule]
  return action
```

- Simple reflex agents act only on the basis of the current percept, ignoring the rest of the percept history.
- The agent function is based on the *condition-action rule*: if condition then action.
- This agent function only succeeds when the environment is fully observable.
- Some reflex agents can also contain information on their current state which allows them to disregard conditions whose actuators are already triggered.
- Infinite loops are often unavoidable for simple reflex agents operating in partially observable environments.
- Note: If the agent can randomize its actions, it may be possible to escape from infinite loops.

Model-based reflex agents



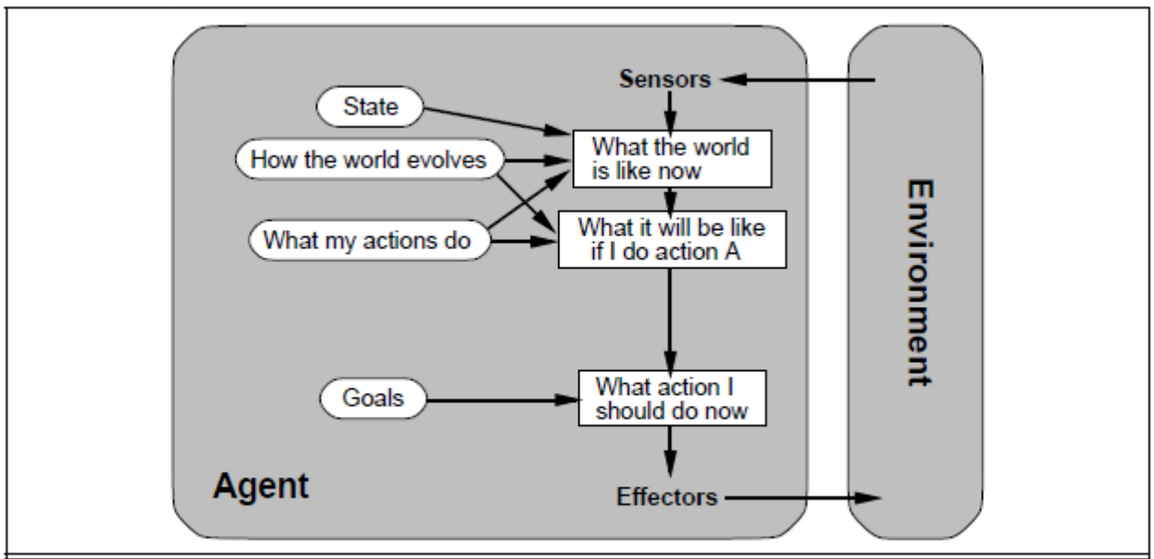
```
function REFLEX-AGENT-WITH-STATE(percept) returns action
  static: state, a description of the current world state
          rules, a set of condition-action rules

  state ← UPDATE-STATE(state, percept)
  rule ← RULE-MATCH(state, rules)
  action ← RULE-ACTION[rule]
  state ← UPDATE-STATE(state, action)
  return action
```

A reflex agent with internal state. It works by finding a rule whose condition matches the current situation (as defined by the percept and the stored internal state) and then doing the action associated with that rule.

- A model-based agent can handle a partially observable environment.
- Its current state is stored inside the agent maintaining some kind of structure which describes the part of the world which cannot be seen.
- This knowledge about "how the world works" is called a model of the world, hence the name "model-based agent".
- A model-based reflex agent should maintain some sort of internal model that depends on the percept history and thereby reflects at least some of the unobserved aspects of the current state.
- It then chooses an action in the same way as the reflex agent.

Goal-based agents

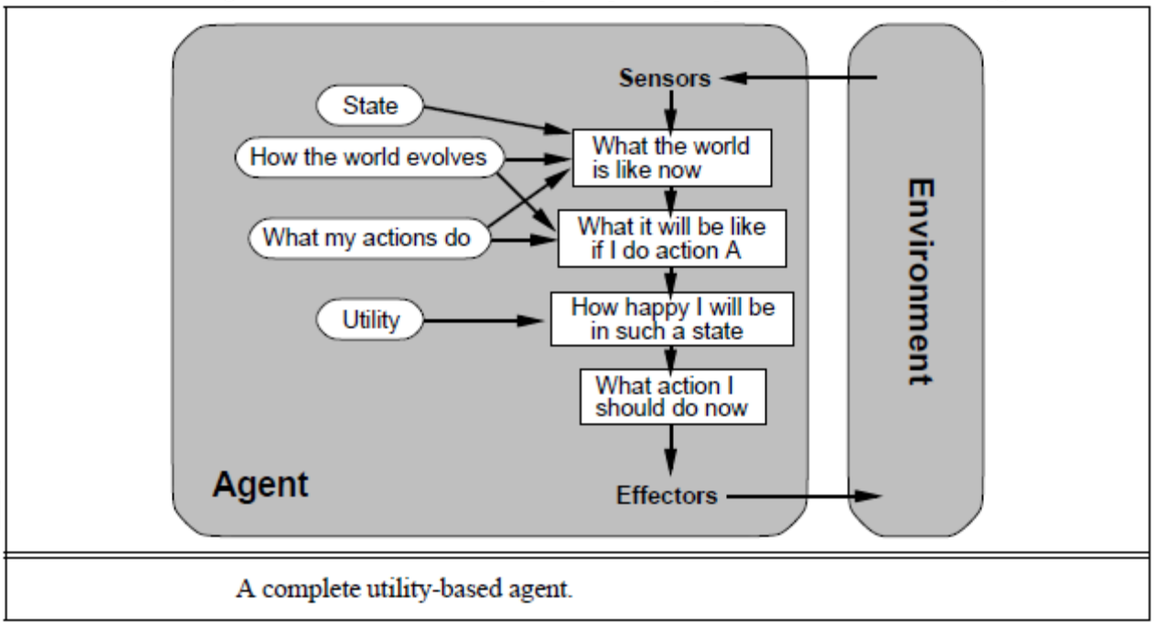


An agent with explicit goals.

- Goal-based agents further expand on the capabilities of the model-based agents, by using "goal" information.
- Goal information describes situations that are desirable.
- This allows the agent a way to choose among multiple possibilities, selecting the one which reaches a goal state.
- Search and planning are the subfields of artificial intelligence devoted to finding action sequences that achieve the agent's goals.
- In some instances the goal-based agent appears to be less efficient; it is more flexible because the knowledge that supports its decisions is represented explicitly and can be modified.

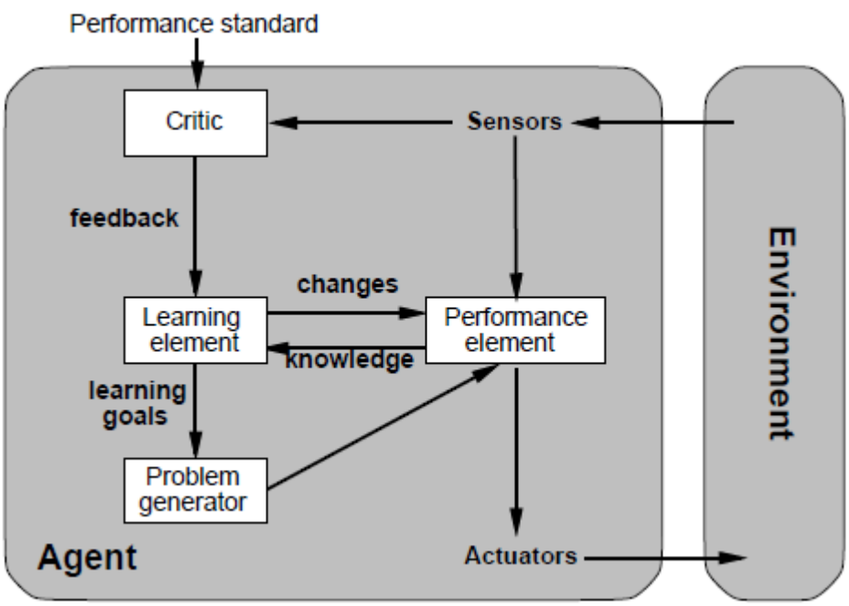
Utility-based agents

- Goal-based agents only distinguish between goal states and non-goal states.
- It is possible to define a measure of how desirable a particular state is.
- This measure can be obtained through the use of a *utility function* which maps a state to a measure of the utility of the state.
- A more general performance measure should allow a comparison of different world states according to exactly how happy they would make the agent.
- The term utility can be used to describe how "happy" the agent is.
- A rational utility-based agent chooses the action that maximizes the expected utility of the action outcomes- that is, the agent expects to derive, on average, given the probabilities and utilities of each outcome.
- A utility-based agent has to model and keep track of its environment, tasks that have involved a great deal of research on perception, representation, reasoning, and learning.



A complete utility-based agent.

Learning agents



- Learning has an advantage that it allows the agents to initially operate in unknown environments and to become more competent than its initial knowledge alone might allow.

- The most important distinction is between the "learning element", which is responsible for making improvements, and the "performance element", which is responsible for selecting external actions.

- The learning element uses feedback from the "critic" on how the agent is doing and determines how the performance element should be modified to do better in the future.

- The performance element is what we have previously considered to be the entire agent: it takes in percepts and decides on actions.

- The last component of the learning agent is the "problem generator".

- It is responsible for suggesting actions that will lead to new and informative experiences.

Ch.3 Problem Solving

PROBLEM-SOLVING AGENTS

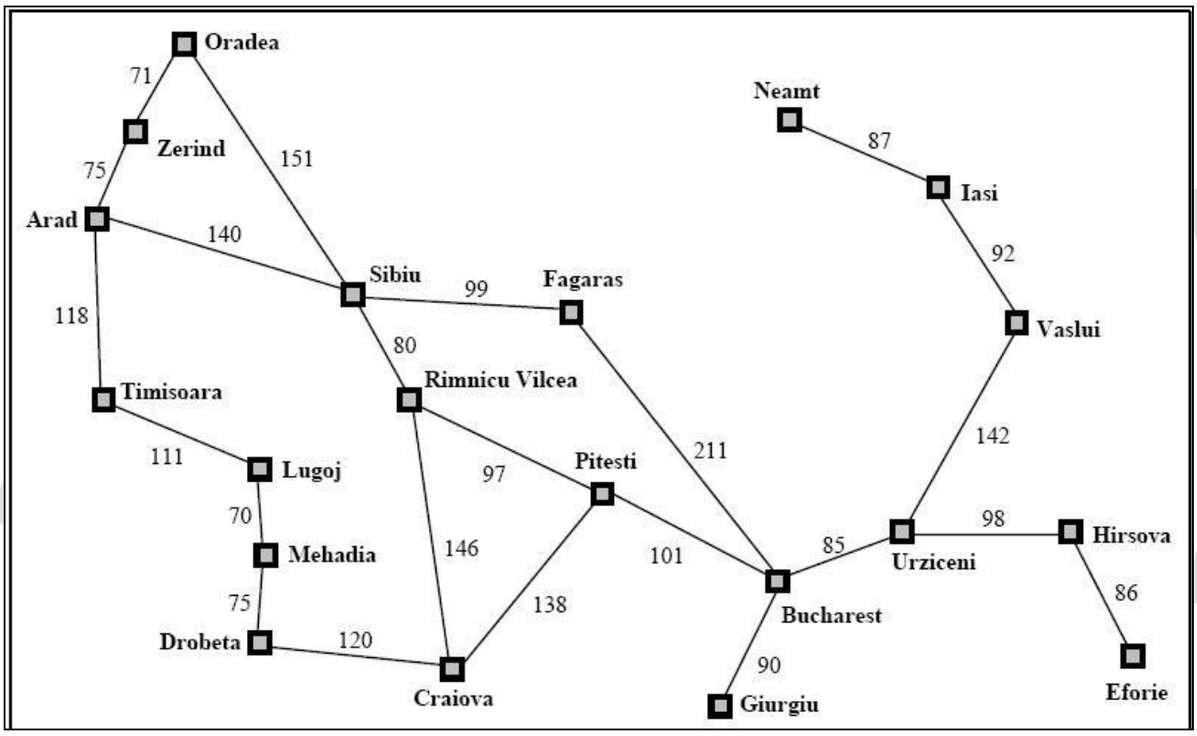
Intelligent agents are supposed to act in such a way that the environment goes through a sequence of states that maximizes the performance measure.

What is Problem solving agent?

- It is a kind of Goal-based Agents
- 4 general steps in problem-solving:

  1. Goal Formulation
  2. Problem Formulation
  3. Search
  4. Execute
- E.g. Driving from Arad to Bucharest...

***Note**: In this chapter we will consider one example that "A map is given with different cities connected and their distance values are also mentioned. Agent starts from one city and reach to other."*



Subclass of goal-based agents

- Goal formulation
- Problem formulation
- Example problems
  - Toy problems
  - Real-world problems
- search
  - search strategies
  - Constraint satisfaction
- solution

Goal Formulation

Goal formulation, based on the current situation, is the first step in problem solving. As well as formulating a goal, the agent may wish to decide on some other factors that affect the desirability of different ways of achieving the goal. For now, let us assume that the agent will consider actions at the level of driving from one major town to another. The states it will consider therefore correspond to being in a particular town.

- Declaring the Goal: Goal information given to agent *i.e. start from Arad and reach to Bucharest.*

- Ignoring the some actions: agent has to ignore some actions that will not lead agent to desire goal. *i.e. there are three roads out of Arad, one toward Sibiu, one to Timisoara, and one to Zerind. None of these achieves the goal, so unless the agent is very familiar with the geography of Romania, it will not know which road to follow. In other words, the agent will not know which of its possible actions is best, because it does not know enough about the state that results from taking each action.*
- Limits the objective that agent is trying to achieve: Agent will decide its action when he has some added knowledge about map. *i.e. map of Romania is given to agent.*
- Goal can be defined as set of world states: The agent can use this information to consider subsequent stages of a hypothetical journey through each of the three towns, to try to find a journey that eventually gets to Bucharest. *i.e. once it has found a path on the map from Arad to Bucharest, it can achieve its goal by carrying out the driving actions.*

Problem Formulation

Problem formulation is the process of deciding what actions and states to consider, given a goal.

- **Process of looking for action sequence (number of action that agent carried out to reach to goal) is called search. A search algorithm takes a problem as input and returns a solution in the form of an action sequence. Once a solution is found, the actions it recommends can be carried out. This is called the execution phase. Thus, we have a simple "formulate, search, execute" design for the agent.**

Well-defined problem and solutions.

A *problem* is defined by four items:

Initial state: *The initial state that the agent starts in.* e.g., the initial state for our agent in *Romania might be described as "In(Arad)".*

Successor function **S(x)** = A description of the possible actions available to the agent. The most common formulation uses a successor function , given a particular state x, SUCCESSOR-FN(x) returns a set of <action, successor> ordered pair where each action is one of the legal actions in state x and each successor is a state that can be reached from x by applying the action. e.g. ,from state *In(Arad),*the successor function for Romania problem would return {<Go(Zerind),In(Zerind)>, <Go(sibiu),In(sibiu)>, <Go(Timisoara),In(Timisoara)>}.

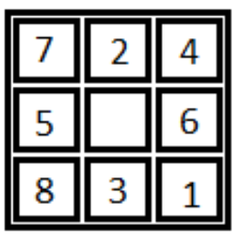
Goal test=It determines whether a given state is a goal state.

path cost (additive)=Function that assigns a numeric cost to each path. e.g., sum of distances, number of actions executed, etc. Usually given as c(x, a, y), the step cost from x to y by action a, assumed to be ≥ 0.

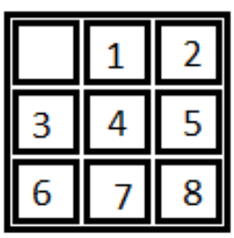
"A *solution* is a sequence of actions leading from the initial state to a goal state".

Example:

The 8-puzzle consists of a 3x3 board with 8 numbered tiles and a blank space. A tile adjacent to the blank space can slide into the space.



Initial state



Final State

The standard formulation is as follows:

States: A state description specifies the location of each of the eight tiles and blank is one of the nine squares.

Initial Function: any state can be designated as the initial state. Note that any given goal can be reached from exactly half of the possible initial states.

Successor function: This generates the legal states that result from trying the four actions (blank *moves Left, Right, Up or Down*).

Goal State: This checks whether the state matches the goal configuration shown in figure.
Path cost: Each step cost 1; so the path cost is the number of steps in path.

There are two types of searching strategies are used in path finding,
1) Uninformed Search strategies.
2) Infirmed Search strategies.

