UNIT - IV

4 Genetic Algorithm

4.1 : Genetic Algorithm : Motivation

Q.1 What is Genetic algorithm ?

Ans. : A genetic algorithm is a search technique used in computing to find true or approximate solutions to optimization and search problems.

Q.2 List the factors motivated the popularity of genetic algorithms. [JNTU : Dec-17, Marks 3]

Ans. : Genetic algorithm is that the problem solving strategy involves using "the strings' fitness to direct the search; therefore they do not require any problem-specific knowledge of the search space, and they can operate well on search spaces that have gaps, jumps, or noise. As each individual string within a population directs the search, the genetic algorithm searches, in parallel, numerous points on the problem state space with numerous search directions.

Q.3 Give an example for fitness function in genetic algorithms. [JNTU : Dec-16, Marks 3]

Ans. : • A fitness function is a particular type of objective function that is used to summaries, as a single figure of merit, how close a given design solution is to achieving the set aims.

- Fitness functions are used in genetic programming and genetic algorithms to guide simulations towards optimal design solutions.

Q.4 Explain the "Darwinian theory of survival"

Ans. : More individuals are produced each generation that can survive. Phenotypic variation exists among individuals and the variation is heritable. Those individuals with heritable traits better suited to the environment will survive.

Q.5 List the basic components used in all genetic algorithms.

Ans. : The basic components common to almost all genetic algorithms are :

1. Fitness function for optimization
2. A population of chromosomes
3. Selection of which chromosomes will reproduce
4. Crossover to produce next generation of chromosomes
5. Random mutation of chromosomes in new generation

Q.6 Compare and contract genetic algorithm with traditional algorithm.

Ans. :

Genetic algorithm	Traditional algorithm
GA generates a population of points at each iteration. The best point in the population approaches an optimal solution.	It generates a single point at each iteration. The sequence of points approaches an optimal solution.
Selects the next population by computation which uses random number generators	Selects the next point in the sequence by a deterministic computation.
Convergence in each iteration in problem independent	Improvement in each iteration is problem specific
Rules are probabilistic.	Rules are fully deterministic.

Q.7 What is use of crossover operator ?

Ans. : A crossover operator is used to recombine two strings to get a better string. In crossover operation, recombination process creates different individuals in the successive generations by

combining material from two individuals of the previous generation

Q.8 Explain two point crossover.

Ans. : Two crossover points are selected, binary string from the beginning of the chromosome to the first crossover point is copied from the first parent, the part from the first to the second crossover point is copied from the other parent and the rest is copied from the first parent again.

Q.9 What is crowding ?

Ans. : Crowding is a phenomenon in which some individual that is more highly fit than others in the population quickly reproduces, so that copies of this individual and very similar individuals take over a large fraction of the population.

Q.10 Why use Genetic algorithm ?

Ans. : • Genetic algorithms evaluate the target function to be optimized at some randomly selected points of the definition domain. Genetic algorithms can be used when no information is available about the gradient of the function at the evaluated points. The function itself does not need to be continuous or differentiable.

- Genetic algorithms can still achieve good results even in cases in which the function has several local minima or maxima.
- Genetic algorithms are stochastic search algorithms which act on a population of possible solutions. They are loosely based on the mechanics of population genetics and selection.
- Genetic Algorithms allow you to explore a space of parameters to find solutions that score well according to a "fitness function".
- Genetic Programming takes genetic algorithms a step further, and treats programs as the parameters. For example, you would breeding path finding algorithms instead of paths, and your fitness function would rate each algorithm based on how well it does.

Q.11 What are limitations on genetic algorithms ?

Ans. : • GAs are not guaranteed to find the global optimum solution to a problem.

- GAs are an extremely general tool and they have no specific way for solving particular problems.
- GAs are usually used when everything else is failed or when we don't have enough knowledge of the search space.
- Even when such specialized techniques exist, it is often interesting to hybridise them with a GA in order to possibly gain some improvements.

Q.12 Explain advantages of Genetic algorithm.

Ans. : • Genetic Algorithm is a stochastic algorithm.

- Randomness as an essential role in both selection and reproduction phases.
- Genetic algorithms always consider a population of solutions. A population base algorithm is also very amenable for parallelization.
- There is no particular requirement on the problem before using genetic algorithms, so it can be applied to resolve any problem (optimization).
- GAs are a new field and parts of the theory have still to be properly established. We can find almost as many opinions on GAs as there are researchers in this field.

Q.13 What is Fitness function ?

Ans. : • Fitness is an important concept in genetic algorithms. The fitness of a chromosome determines how likely it is that it will reproduce. Fitness is usually measured in terms of how well the chromosome solves some goal problem. Fitness can also be subjective (aesthetic). E.g., if the genetic algorithm is to be used to sort numbers, then the fitness of a chromosome will be determined by how close to a correct sorting it produces.