Uninformed Search Methods

Uninformed search means that they have no additional information about states beyond that provided in the problem definition. All they can do is generate successors and distinguish a goal state from non-goal state.

***Uninformed* strategies** use only the information available in the problem definition

1) Breadth-first search
2) Depth-first search
3) Depth-limited search
4) Iterative deepening search

Note: (Only for Understanding)

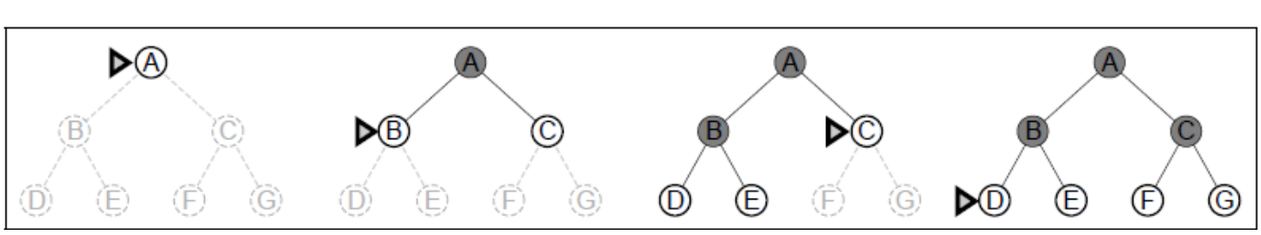
1) It is important to understand the distinction between nodes and states.
   A node is book keeping data structure used to represent the search tree.
   A state corresponds to a configuration of the world.
2) We also need to represent the collection of nodes that have been generated but not yet expanded; this collection is called the **fringe.**
3) In AI ,where the graph is represented implicitly by the initial state and successor function and is frequently infinite , its complexity expressed in terms of three quantities:

   b ; the branching factor or maximum number of successor of any node.
   d; the depth of shallowest goal node; and
   m; the maximum length of any path in the state space.

Breadth First Search (BFS Algorithm)

- **Breadth First Search** (BFS) searches breadth-wise in the problem space.
- Breadth-First search is like traversing a tree where each node is a state which may be a potential candidate for solution.
- Breadth first search expands nodes from the root of the tree and then generates one level of the tree at a time until a solution is found.
- It is very easily implemented by maintaining a queue of nodes.
- Initially the queue contains just the root.
- In each iteration, node at the head of the queue is removed and then expanded.
- The generated child nodes are then added to the tail of the queue.

Algorithm:
1. Place the starting node.
2. If the queue is empty return failure and stop.
3. If the first element on the queue is a goal node, return success and stop otherwise.
4. Remove and expand the first element from the queue and place all children at the end of the queue in any order.
5. Go back to step 1.



Advantages:
Breadth first search will never get trapped exploring the useless path forever. If there is a solution, BFS will definitely find it out. If there is more than one solution then BFS can find the minimal one that requires less number of steps.

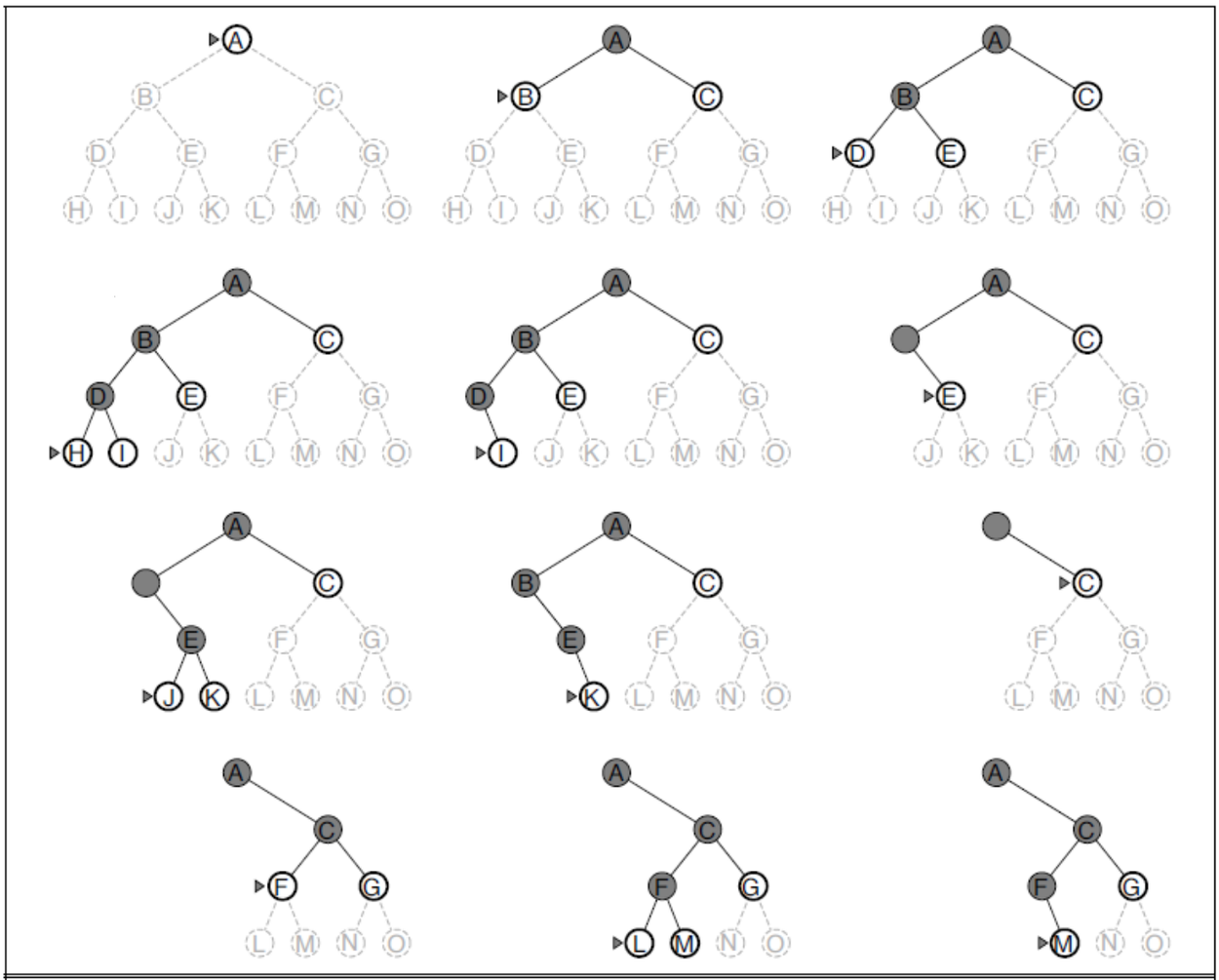
Disadvantages:
If the solution is farther away from the root, breath first search will consume lot of time.

Depth First Search (DFS)
Depth-first search (**DFS**) is an algorithm for traversing or searching a tree, tree structure, or graph. One starts at the root (selecting some node as the root in the graph case) and explores as far as possible along each branch before backtracking.

Algorithm:
1. Push the root node onto a stack.
2. Pop a node from the stack and examine it.
   - If the element sought is found in this node, quit the search and return a result.
   - Otherwise push all its successors (child nodes) that have not yet been discovered onto the stack.
3. If the stack is empty, every node in the tree has been examined – quit the search and return "not found".
4. If the stack is not empty, repeat from Step 2.

Advantages:
- If depth-first search finds solution without exploring much in a path then the time and space it takes will be very less.
- The advantage of depth-first Search is that memory requirement is only linear with respect to the search graph. This is in contrast with breadth-first search which requires more space.

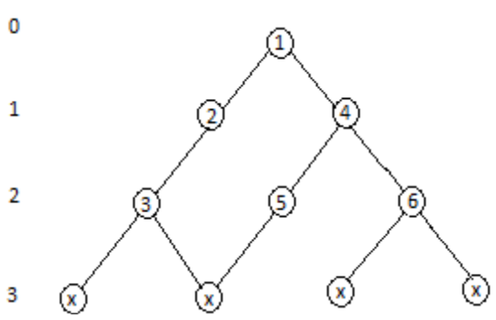
Disadvantages:
- Depth-First Search is not guaranteed to find the solution.
- It is not complete algorithm, if it go into infinite loops.



Depth-first search on a binary tree. The unexplored region is shown in light gray. Explored nodes with no descendants in the frontier are removed from memory. Nodes at depth 3 have no successors, and M is the only goal node.

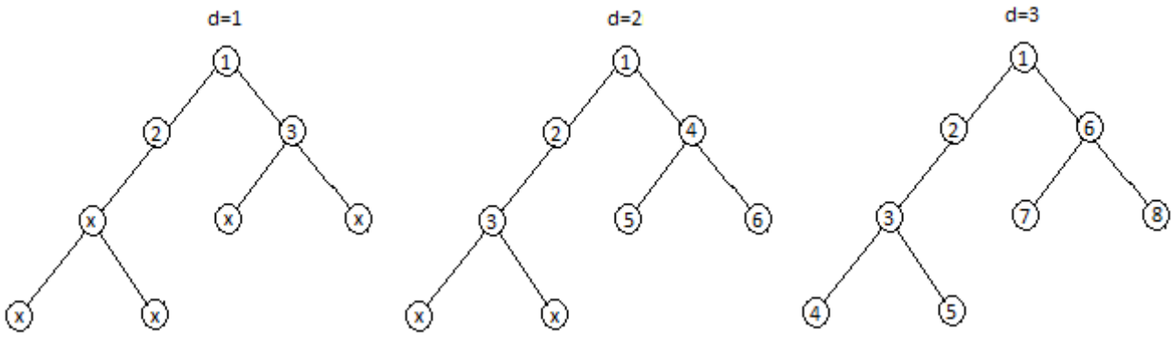
Depth Limited Search:

- Depth limited search (DLS) is a modification of depth-first search that minimizes the depth that the search algorithm may go.
- In addition to starting with a root and goal node, a depth is provided that the algorithm will not descend below.
- Any nodes below that depth are omitted from the search.
- This modification keeps the algorithm from indefinitely cycling by halting the search after the pre-imposed depth.
- The time and space complexity is similar to DFS from which the algorithm is derived.
- Space complexity: O(bd) and Time complexity : O(bd)



Iterative Deepening Search:

- Iterative Deepening Search (IDS) is a derivative of DLS and combines the feature of depth-first search with that of breadth-first search.
- IDS operate by performing DLS searches with increased depths until the goal is found.
- The depth begins at one, and increases until the goal is found, or no further nodes can be enumerated.
- By minimizing the depth of the search, we force the algorithm to also search the breadth of a graph.
- If the goal is not found, the depth that the algorithm is permitted to search is increased and the algorithm is started again.



Comparing Search Strategies:

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Time	b^d	b^d	b^m	b^l	b^d	$b^{d/2}$
Space	b^d	b^d	bm	bl	bd	$b^{d/2}$
Optimal?	Yes	Yes	No	No	Yes	Yes
Complete?	Yes	Yes	No	Yes, if $l \geq d$	Yes	Yes

Informed Search techniques:

- A strategy that uses problem-specific knowledge beyond the definition of the problem itself.
- Also known as "heuristic search," informed search strategies use information about the domain to (try to) (usually) head in the general direction of the goal node(s)
- Informed search methods: Hill climbing, best-first, greedy search, beam search, A, A*.

Best First Search:

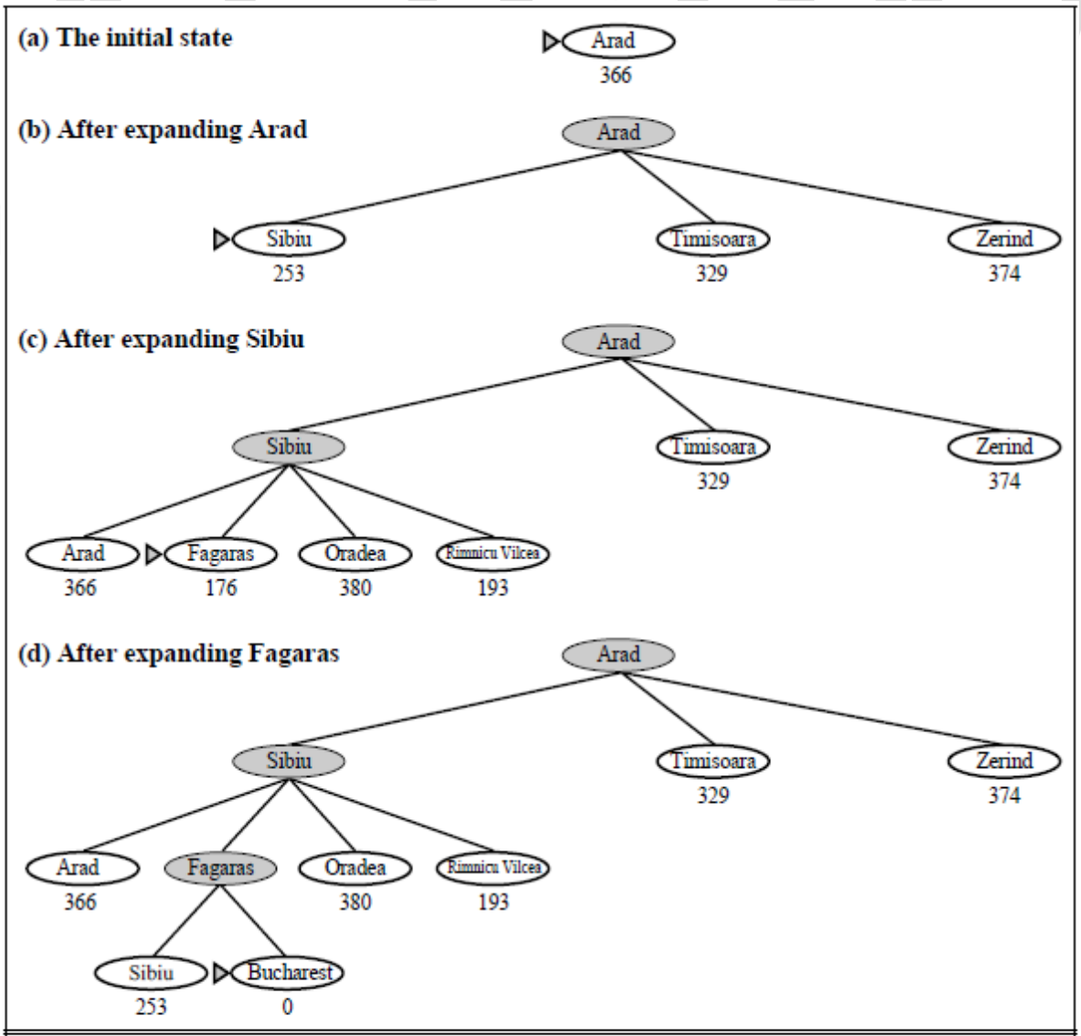
- It is an algorithm in which a node is selected for expansion based on an evaluation function f(n).
- Traditionally the node with the lowest evaluation function is selected.
- Not an accurate name…expanding the best node first would be a straight march to the goal.
- Choose the node that *appears* to be the best.
- There is a whole family of Best-First Search algorithms with different evaluation functions. Each has a heuristic function h(n).
- *h(n) = estimated cost of the cheapest path from node n to a goal node*
- Example: in route planning the estimate of the cost of the cheapest path might be the straight line distance between two cities.
- Quick Review,
  - g(n) = cost from the initial state to the current state n
  - h(n) = estimated cost of the cheapest path from node n to a goal node
  - f(n) = evaluation function to select a node for expansion (usually the lowest cost node)
- Best-First Search can be represented in two ways,
  - Greedy Best-First Search
  - A* search

Greedy Best-First Search:

• Greedy Best-First search tries to expand the node that is closest to the goal assuming it will lead to a solution quickly
  – f(n) = h(n)
  – aka "Greedy Search"
• Implementation
  – Expand the "most desirable" node into the fringe queue.
  – Sort the queue in decreasing order of desirability.
• Example: consider the straight-line distance heuristic hSLD
  – Expand the node that appears to be closest to the goal.
• hSLD(In(Arid)) = 366
• Notice that the values of hSLD cannot be computed from the problem itself
• It takes some experience to know that hSLD is correlated with actual road distances
  – Therefore a useful heuristic.

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

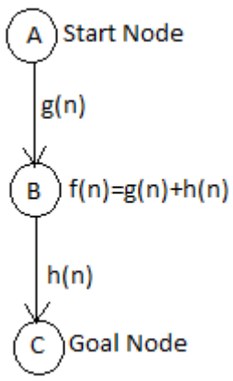
Values of h_{SLD}—straight-line distances to Bucharest. (Refer Pg.no.49 for Map of Romania)



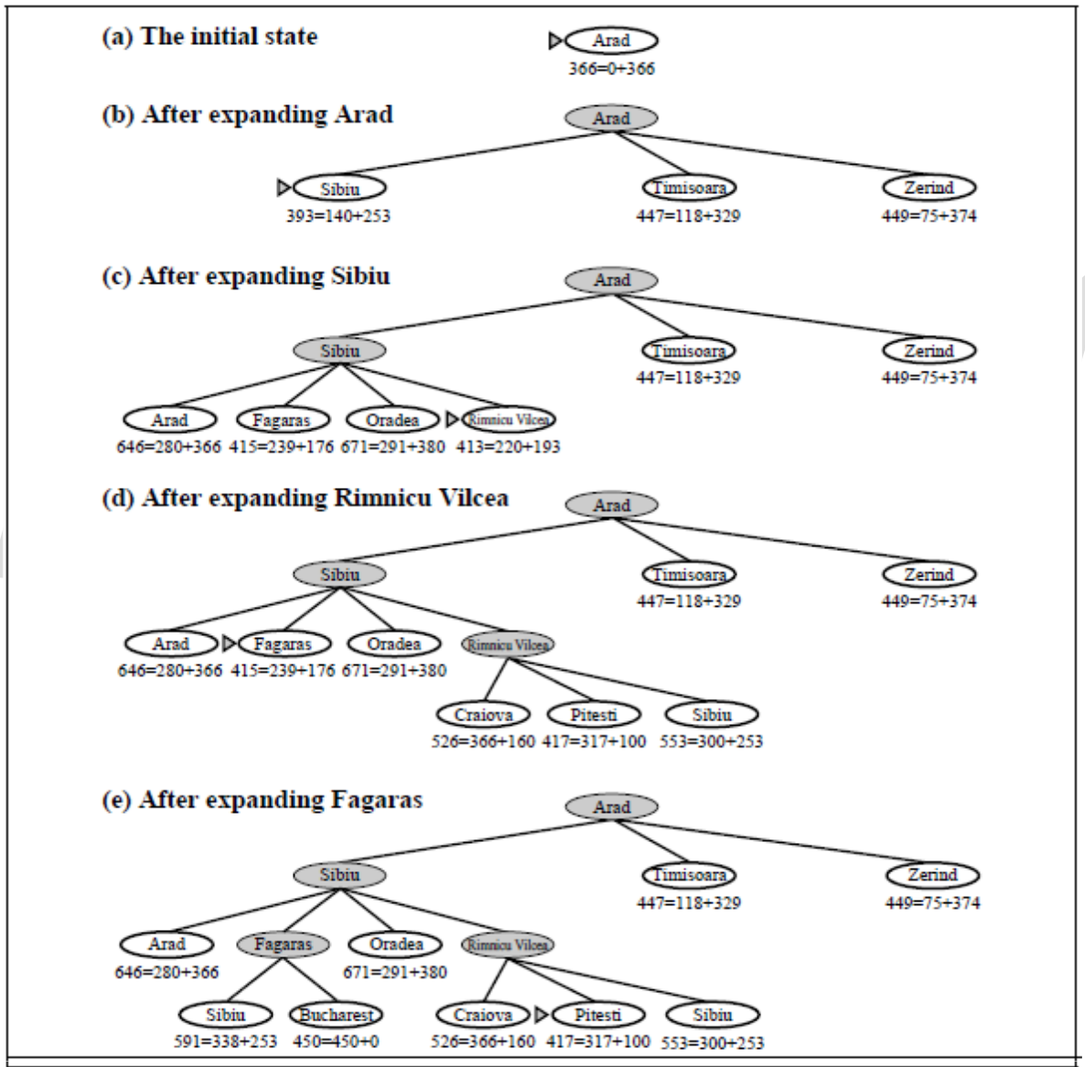
Stages in a greedy best-first tree search for Bucharest using the straight-line distance heuristic h_{SLD}. Nodes are labeled with their h-values.

So the path is: Arad-Sibiu-Fagaras-Bucharest

A* search



- A* (A star) is the most widely known form of Best-First search
  - It evaluates nodes by combining g(n) and h(n).
  - f(n) = g(n) + h(n).
  - Where
    - g(n) = cost so far to reach n.
    - h(n) = estimated cost to goal from n.
    - f(n) = estimated total cost of path through n.
- When h(n) = actual cost to goal
  - Only nodes in the correct path are expanded
  - Optimal solution is found
- When h(n) < actual cost to goal
  - Additional nodes are expanded
  - Optimal solution is found
- When h(n) > actual cost to goal
  - Optimal solution can be overlooked.



Stages in an A* search for Bucharest. Nodes are labeled with $f = g + h$. The h values are the straight-line distances to Bucharest taken from Pg.no.58

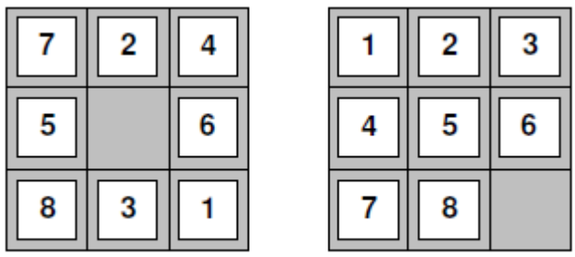
The main drawback of A* algorithm and indeed of any best-first search is its memory requirement.

Heuristic Function:

"A rule of thumb, simplification, or educated guess that reduces or limits the search for solutions in domains that are difficult and poorly understood."

- ***h(n)* = estimated cost of the cheapest path from node *n* to goal node.**
- **If *n* is goal then *h(n)=0***

It is a technique which evaluation the state and finds the significance of that state w.r.t. goal state because of this it is possible to compare various states and choose the best state to visit next.
Heuristics are used to improve efficiency of search process. The search can be improved by evaluating the states. The states can be evaluated by applying some evaluation function which tells the significance of state to achieve goal state. The function which evaluates the state is the heuristic function and the value calculated by this function is heuristic value of state.
The heuristic function is represented as h(n)
Eg. 8-puzzle problem



Start State Goal State

Admissible heuristics
$h1(n)$ = number of misplaced tiles
$h2(n)$ = total Manhattan distance (i.e., no. of squares from desired location of each tile)
In the example
$h1(S) = 6$
$h2(S) = 2 + 0 + 3 + 1 + 0 + 1 + 3 + 4 = 14$
If $h2$ dominates $h1$, then $h2$ is better for search than $h1$.

Memory Bounded Heuristic Search

The simplest way to reduce memory requirements for A* is to adapt the idea of iterative deepening search to the heuristic search context.

Types of memory bounded algorithms

1. **Iterative deepening A*(IDA*)-** Here cutoff information is the *f*-cost (*g+h*) instead of depth
2. **Recursive best first search (RBFS) -** Recursive algorithm that attempts to mimic standard best-first search with linear space.
3. **Simplified Memory bounded A* (SMA*)-** Drop the worst-leaf node when memory is full

Iterative deepening A*(IDA*)

Just as iterative deepening solved the space problem of breadth-first search, iterative deepening A* (IDA*) eliminates the memory constraints of A* search algorithm without sacrificing solution optimality. Each iteration of the algorithm is a depth-first search that keeps track of the cost, f(n) = g(n) + h(n), of each node generated. As soon as a node is generated whose cost exceeds a threshold for that iteration, its path is cut off, and the search backtracks before continuing. The cost threshold is initialized to the heuristic estimate of the initial state, and in each successive iteration is increased to the total cost of the lowest-cost node that was pruned during the previous iteration. The algorithm terminates when a goal state is reached whose total cost does not exceed the current threshold.

Since Iterative Deepening A* performs a series of depth-first searches, its memory requirement is linear with respect to the maximum search depth. In addition, if the heuristic function is admissible, IDA* finds an optimal solution. Finally, by an argument similar to that presented for DFID, IDA* expands the same number of nodes, asymptotically, as A* on a tree, provided that the number of

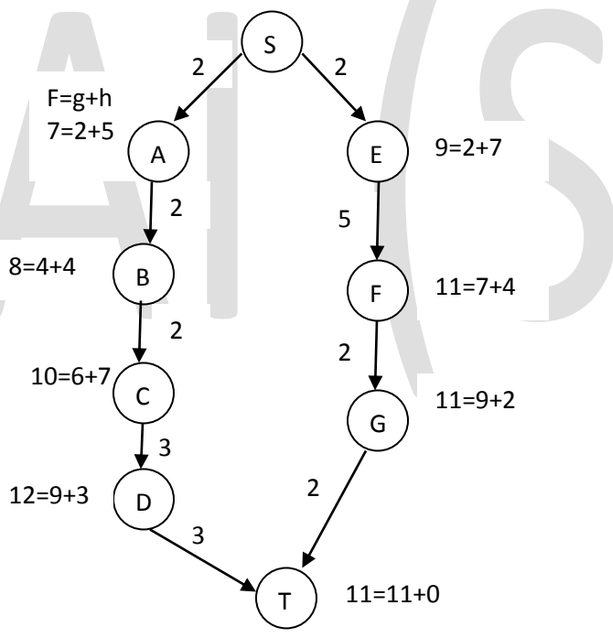
nodes, asymptotically, as A* on a tree, provided that the number of nodes grows exponentially with solution cost. These costs, together with the optimality of A*, imply that IDA* is asymptotically optimal in time and space over all heuristic search algorithms that find optimal solutions on a tree. Additional benefits of IDA* are that it is much easier to implement, and often runs faster than A*, since it does not incur the overhead of managing the open and closed lists.

Recursive best first search (RBFS)

IDA* search is no longer a best-first search since the total cost of a child can be less than that of its parent, and thus nodes are not necessarily expanded in best-first order. Recursive Best-First Search (RBFS) is an alternative algorithm. Recursive best-first search is a best-first search that runs in space that is linear with respect to the maximum search depth, regardless of the cost function used. Even with an admissible cost function, Recursive Best-First Search generates fewer nodes than IDA*, and is generally superior to IDA*, except for a small increase in the cost per node generation.

Keeps track of the f-value of the best-alternative path available.

- If current f-values exceeds this alternative f-value than backtrack to alternative path.
- Upon backtracking change f-value to best f-value of its children.
- Re-expansion of this result is thus still possible.



– S is expanded – A found to be best child
– A is expanded with bound 9 – C has F value 10,stop expansion – Backup F value
– Forget expansion from A – A has backed up F value 10 – E is best to expand next
– E is expanded with bound 10 – F has F-value 11 – Stop expansion, back up F value
– Forget expansion from E – E has backed up F value 10 – A is best to expand next
– When B and C are regenerated ,they inherit F value 10 from parent – A is expanded with bound 11 – D has F value 12 – Stop expansion, back up F value
– Forget expansion from A – A has backed up F value 12 – E is best to expand next
– E is expanded with bound 12 – Reach goal, search ends

For above example go through class notes.

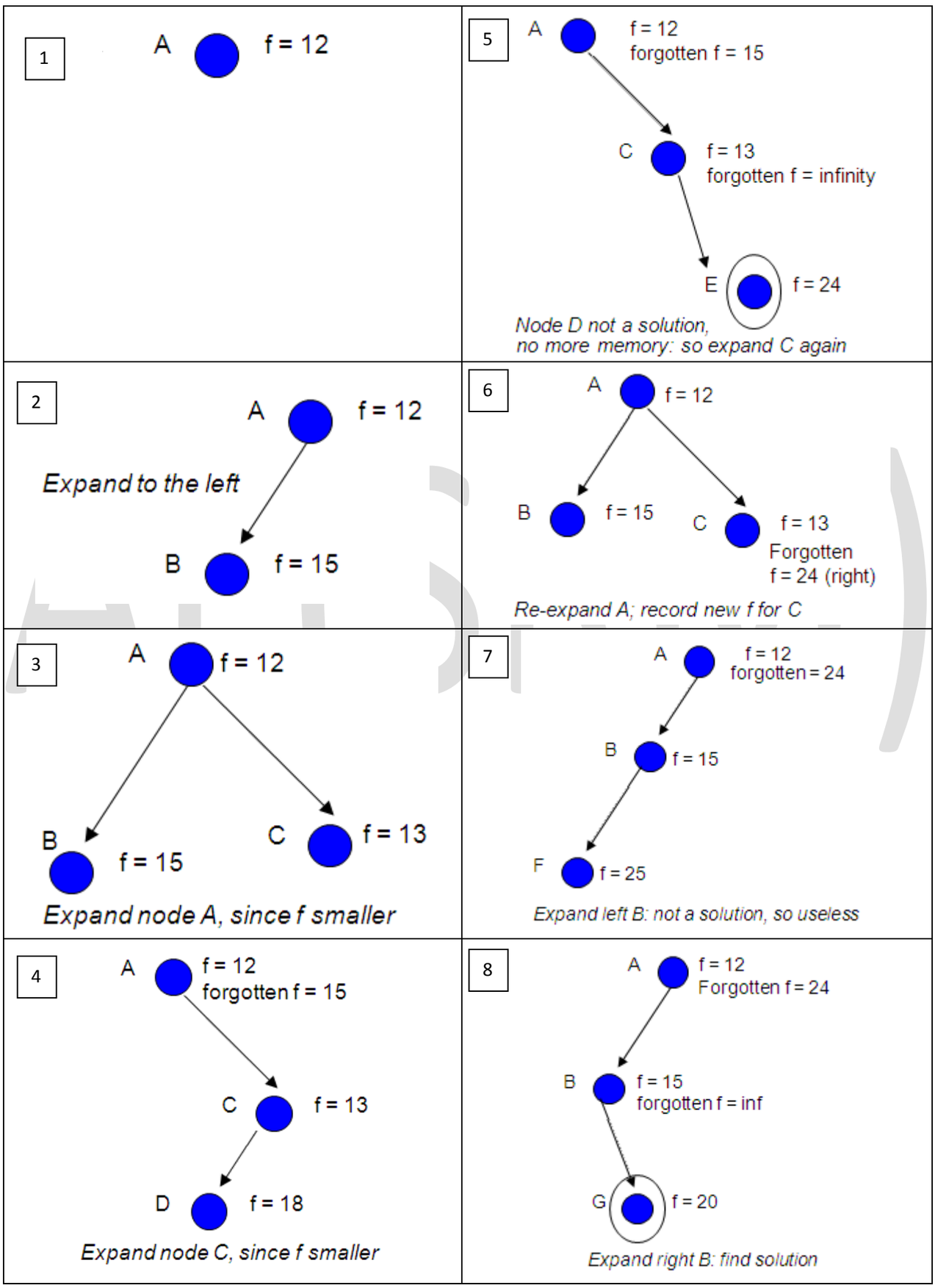
Simplified Memory bounded A* (SMA*)

- Use all available memory.
  - I.e. expand best leafs until available memory is full
  - When full, SMA* drops worst leaf node (highest *f*-value)
  - Like RFBS backup forgotten node to its parent
- What if all leafs have the same *f*-value?
  - Same node could be selected for expansion and deletion.
  - SMA* solves this by expanding *newest* best leaf and deleting *oldest* worst leaf.
- SMA* is complete if solution is reachable, optimal if optimal solution is reachable.

- SMA* will utilize whatever memory is made available to it.
- It avoids repeated states as far as its memory allow.
- It is complete if the available memory is sufficient to store the shallowest solution path.

Example

Aim: find lowest-cost goal node (G) with memory size 3.



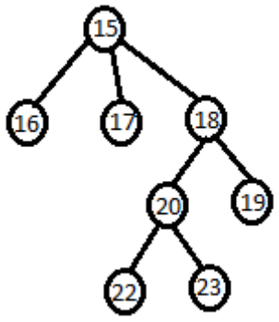
Local Search Algorithms and Optimization Problems

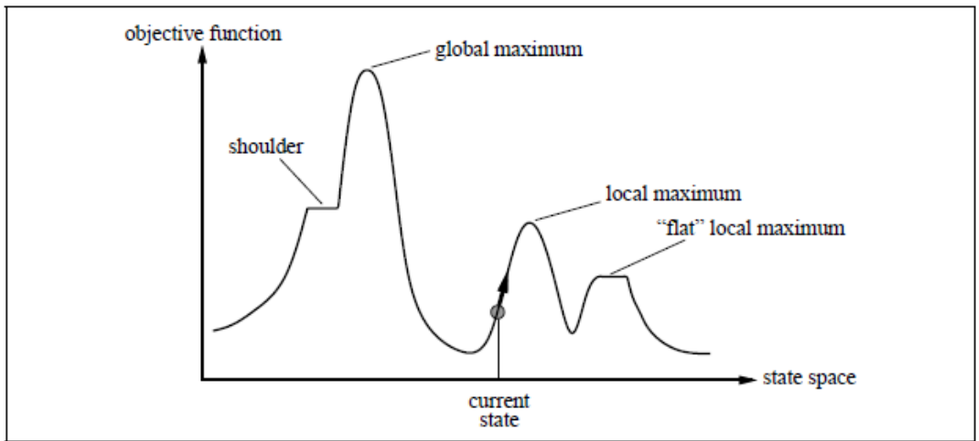
The previous sections have considered algorithms that systematically search the space. If the space is finite, they will either find a solution or report that no solution exists. Unfortunately, many search spaces are too big for systematic search and are possibly even infinite. In any reasonable time, systematic search will have failed to consider enough of the search space to give any meaningful results. This section and the next consider methods intended to work in these very large spaces. The methods do not systematically search the whole search space but they are designed to find solutions quickly on average. They do not guarantee that a solution will be found even if one exists, and so they are not able to prove that no solution exists. They are often the method of choice for applications where solutions are known to exist or are very likely to exist.

- **Previously: systematic (classical search) exploration of search space.**
  - **Path to goal is solution to problem**
- **YET, for some problems path is irrelevant.**
  - **E.g 8-queens**

Hill Climbing

- **Hill climbing** is a mathematical optimization technique which belongs to the family of local search.
- It is an iterative algorithm that starts with an arbitrary solution to a problem, then attempts to find a better solution by incrementally changing a single element of the solution.
- If the change produces a better solution, an incremental change is made to the new solution, repeating until no further improvements can be found.
- Hill climbing is good for finding a local optimum (a solution that cannot be improved by considering a neighboring configuration) but it is not guaranteed to find the best possible solution (the global optimum) out of all possible solutions (the search space).
- Global maximum is the best possible solution and the objective of this search to reach at global maximum (highest peak on hill).





A one-dimensional state-space landscape in which elevation corresponds to the objective function. The aim is to find the global maximum. Hill-climbing search modifies the current state to try to improve it, as shown by the arrow. The various topographic features are defined in the text.

Problems in Hill climbing (Drawbacks)

Local maxima:

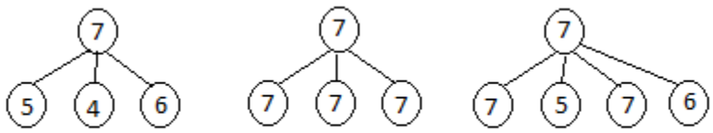
- Local maxima = no uphill step
- Algorithms on previous slide fail (not complete)
- Allow "random restart" which is complete, but might take a very long time.

Ridges:

- A "ridge" which is an area in the search that is higher than the surrounding areas, but cannot be searched in a simple move.

Plateau:

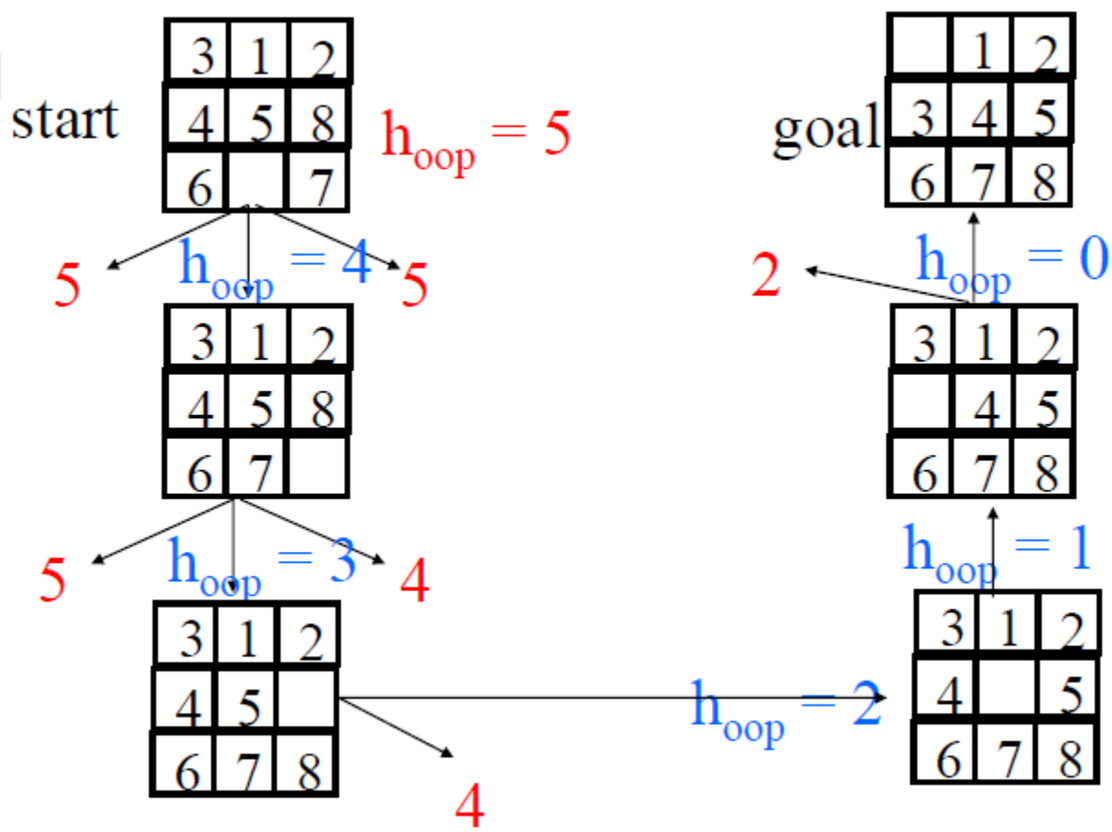
- All steps equal (flat or shoulder)
- A plateau is encountered when the search space is flat, or sufficiently flat that the value returned by the target function is indistinguishable from the value returned for nearby regions due to the precision used by the machine to represent its value.
- In such cases, the hill climber may not be able to determine in which direction it should step, and may wander in a direction that never leads to improvement.



(a) Local maxima (b) Platue (c) Ridge

a). Successor of current state are heuristically worst than the current state.
b).Successor of current state are heuristically equivalent to the current state.
c).Successor of current state are heuristically worst or heuristically equivalent than the current state.

Hill climbing Example



Simulated Annealing

- ***Annealing*:** the process by which a metal cools and freezes into a minimum-energy crystalline structure (the annealing process)
- SA exploits an analogy between annealing and the search for a minimum in a more general system.
  - Switch viewpoint from *hill-climbing* to *gradient descent*
- SA can avoid becoming trapped at local minima.
- SA uses a random search that accepts changes that decrease objective function *f*, as well as some that *increase* it.
- SA uses a control parameter T, which by analogy with the original application is known as the system *"temperature."*
- T starts out high and gradually decreases toward 0.

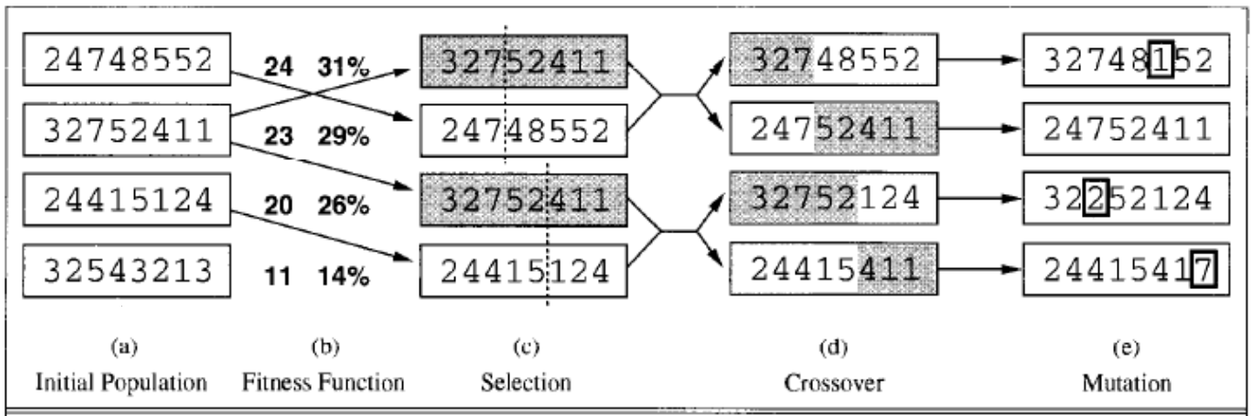
Genetic Algorithm

- An *algorithm* is a set of instructions that is repeated to solve a problem.
- A *genetic algorithm* conceptually follows steps inspired by the biological processes of evolution.
- Genetic Algorithms follow the idea of SURVIVAL OF THE FITTEST- Better and better solutions evolve from previous generations until a near optimal solution is obtained.
- Also known as *evolutionary algorithms*, genetic algorithms demonstrate self organization and adaptation similar to the way that the fittest biological organism survive and reproduce.
- A genetic algorithm is an iterative procedure that represents its candidate solutions as strings of genes called chromosomes.
- Generally applied to spaces which are too large
- Genetic Algorithms are often used to improve the performance of other AI methods such as expert systems or neural networks.
- The method learns by producing offspring that are better and better as measured by a fitness function, which is a measure of the objective to be obtained (maximum or minimum).

Simple GA

```
{
 initialize population;
evaluate population;
while TerminationCriteriaNotSatisfied
{
        select parents for reproduction;
        perform crossover  and mutation;
        repair();
evaluate population;
} }
```

Every loop called generation



The genetic algorithm. The initial population in (a) is ranked by the fitness function in (b), resulting in pairs for mating in (c). They produce offspring in (d), which are subject to mutation in (e).

Concepets

- *Population*:set of individuals each representing a possible solution to a given problem.
- *Gene*:a solution to problem represented as a set of parameters ,these parameters known as genes.
- *Chromosome*:genes joined together to form a string of values called chromosome.
- *Fitness score(value)*:every chromosome has fitness score can be inferred from the chromosome itself by using fitness function.

Stochastic operators

- *Selection* replicates the most successful solutions found in a population at a rate proportional to their relative quality
- *Recombination* (Crossover) decomposes two distinct solutions and then randomly mixes their parts to form novel solutions
- *Mutation* randomly perturbs a candidate solution

Example:

Suppose a Genetic Algorithm uses chromosomes of the form x=abcdefgh with a fixed length of eight genes. Each gene can be any digit between 0 and 9. Let the fitness of individual x be calculated as :

f(x) =(a+b)-(c+d)+(e+f)- (g+h)

And let the initial population consist of four individuals x1, ... ,x4 with the following chromosomes :

X1 = 6 5 4 1 3 5 3 2

F(x1) = (6+5)-(4+1)+(3+5)-(3+2) = 9

X2 = 8 7 1 2 6 6 0 1

F(x2) = (8+7)-(1+2)+(6+6)-(0+1) = 23

X3 = 2 3 9 2 1 2 8 5

F(x3) = (2+3)-(9+2)+(1+2)-(8+5) = -16

X4= 4 1 8 5 2 0 9 4

F(x4) = (4+1)-(8+5)+(2+0)-(9+4) = -19

The arrangement is (assume maximization)

X2 x1 x3 x4

(the fittest individual) (least fit individual)

Put the calculations in table for simplicity

Individuals	String Representation	Fitness	Arrangement Assume maximization
X1	65413532	9	X2(fittest individual)
X2	87126601	23	X1(second fittest individual)
X3	23921285	-16	X3 (third fittest individual)
X4	41852094	-19	X4 (least fit individual)

Average fitness = (9+23+ -16 + -19)/ 4 =-0.75

So Average fitness:-0.75 Best: 23 Worst: -19

Next step is to apply crossover operation

Calculating fitness function of offspring.

Offspring 1 = 8 7 1 2 3 5 3 2

F (Offspring 1) =(8+7)-(1+2)+(3+5)-(3+2) = **15**

Offspring 2 = 6 5 4 1 6 6 0 1

F (Offspring 2) =(6+5)-(4+1)+(6+6)-(0+1) = **17**

Offspring 3 = 6 5 9 2 1 2 3 2

F (Offspring 3) =(6+5)-(9+2)+(1+2)-(3+2) = **-2**

Offspring 4 = 2 3 4 1 3 5 8 5

F (Offspring 4) =(2+3)-(4+1)+(3+5)-(8+5) = **-5**

Put the calculation in table for simplicity

Individuals	String Representation	Fitness
Offspring 1	87123532	15
Offspring 2	65416601	17
Offspring 3	65921232	-2
Offspring 4	23413585	-5

Average fitness: 6.25 Best: 17 Worst: -5

So that, the overall fitness is improved, since the average is better and worst is improved.

Average fitness = (15+17+ -5 + -2)/ 4 = 6.25

Adversarial Search

Games

Multiagent environments, in which each agent needs to consider the actions of the other agents and how they affect its own welfare. The unpredictability of these other agents can introduce contingencies into agent's problem solving process. In this section we cover competitive environments, in which agent's goals are conflict, giving rise to adversarial search-often known as games.

Examine the problems that arise when we try to plan ahead in a world where other agents are planning against us. A good example is in board games. Adversarial games, while much studied in AI, are a small part of game theory in economics.

Search versus Games.

Search – no adversary	Games – adversary
1. Solution is (heuristic) method for finding goal. 2. Heuristic techniques can find optimal solution 3. Evaluation function: estimate of cost from start to goal through given node 4. Examples: path planning, scheduling activities	1. Solution is strategy (strategy specifies move for every possible opponent reply). 2. Optimality depends on opponent. 3. Time limits force an approximate solution. 4. Evaluation function: evaluate "goodness" of game position 5. Examples: chess, checkers, Othello, backgammon.

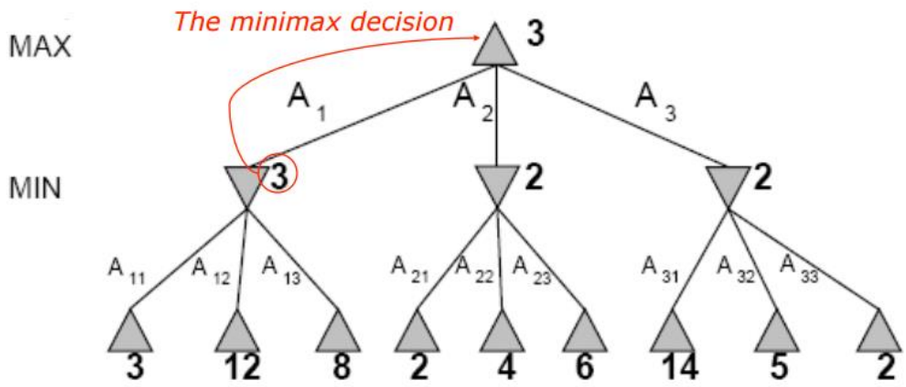
Game Setup

- Two players: MAX and MIN
- MAX moves first and they take turns until the game is over
  Winner gets award, loser gets penalty.
- Games as search:
  - Initial state: e.g. board configuration of chess
  - Successor function: list of (move, state) pairs specifying legal moves.
  - Terminal test: Is the game finished?
  - Utility function: Gives numerical value of terminal states. E.g. win (+1), lose (-1) and draw (0) in tic-tac-toe or chess MAX uses search tree to determine next move.

Minimax strategy

The Min-Max algorithm is applied in two player games, such as tic-tac-toe, checkers, chess, and so on. All these games have at least one thing in common, they are logic games. This means that they can be described by a set of rules and premises. With them, it is possible to know from a given point in the game, what are the next available moves. So they also share other characteristic, they are 'full information games'. Each player knows everything about the possible moves of the adversary.

There are two players involved, MAX and MIN. A search tree is generated, depth-first, starting with the current game position up to the end game position. Then, the final game position is evaluated from MAX's point of view. Afterwards, the inner node values of the tree are filled bottom-up with the evaluated values. The nodes that belong to the MAX player receive the maximum value of it's children. The nodes for the MIN player will select the minimum value of it's children.



The values represent how good a game move is. So the MAX player will try to select the move with highest value in the end. But the MIN player also has something to say about it and he will try to select the moves that are better to him, thus minimizing MAX's outcome.

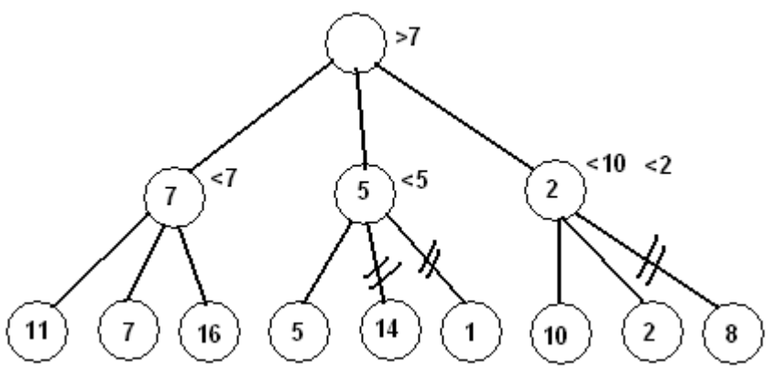
(For problems see class notes)

Alpha-Beta Pruning

Alpha–beta pruning is a search algorithm that seeks to decrease the number of nodes that are evaluated by the algorithm its search tree. It is an adversarial search algorithm used commonly for machine playing of two-player games (Tic-tac-toe, Chess etc.).

α: Best already explored option along path to the root for miximizer.

β: Best already explored option along path to the root for minimizer.



sureshmestry82@gmail.com

Ch.4 knowledge and Reasoning

Knowledge is a general term.
An answer to the question, *"how to represent knowledge"*, requires an analysis to distinguish between knowledge "how" and knowledge "that".

- Knowing *"how to do something".*
  e.g. "how to drive a car" is Procedural knowledge.
- Knowing *"that something is true or false".*
  e.g. "that is the speed limit for a car on a motorway" is Declarative knowledge.

Knowledge and Representation are distinct entities that play a central but distinguishable role in intelligent system.

- Knowledge is a description of the world.
  – It determines a system's competence by what it knows.
- Representation is the way knowledge is encoded.
  – It defines the performance of a system in doing something.

- Different types of knowledge require different kinds of representation.
  The Knowledge Representation ***models/mechanisms*** are often based on:
  ◊ *Logic* ◊ *Rules*
  ◊ *Frames* ◊ *Semantic Net*

- Different types of knowledge require different kinds of ***reasoning***.

Knowledge-based agent:

- A knowledge-based agent includes a knowledge base and an inference system.
- A knowledge base is a set of representations of facts of the world.
- Each individual representation is called a **sentence**.
- The sentences are expressed in a knowledge **representation language**.
- The agent operates as follows:

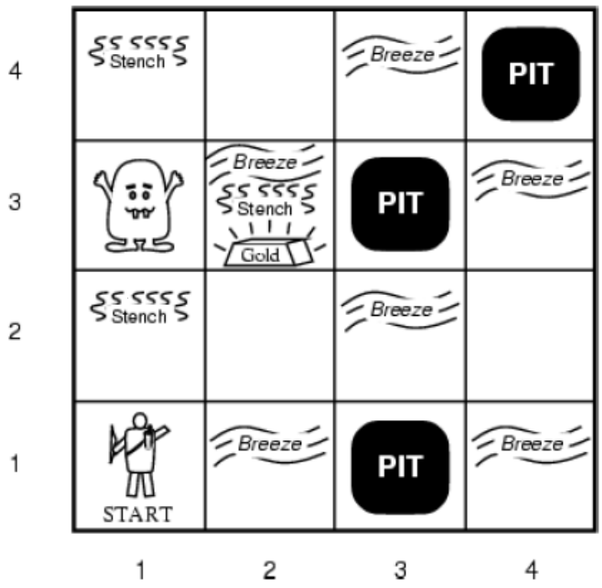
  1. It TELLs the knowledge base what it perceives.
  2. It ASKs the knowledge base what action it should perform.
  3. It performs the chosen action.

The WUMPUS WORLD Environment

• The Wumpus computer game
• The agent explores a cave consisting of rooms connected by passageways.
• Lurking somewhere in the cave is the Wumpus, a beast that eats any agent that enters its room.
• Some rooms contain bottomless pits that trap any agent that wanders into the room.
• Occasionally, there is a heap of gold in a room.
• The goal is to collect the gold and exit the world without being eaten.

Typical Wumpus World:

- The agent always starts in the field [1, 1].
- The task of the agent is to find the gold,
  return to the field [1, 1] and climb out of the cave.



WUMPUS WORLD PEAS description

Performance:
Points are awarded and/or deducted:
- find gold: +1000
- death by Wumpus: -1000
- death by endless pit: -1000
- each action: -1
- picking up the arrow: -10

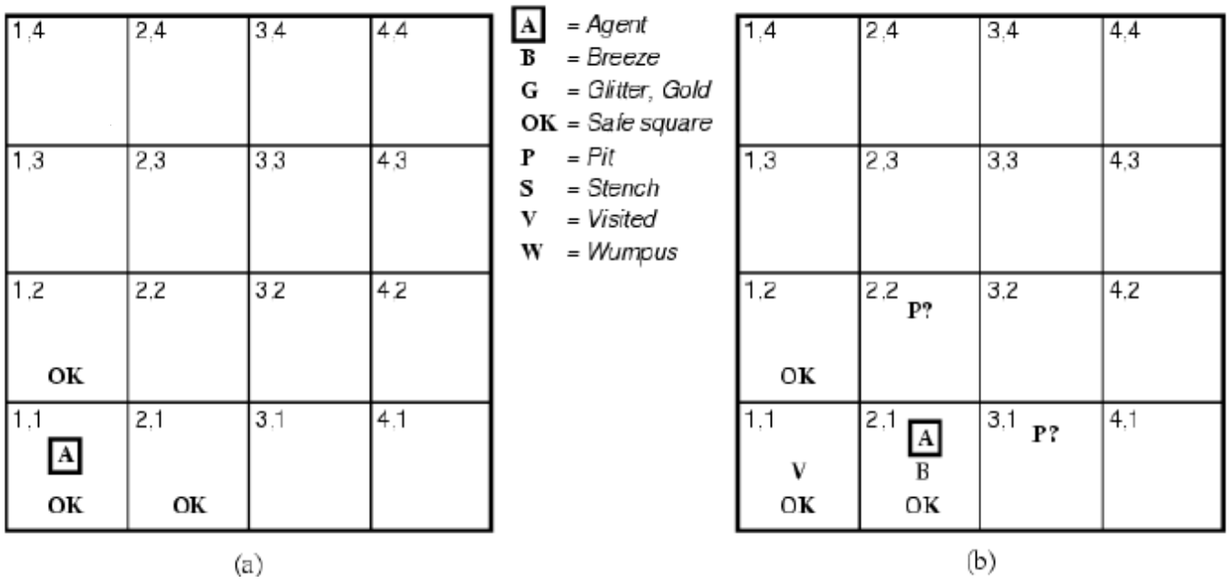
Environment:
- 4x4 grid.
- Agent starts at 1,1 (lower left).
- Agent starts facing to the right.
- The gold and the Wumpus are placed at random locations in the grid. (can't be in start room).
- Each room other than starting room can be a bottomless pit with probability 0.2

Actuators:
- Turn left 90°
- Turn right 90°
- Move forward into room straight ahead.
  - blocked by walls.
  - Eaten by a live Wumpus immediately.
  - Fall into pit immediately.
- Grab an object in current room.
- Shoot arrow in straight line (only allowed once).

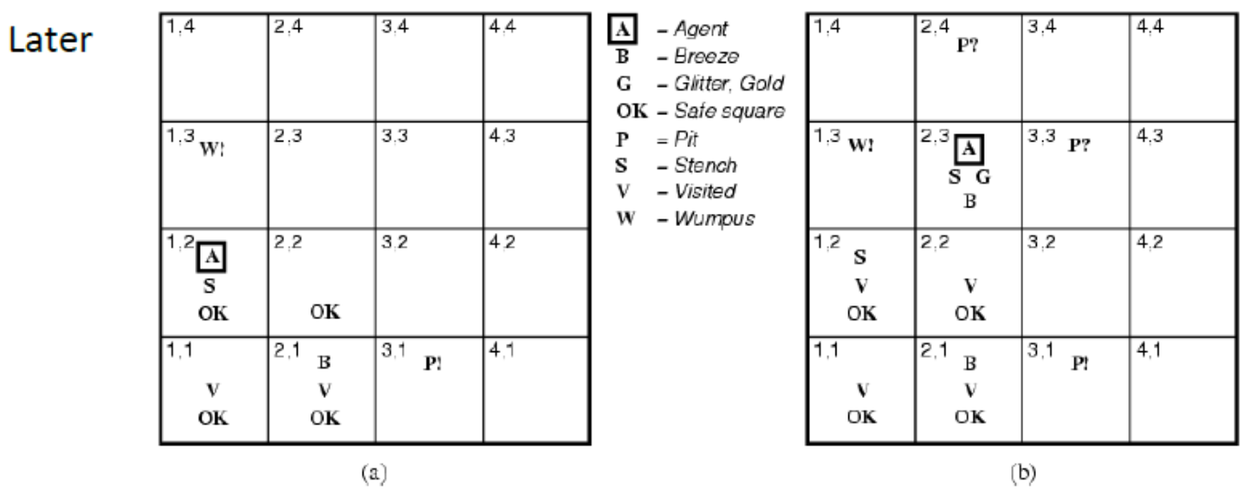
Sensor:
- Agent can smell, detect a breeze, see gold glitter, detect a bump (into a wall) and hear the Wumpus scream when it is killed.
- Sensors:
  - *Stench*: a neighboring room holds the Wumpus.
  - *Breeze*: a neighboring room holds a pit.
  - Glitter: The room the agent is in has some gold.
  - *Bump*: The agent just banged into a wall.
  - *Scream*: The Wumpus just died (shot with arrow).

Based on sensors, agent gets following percept: **Percept (Breeze, Stench, Glitter, Bump, Scream).**

The Wumpus agent's first step



Later



Percept Sequence:
Percept (1,1)=(None, None, None, None, None,)
Percept (2,1)=(**Breeze**, None, None, None, None,)
Percept (1,1)=(None, None, None, None, None,)
Percept (1,2)=(None, **Stench**, None, None, None,)
Percept (2,2)=(None, None, None, None, None,)
Percept (3,2)=(**Breeze**, None, None, None, None,)
Percept (2,3)=(None, None, **Glitter**, None, None,)

Propositional Logic (PL)

- A simple language that is useful for showing key ideas and definitions
- User defines a set of propositional **symbols**, like *P* and *Q*. User define the semantics of each of these symbols. For example,
  - P means "It is hot"
  - Q means "It is humid"
  - R means "It is raining"
- A **sentence** (also called a formula or well-formed formula or wff) is defined as:
  1. A symbol
  2. If S is a sentence, then ~S is a sentence, where "~" is the "not" logical operator

3. If S and T are sentences, then (S v T), (S ^ T), (S => T), and (S <=> T) are sentences, where the four logical connectives correspond to "or," "and," "implies," and "if and only if," respectively
4. A finite number of applications of (1)-(3)

- Examples of PL sentences:
  - (P ^ Q) => R (here meaning "If it is hot and humid, then it is raining")
  - Q => P (here meaning "If it is humid, then it is hot")
  - Q (here meaning "It is humid.")

- Given the truth values of all of the constituent symbols in a sentence, that sentence can be "evaluated" to determine its truth value (True or False). This is called an **interpretation** of the sentence.
- A **model** is an interpretation (i.e., an assignment of truth values to symbols) of a set of sentences such that each sentence is True. A model is just a formal mathematical structure that "stands in" for the world.
- A **valid** sentence (also called a **tautology**) is a sentence that is True under *all* interpretations. Hence, no matter what the world is actually like or what the semantics is, the sentence is True. For example "It's raining or it's not raining."
- An **inconsistent** sentence (also called **unsatisfiable** or a **contradiction**) is a sentence that is False under *all* interpretations. Hence the world is never like what it describes. For example, "It's raining and it's not raining."

Examples:

Q.1) consider following set of facts.

I. Rani is hungry.

II. If rani is hungry she barks.

III. If rani is barking then raja is angry.

Convert into proposition logic statements.

Solution:

step1: we can use following propositional symbols

P: Rani is hungry

Q: Rani is Barking

R: Raja is Angry.

Step2: The propositional logic statements are,

I. P

II. P=>Q

III. Q=>R

Predicate Logic:

First-Order Logic or first order Predicate Logic (FOL or FOPL) Syntax

- User defines these primitives:
  - **Constant symbols** (i.e., the "individuals" in the world) E.g., Mary, 3
  - **Function symbols** (mapping individuals to individuals)
    E.g., father-of(Mary) = John, color-of(Sky) = Blue
  - **Predicate symbols** (mapping from individuals to truth values) E.g., greater(5,3), green(Grass), color(Grass, Green)

- FOL supplies these primitives:

  - **Variable symbols**. E.g., x, y
  - **Connectives**. Same as in PL: not (~), and (^), or (v), implies (=>), if and only if (<=>)
  - **Quantifiers**: Universal (∀) and Existential (∃)
    - Universal quantification corresponds to conjunction ("and") in that (Ax):P(x) means that P holds for all values of x in the domain associated with that variable.

      E.g., (∀x) dolphin(x) => mammal(x)
    - Existential quantification corresponds to disjunction ("or") in that (∃x)P(x) means that P holds for some value of x in the domain associated with that variable.

      E.g., (∃x) mammal(x) ^ lays-eggs(x)
    - Universal quantifiers usually used with "implies" to form "if-then rules."

      E.g., (∀x) cs540-student(x) => smart(x) means "All cs540 students are smart.

      "You rarely use universal quantification to make blanket statements about every individual in the world: (∀x) cs540-student(x) ^ smart(x) meaning that everyone in the world is a cs540 student and is smart.
    - Existential quantifiers usually used with "and" to specify a list of properties or facts about an individual.

      E.g., (∃x) cs540-student(x) ^ smart(x) means "there is a cs540 student who is smart."

      A common mistake is to represent this English sentence as the FOL sentence:

      (∃x) cs540-student(x) => smart(x) But consider what happens when there is a person who is NOT a cs540-student.

    - Switching the order of universal quantifiers does not change the meaning: (∀x)(∀y)P(x,y) is logically equivalent to (∀y)(∀x)P(x,y). Similarly, you can switch the order of existential quantifiers.
    - Switching the order of universals and existential *does* change meaning:

      - Everyone likes someone: (∀x):(∃y):likes(x,y)
      - Someone is liked by everyone: (∃y)(∀x)likes(x,y)

Translating English to First Order Logic (FOL)
Some examples:

- Every gardener likes the sun.
  (∀x): gardener(x) => likes(x,Sun)

- You can fool some of the people all of the time.
  (∃x): (person(x) ^ (At)(time(t) => can-fool(x,t)))

- You can fool all of the people some of the time.
  (∀x): (person(x) => (Et) (time(t) ^ can-fool(x,t)))

- All purple mushrooms are poisonous.
  (∀x): (mushroom(x) ^ purple(x)) => poisonous(x)

- No purple mushroom is poisonous.
  ~(∃x): purple(x) ^ mushroom(x) ^ poisonous(x)
  Or, equivalently,
  (∀x): (mushroom(x) ^ purple(x)) => ~poisonous(x)

- There are exactly two purple mushrooms.
  (∃x):(∃y): mushroom(x) ^ purple(x) ^ mushroom(y) ^ purple(y) ^ ~(x=y) ^ (∀z) (mushroom(z) ^ purple(z)) => ((x=z) v (y=z))

- Deb is not tall.
  ~tall(Deb)

(For more examples, see class notes)

Resolution

It is a mechanism to infer some fact or to reach to some conclusion using logic. In resolution to prove some fact are true, we resolve that the negation of that fact is not true.
Following steps are in resolution;

1) Convert English statements to either propositional logic or predicate logic statements.
2) Convert logical statement to CNF (Conjunctive Normal Form).
3) Negate the conclusion.
4) Resolve the negation of conclusion is not true using resolution tree.

Conjunctive Normal Form (CNF)

Following are the steps to convert logic into CNF.
In CNF the fact will be connected only with 'V' (conjunctive).If we eliminate implication (=>), Universal (∀), and Existential (∃), then the statement is converted to CNF.

1) Eliminate implication '=>'

- a=>b = ~a V b

- (a ∧b)=>c =~(a ∧b) V c

2) Eliminate '∧'

- a∧b=a
  b

- a∧b∧~c=a
  b
  ~c

- a∧(bVc) = a
  bVc

- aV(b∧c) = aVb
  aVc

3) Eliminate ' ∃ '

We can eliminate ∃ by substituting for the variable a reference to a function that produces a desired value.

Eg.There is a teacher

∃x: Teacher(x) ,then by eliminating '∃' we get,

"Teacher (s1)" where s1 is a function that somehow produces a value that satisfies the predicate teacher.

4) **Eliminate '∀'**

To eliminate '∀' , convert the fact into prefix normal form in which all the universal quantifiers are at the beginning of formula.

Eg. All students are intelligent.

∀x: Student(x)->Intelligent(x)

After eliminating ∀x we get,

Student(x)->Intelligent(x).

Problems based on Resolution. (Refer class notes)

Q.1) consider following set of facts.

I. **Rani is hungry.**
II. **If rani is hungry she barks.**
III. **If rani is barking then raja is angry.**

Convert into proposition logic statements.

And prove that 'Raja is Angry' by resolution

Sol: step1: we can use following propositional symbols

P: Rani is hungry

Q: Rani is Barking

R: Raja is Angry.

Step2: The propositional logic statements are,

I. P
II. P=>Q
III. Q=>R

Step 3: Convert logic into CNF

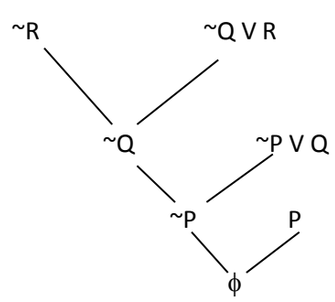
I. P
II. ~P V Q
III. ~Q V R

Step 4: Negate the conclusion

Raja is angry: R

After Negation raja is not Angry i.e. **~R**

Step 5: se resolution tree



Thus ,we get empty string and we can conclude that 'Raja is angry'

Q2.Consider following facts

1. **It is Humid**
2. **If it is Humid then it is hot**
3. **It it is hot and humid then it will rain.**
   Prove that "It will Rain"

Solution: Sol: step1: we can use following propositional symbols

P: It is Humid
Q: It is Hot
R: It is raining.

Step2: The propositional logic statements are,

I. P
II. P=>Q
III. (Q ∧ P)=>R

Step 3: Convert logic into CNF

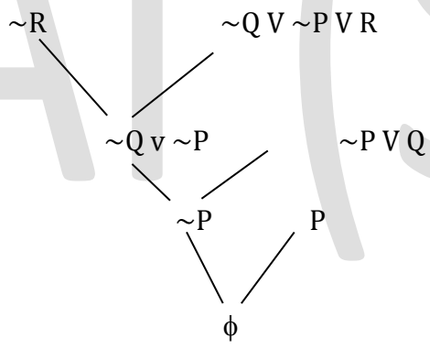
I. P
II. ~P V Q
III. ~Q V~P V R

Step 4: Negate the conclusion

It is raining: R

After Negation It is not raining i.e. **~R**

Step 5: use resolution tree



Thus, we get empty string and we can conclude that 'It will Rain'

Q.3 Consider following facts

1. **If maid stole the jewelry then butler was not guilty.**
2. **Either maid stole jewelry or she milk the cow**
3. **If maid milked the cow then butler got the cream.**
4. **Therefore if butler was guilty then he got the cream.**

Prove that the conclusion 'step 4' is valid using resolution.

Solution: step1: we can use following propositional symbols

P: maid stole the jewelry
Q: butler was guilty
R: maid milked the cow
T: Butler got cream

Step2: The propositional logic statements are,

I. P=>~Q
II. P V R
III. R => S
IV. Q => S

Step 3: Convert logic into CNF

I. ~P V ~Q
II. P V R
III. ~R V S
IV. ~Q V S

Step 4: Negate the conclusion

~ (Q =>S) (*After CNF :~Q V S*)

After Negation: Q *(Before negation, conclusion must b in CNF)*
~S (Here after negation we get two separate results)

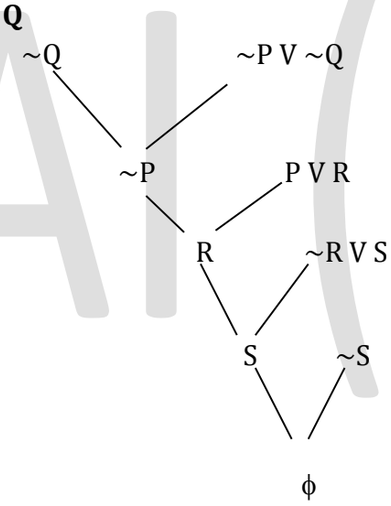
Step 5: use resolution tree

Note: (Only for understanding)

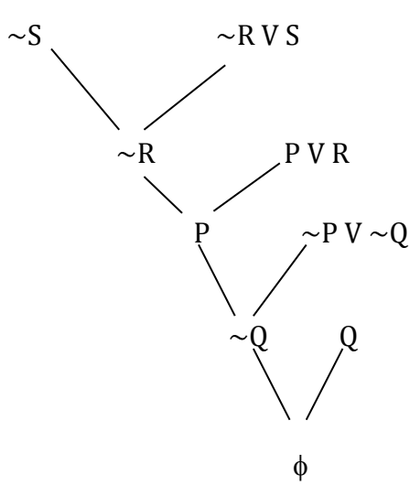
1. If conclusion is in more than one part then start with anyone.
2. If conclusion is in more than one part then all of them should be used in Resolution tree before getting a null set.
3. It is not necessary to exhaust all the facts.
4. A fact once used cannot be used again

Here we have to draw two trees for both conclusion i.e for Q and ~S

Using Q



Using ~S



Thus, we get empty string and we can conclude that 'If butler was guilty then he got the cream'.

Forward and Backward Chaining

Forward Chaining

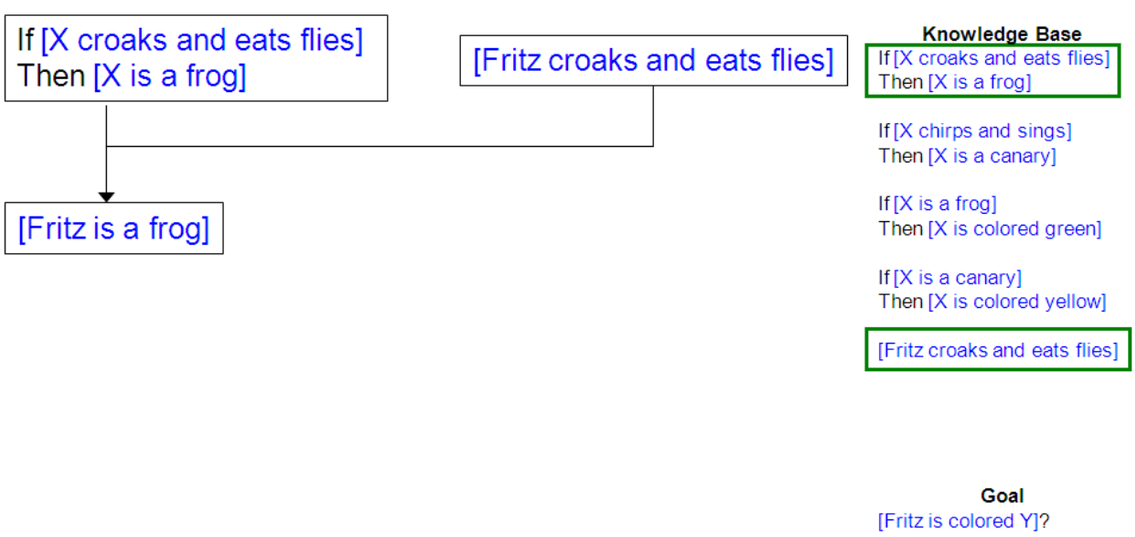
- Forward chaining is a *data driven* method of deriving a particular goal from a given knowledge base and set of inference rules
- Inference rules are applied by matching facts to the antecedents of consequence relations in the knowledge base
- The application of inference rules results in new knowledge (from the consequents of the relations matched), which is then added to the knowledge base.
- It uses Modus Ponen rule.
- Inference rules are successively applied to elements of the knowledge base until the goal is reached
- A search control method is needed to select which element(s) of the knowledge base to apply the inference rule to at any point in the deduction

Example:

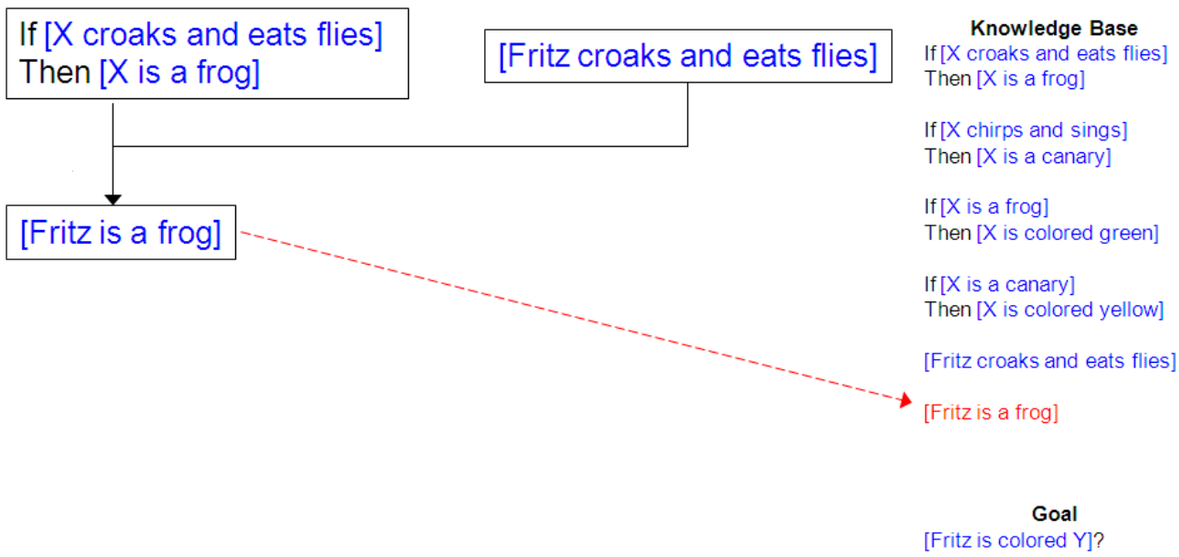
- Knowledge Base:
  - If [X croaks and eats flies] Then [X is a frog]
  - If [X chirps and sings] Then [X is a canary]
  - If [X is a frog] Then [X is colored green]
  - If [X is a canary] Then [X is colored yellow]
  - [Fritz croaks and eats flies]
- Goal:
  - [Fritz is colored Y]?

Solution:

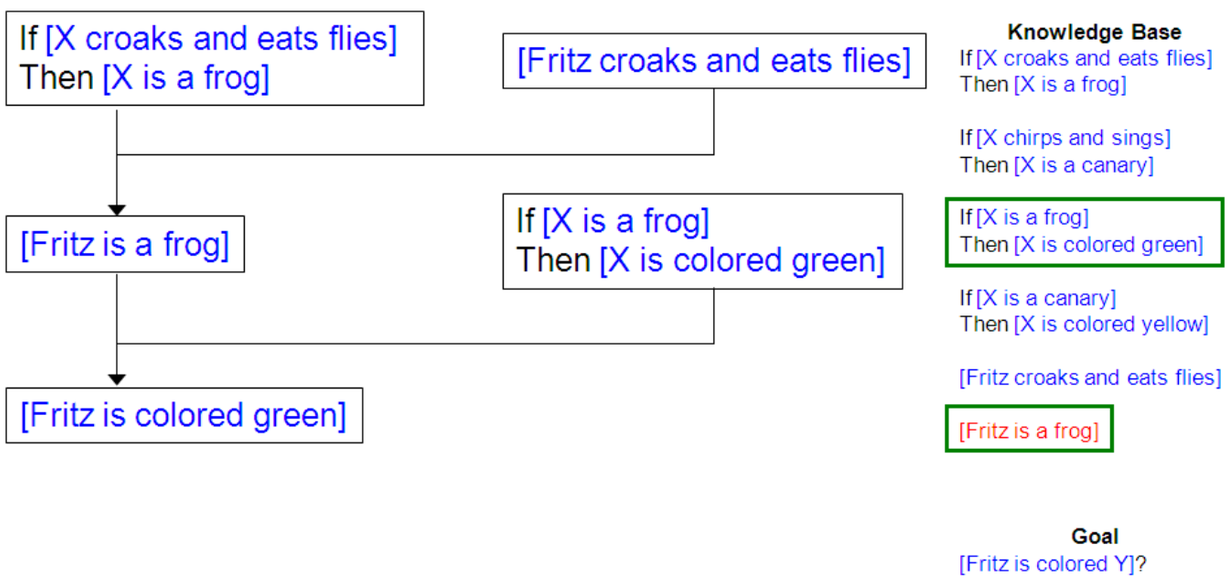
Step 1:



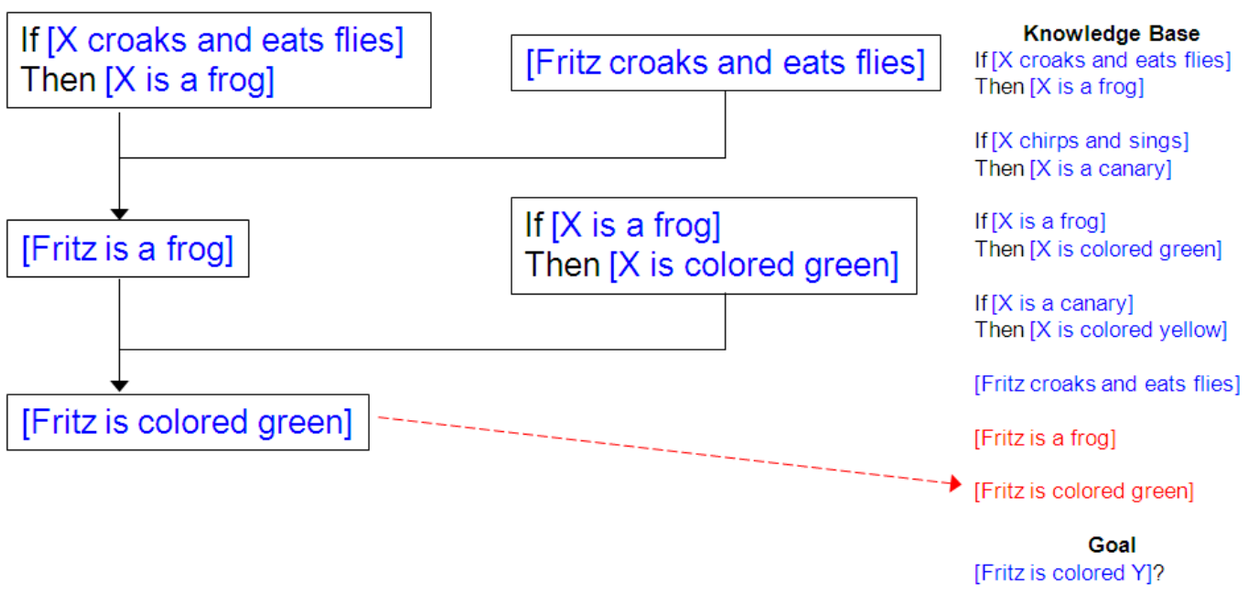
Step 2: New rule "Fritz is a frog" is added to knowledge base

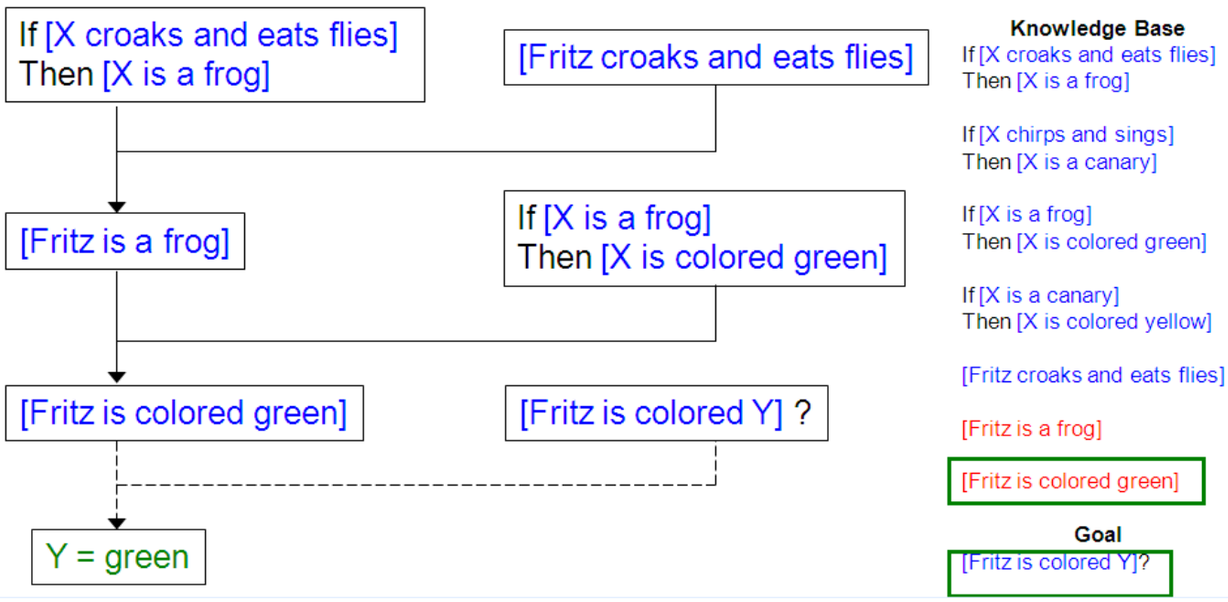


Step 3: Again applying inference rule between "If [X is a frog] Then [X is colored green]" and "[Fritz is a frog]".



Step 4: New rule is added to knowledge base. Every resulted rule must compare with goal.

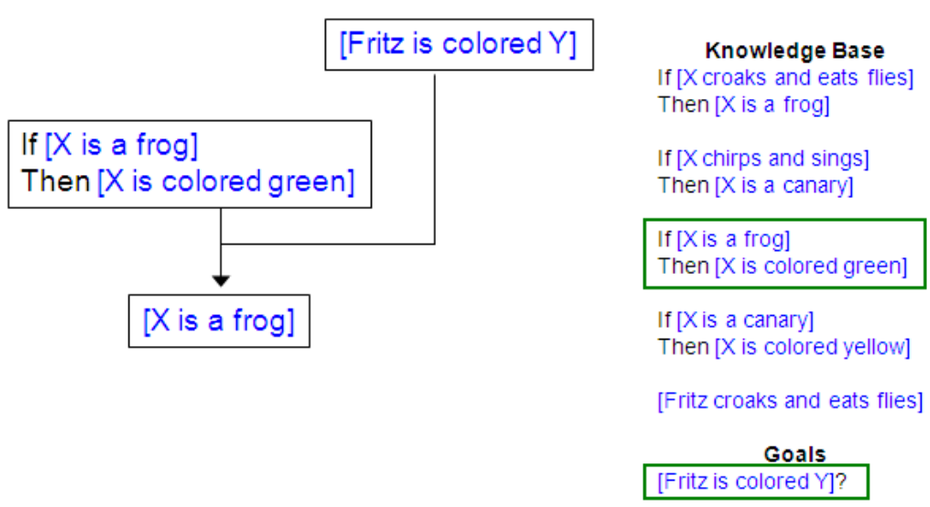




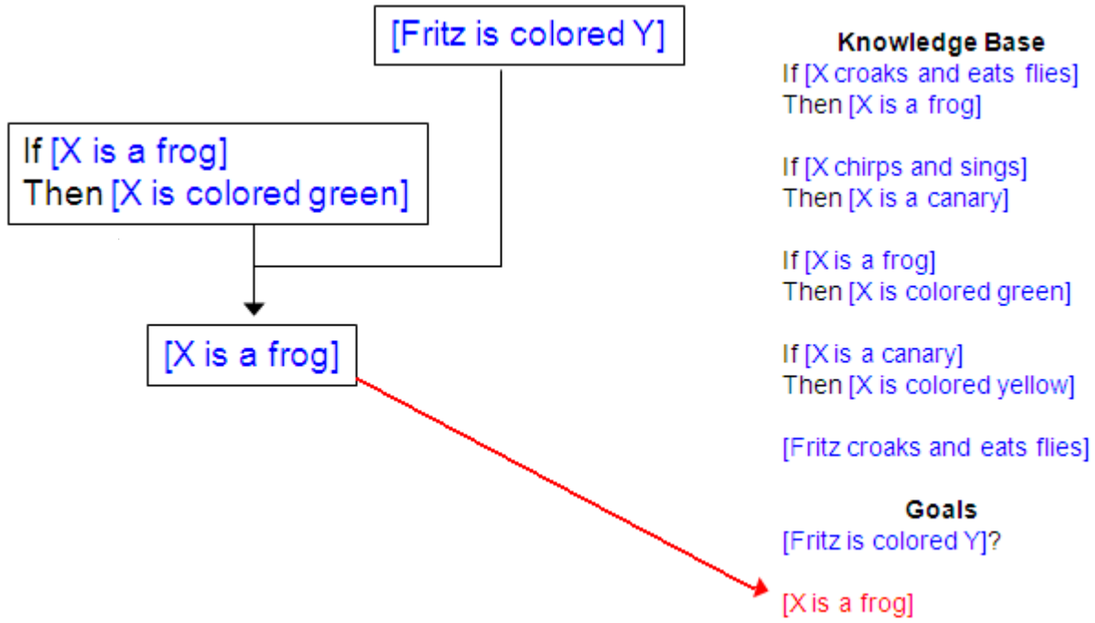
Backward Chaining

- Backward chaining is a *goal driven* method of deriving a particular goal from a given knowledge base and set of inference rules
- Inference rules are applied by matching the goal of the search to the consequents of the relations stored in the knowledge base
- When such a relation is found, the antecedent of the relation is added to the list of goals (and not into the knowledge base, as is done in forward chaining)
- It uses modus tolen inference rule.
- Search proceeds in this manner until a goal can be matched against a fact in the knowledge base
  - Remember: facts are simply consequence relations with empty antecedents, so this is like adding the 'empty goal' to the list of goals
- As with forward chaining, a search control method is needed to select which goals will be matched against which consequence relations from the knowledge base

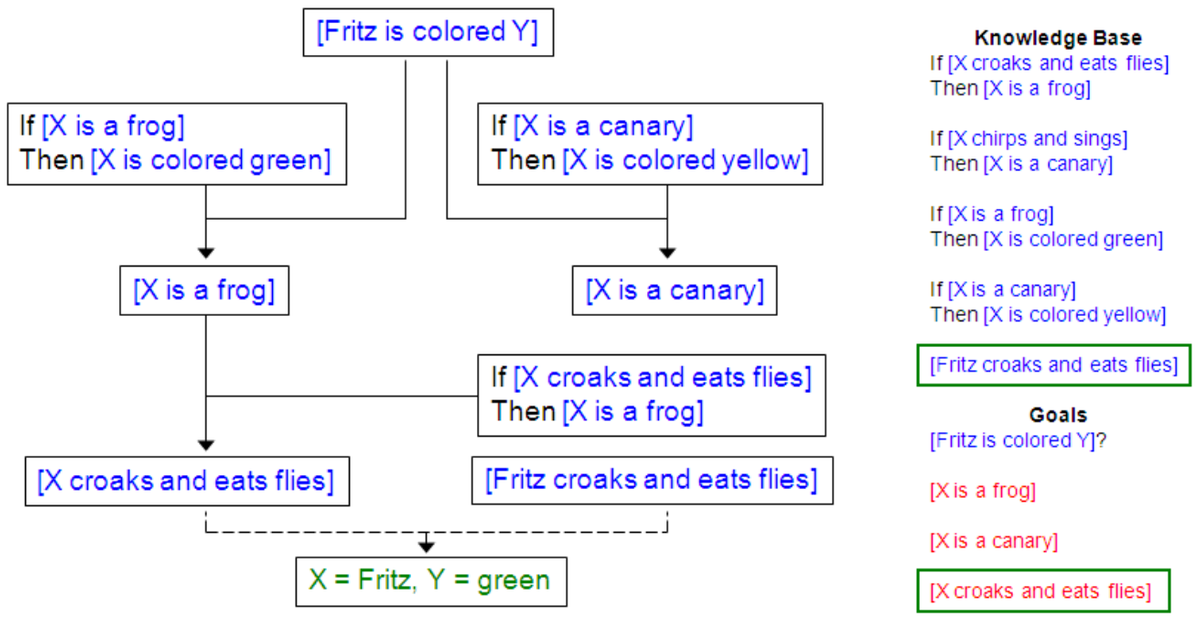
Example: Step 1: Apply inference rule between knowledge and goal.



Step 2: Unlike forward chaining ,new rule get added to goal.



Step 3: final step



Uncertain Knowledge and Reasoning

Uncertainty

Many times in complex world theory of agent and events in environment are contradicted to each other, and this result in reduction of performance measure. E.g. "let agent's job is to leave the passenger on time, before the flight departs. But agent knows the problems it can face during journey. Means Traffic, flat tire, or accident. In these cases agent cannot give its full performance. "This is called as uncertainty.

Probability:

- Objective probability
  - Averages over repeated experiments of random events
    - E.g. estimate P (Rain) from historical observation
  - Makes assertions about future experiments
  - New evidence changes the reference class

- Subjective / Bayesian probability
  - Degrees of belief about unobserved event: state of knowledge

Probability Basics

Priori probability

The prior probability of an event is the probability of the event computed before the collection of new data. One begins with a prior probability of an event and revises it in the light of new data. For example, if 0.01 of a population has schizophrenia then the probability that a person drawn at random would have schizophrenia is 0.01. This is the prior probability. If you then learn that that there score on a personality test suggests the person is schizophrenic, you would adjust your probability accordingly. The adjusted probability is the posterior probability.

Bayes' Theorem:

Bayes' theorem considers both the prior probability of an event and the diagnostic value of a test to determine the posterior probability of the event. The theorem is shown below:

$$P(D \mid T) = \frac{P(T \mid D)P(D)}{P(T \mid D)P(D) + P(T \mid D')P(D')}$$

where P(D|T) is the posterior probability of Diagnosis D given Test result T, P(T|D) is the conditional probability of T given D, P(D) is the prior probability of D, P(T|D') is the conditional probability of T given not D, and P(D') is the probability of not D'.

Conditional Probability Formulae

Definition of conditional probability
P(a|b) =P(a ^ b)/P(b) if P(b) 6= 0
– Probability of observing event a given evidence (knowledge / observation) of event b.

Bayesian Networks (IMP)

- A simple, graphical notation for conditional independence assertions
  - Compact specification of full joint distributions
- Syntax
  - a set of nodes, one per variable Xl
  - a directed, acyclic graph (link _ “directly influences”)
  - a conditional probability distribution (CPD) for each node given its parents.
    P(Xi|Parents(Xi))
- Simplest case: CPD is a conditional probability table (CPT)
  - Giving distribution over Xi for each combination of parent values.

<u>Example of Bayesian network</u>

<u>Example 1</u>

Suppose that there are two events which could cause grass to be wet: either the sprinkler is on or it's raining. Also, suppose that the rain has a direct effect on the use of the sprinkler (namely that when it rains, the sprinkler is usually not turned on). Then the situation can be modeled with a Bayesian network (shown). All three variables have two possible values, T (for true) and F (for false).

The joint probability function is:

$P(G,S,R) = P(G \mid S,R)P(S \mid R)P(R)$

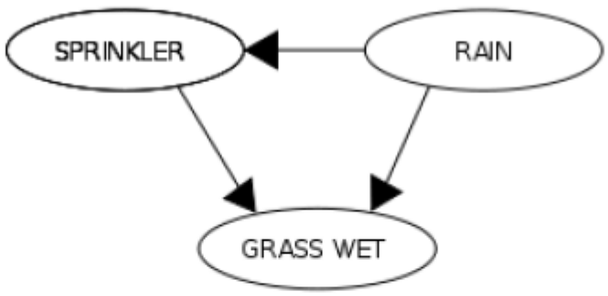
where the names of the variables have been abbreviated to *G = Grass wet*, *S = Sprinkler*, and *R = Rain*.

The model can answer questions like "What is the probability that it is raining, given the grass is wet?" by using the conditional probability formula and summing over all nuisance variables:

$$\mathrm{P}(R = T \mid G = T) = \frac{\mathrm{P}(G = T, R = T)}{\mathrm{P}(G = T)} = \frac{\sum_{S \in \{T,F\}} \mathrm{P}(G = T, S, R = T)}{\sum_{S,R \in \{T,F\}} \mathrm{P}(G = T, S, R)}$$

$$= \frac{(0.99 \times 0.01 \times 0.2 = 0.00198_{TTT}) + (0.8 \times 0.99 \times 0.2 = 0.1584_{TFT})}{0.00198_{TTT} + 0.288_{TTF} + 0.1584_{TFT} + 0_{TFF}} \approx 35.77\%.$$

RAIN	SPRINKLER T	SPRINKLER F
F	0.4	0.6
T	0.01	0.99



RAIN T	RAIN F
0.2	0.8

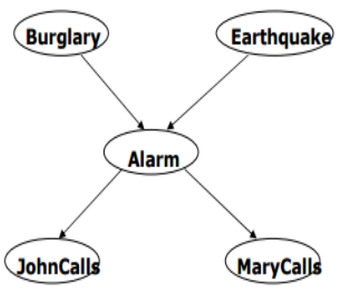
SPRINKLER	RAIN	GRASS WET T	GRASS WET F
F	F	0.0	1.0
F	T	0.8	0.2
T	F	0.9	0.1
T	T	0.99	0.01

Example 2:

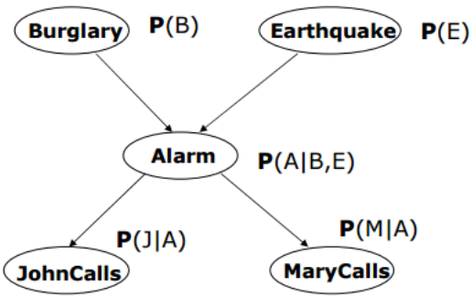
• Assume your house has an alarm system against burglary. You live in the seismically active area and the alarm system can get occasionally set off by an earthquake. You have two neighbors, Mary and John, who do not know each other. If they hear the alarm they call you, but this is not guaranteed.

• We want to represent the probability distribution of events: – Burglary, Earthquake, Alarm, Mary calls and John calls

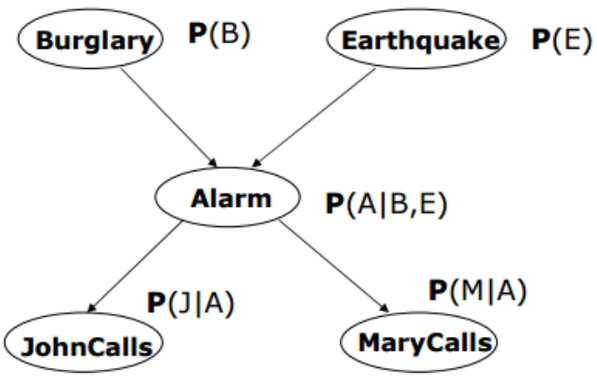
Causal relations



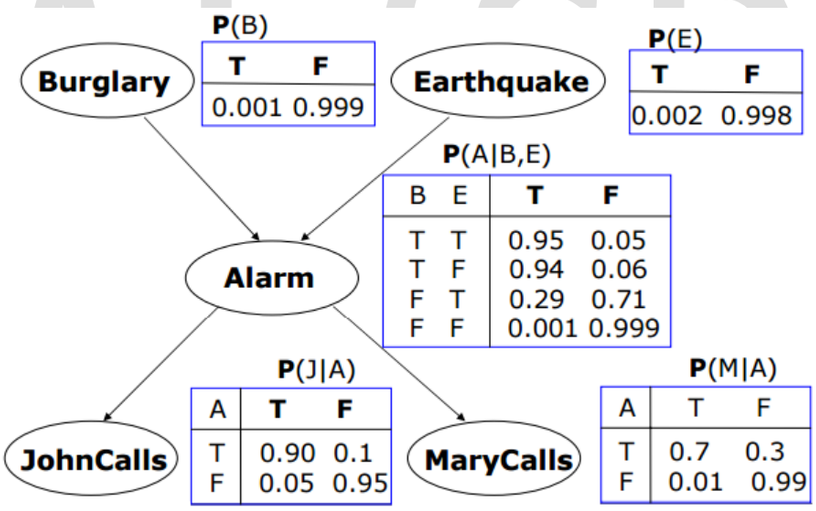
1. **Directed acyclic graph**
   • Nodes = random variables Burglary, Earthquake, Alarm, Mary calls and John calls
   • Links = direct (causal) dependencies between variables. The chance of Alarm is influenced by Earthquake, The chance of John calling is affected by the Alarm



2. **Local conditional distributions**
   - relate variables and their parents



3. In the BBN the full joint distribution is expressed using a set of local conditional distributions



Ch.5 Planning and Learning

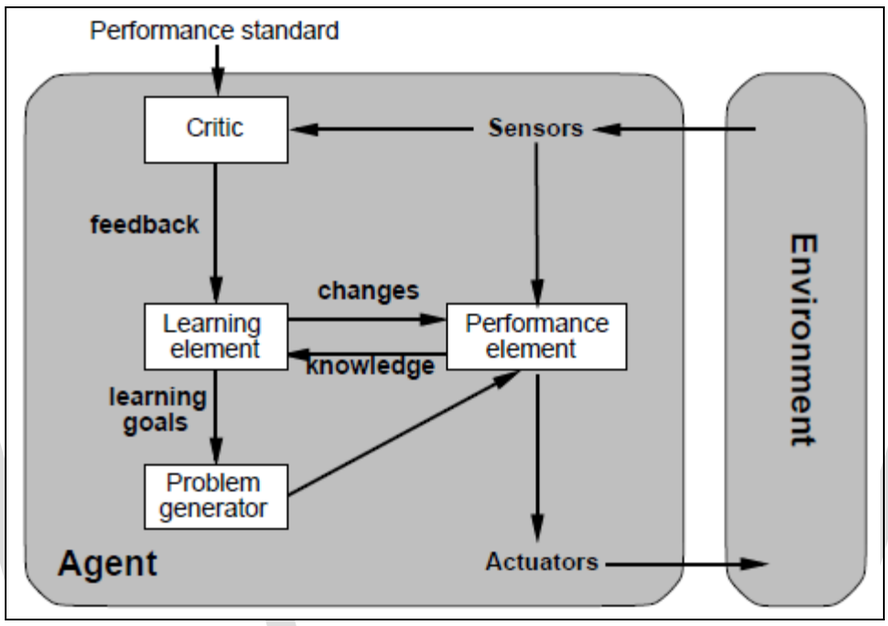
Planning

Learning

Learning denotes changes in a system that enables the system to do the same task more efficiently next time.
Learning is an important feature of Intelligence".
Learning is a branch of artificial intelligence, is a scientific discipline concerned with the design and development of algorithms that allow computers to evolve behaviors based on empirical data, such as from sensor data or databases.

General model of Learning Agent:

Learning agents



- Learning has an advantage that it allows the agents to initially operate in unknown environments and to become more competent than its initial knowledge alone might allow.
- The most important distinction is between the "learning element", which is responsible for making improvements, and the "performance element", which is responsible for selecting external actions.
- The learning element uses feedback from the "critic" on how the agent is doing and determines how the performance element should be modified to do better in the future.
- The performance element is what we have previously considered to be the entire agent: it takes in percepts and decides on actions.
- The last component of the learning agent is the "problem generator".
- It is responsible for suggesting actions that will lead to new and informative experiences.

Forms of Learning

Supervised Learning

- An agent tries to find a function that matches examples from a sample set; each example provides an input together with the correct output
- A teacher provides feedback on the outcome, the teacher can be an outside entity, or part of the environment
- Goal is to build general model that will produce correct output on novel input.

- Supervision: The data (observations, measurements, etc.) are labeled with pre-defined classes. It is like that a "teacher" gives the classes (supervision).
- Supervised learning is the most common technique for training neural networks and decision trees.
- eg. chess — given game situation and best move

Unsupervised Learning

Unsupervised learning seems much harder: the goal is to have the computer learn how to do something that we don't tell it how to do.

- Learning about data by looking at its features
- No specific feedback from users
- Usually entails clustering data

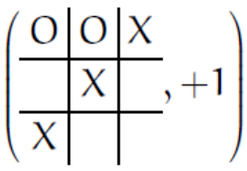
Reinforcement Learning

Learning from feedback (+ve or -ve reward) given at end of a sequence of steps. Unlike supervised learning, the reinforcement learning takes place in an environment where the agent cannot directly compare the results of its action to a desired result. Instead, it is given some reward or punishment that relates to its actions. It may win or lose a game, or be told it has made a good move or a poor one. The job of reinforcement learning is to find a successful function using these rewards.

- Addresses the question of how an autonomous agent that senses and acts in its environment can learn to choose optimal actions to achieve its goals
- Use reward or penalty to indicate the desirability of the resulting state
- Example problems
  - control a mobile robot
  - learn to optimize operations in a factory
  - learn to play a board game

Inductive Learning

- Inductive learning is supervised learning.
- Simplest form: learn a function from examples
  - f is the target function
  - An example is a pair (x; f(x)), e.g.



- Problem: find a hypothesis h
  such that h ≈ f,
  given training set of examples

- Highly simplified model of real learning:
  - Ignores prior knowledge
  - Assumes a deterministic, observable environment
  - Assumes examples are given
  - Assumes that the agent wants to learn f

Inductive Learning Method

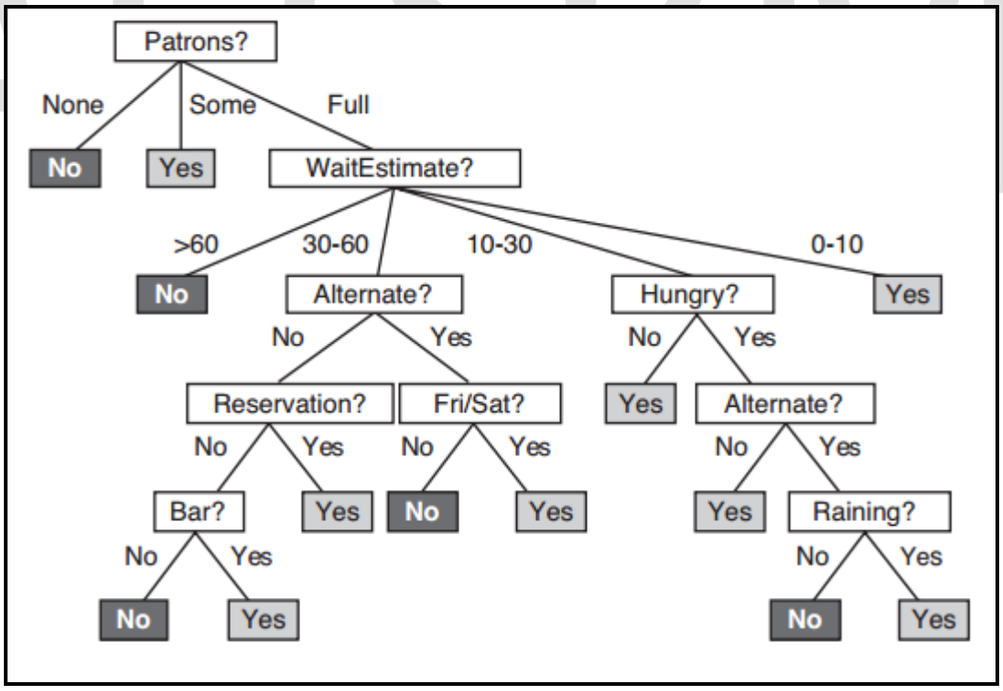
- Construct/adjust h to agree with f on training set
- h is consistent if it agrees with f on all examples.

Learning Decision Tree

- A **decision tree** is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility.
- It is one way to display an algorithm.
- Decision trees are commonly used in operations research, specifically in decision analysis, to help identify a strategy most likely to reach a goal.
- Another use of decision trees is as a descriptive means for calculating conditional probabilities.
- Decision trees can express any function of the input attributes.
- Training input:
  - Data points with a set of attributes
- Classifier output:
  - Can be boolean or have multiple outputs
  - Each leaf stores an "answer"

Example

- Should we wait for a table at a restaurant?
- Possible attributes:
  - Alternate restaurant nearby?
  - Is there a bar to wait in?
  - Is it Friday or Saturday?
  - How hungry are we?
  - How busy is the restaurant?
  - How many people in the restaurant?



Expert System

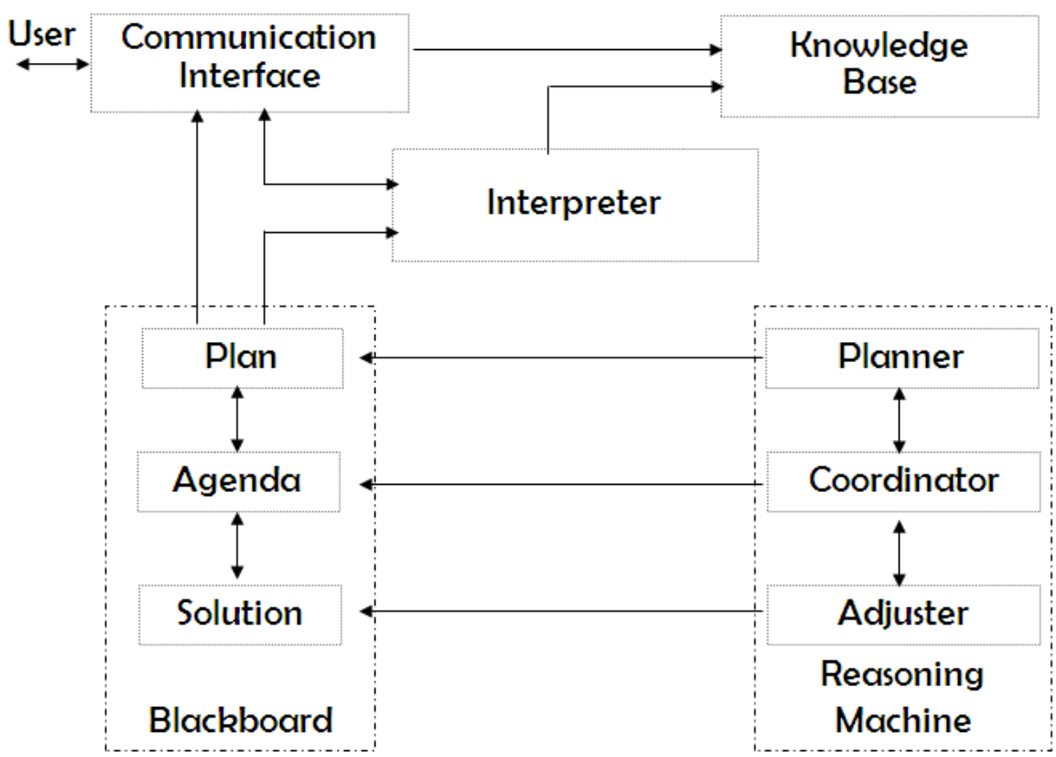
The fundamental function of the expert system depends upon its knowledge; therefore, the expert system is sometimes called knowledge-based system.

"In short, an ES is an intelligent computer program that can perform special and difficult task(s) in some field(s) at the level of human experts."

Why do we need Expert Systems?

- Increased availability
- Reduced Danger
- Reduced Cost
- Multiple expertise
- Increased Reliability
- Explanation facility
- Fast Response
- Steady, emotional & complete response
- Intelligent tutor

Architecture of ideal expert system



- **Knowledge Base**

To store knowledge from the experts of special field(s). It contains facts and feasible operators or rules for heuristic planning and problem solving.

The other data is stored in a separate database called global database, or database simply.

- **Reasoning Machine**

To memorize the reasoning rules and the control strategies applied.

According to the information from the knowledge base, the reasoning machine can coordinate the whole system in a logical manner, draw inference and make a decision.

- **User Interface**

To communicate between the user and the expert system.

The user interacts with the expert system in problem-oriented language such as in restricted English, graphics or a structure editor. The interface mediates information exchanges between the expert system and the human user.

- **Interpreter**

Through the user interface, interpreter explains user questions, commands and other information generated by the expert system, including answers to questions, explanations and justifications for its behavior, and requests for data.

- **Blackboard**

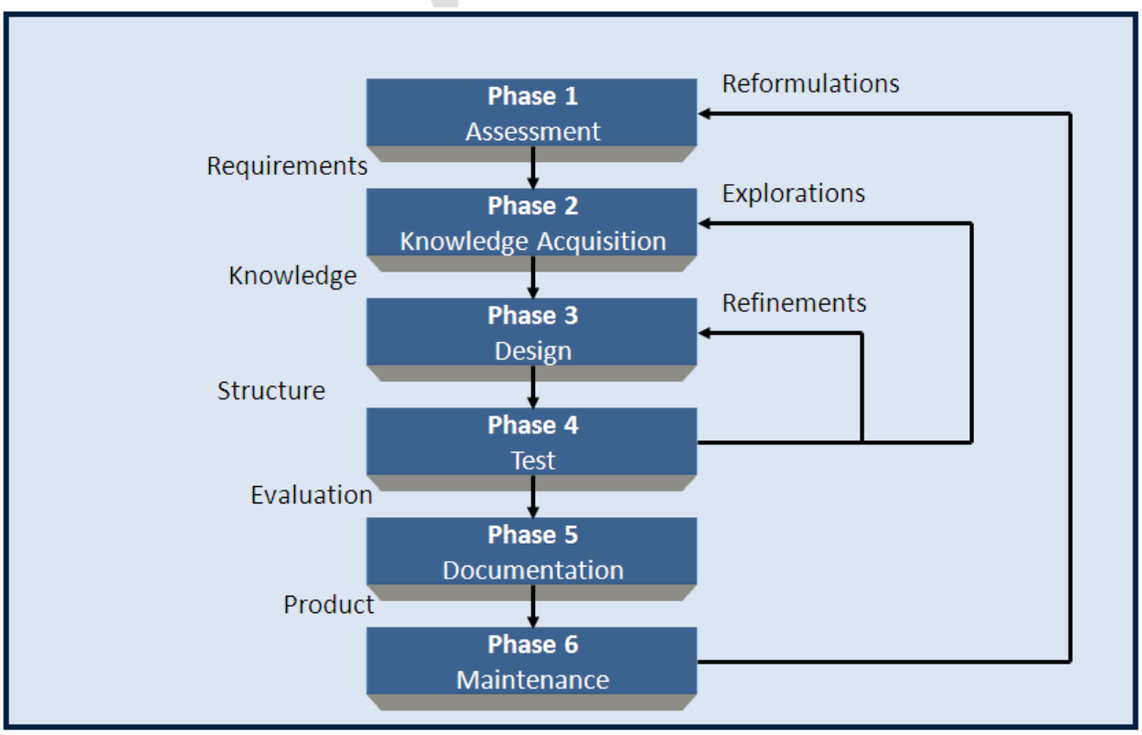
To record intermediate hypotheses and decisions that the expert system manipulates.

Note:

Almost no exiting expert system contains all the components shown above, but some components, especially the knowledge base and reasoning machine, occur in almost all expert systems. Many ESs use global database in place of the blackboard. The global database contains information related to specific tasks and the current state.

Phases in Building Expert System

- The key for successfully building an expert system is to begin it from a smaller one, and extend and test it step by step, make it into a larger-scale and more perfect system.



1. Assessment

    - Determine feasibility & justification of the problem

- Define overall goal and scope of the project
- Resources requirement
- Sources of knowledge

2. Knowledge Acquisition
    1. Acquire the knowledge of the problem
    2. Involves meetings with expert
    3. Bottleneck in ES development
3. Design
    1. Selecting knowledge representations approach and problem solving strategies
    2. Defined overall structure and organization of system knowledge
    3. Selection of software tools
    4. Built initial prototype
    5. Iterative process
4. Testing
    1. Continual process throughout the project
    2. Testing and modifying system knowledge
    3. Study the acceptability of the system by end user
    4. Work closely with domain expert that guide the growth of the knowledge and end user that guide in user interface design
5. Documentation
    1. Compile all the projects information into a document for the user and developers of the system such as:
        - User manual
        - diagrams
        - Knowledge dictionary
6. Maintenance
    1. Refined and update system knowledge to meet current needs

Expert Systems Applications vs Conventional Systems Applications

Expert Systems Applications	**Conventional Systems Applications (Traditional system)**
Knowledge is fragmented and implicit, is difficult to communicate except in small "chunks", and is often distributed amongst individuals who may disagree.	Knowledge is complete and explicit, and is easily communicated with formulas and algorithms.

Rules are complex, conditional and often defined as imprecise “rules of thumb”.	Rules are simple with few conditions.
The finished system captures, distributes and leverages expertise	The finished product automates manual procedures
Problem-solving demands dynamic, context-driven application of facts, relationship and rules	Problem-solving requires predictable and repetetive sequences of actions.
System performance is measured in degrees of accuracy and completeness where explanations may be required to establish correctness.	Simple criteria are used to determine accuracy and completeness.