- A fitness function quantifies the optimality of a solution (chromosome) so that particular solution may be ranked against all the other solutions. A fitness value is assigned to each solution depending on how close it actually is to solving the problem.
- Ideal fitness function correlates closely to goal plus quickly computable.
- Example : In TSP, f(x) is sum of distances between the cities in solution. The lesser the value, the fitter the solution is.

- The performance of the individual strings is measured by a fitness function. A fitness function is a problem specific user defined heuristic. After each iteration, the members are given a performance measure derived from the fitness function and the "fittest" members of the population will propagate the next iteration.

$$\text{Fitness} = F_i, \text{ Hit} = \lambda_i, \text{ Survival} = \phi_i, \text{ Death} = \delta_i$$

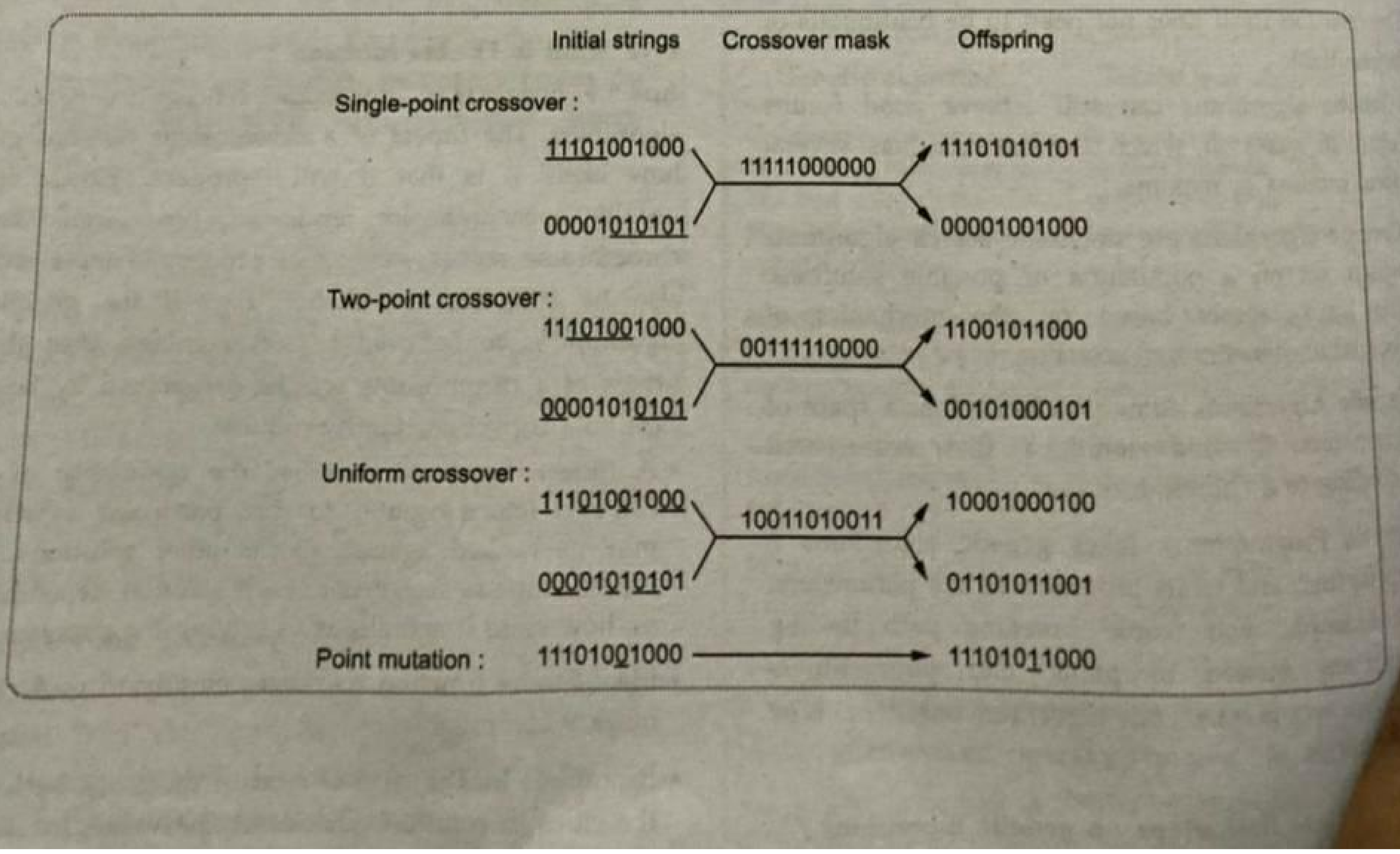
$$F_i = 2\lambda_i - \delta_i + \sum \phi_i$$

- The fitness function is defined over the genetic representation and measures the *quality* of the represented solution. The fitness function is always problem dependent.
- For instance, in the knapsack problem we want to maximize the total value of objects that we can put in a knapsack of some fixed capacity. A representation of a solution might be an array of bits, where each bit represents a different object and the value of the bit (0 or 1) represents whether or not the object is in the knapsack.
- Not every such representation is valid, as the size of objects may exceed the capacity of the knapsack. The *fitness* of the solution is the sum of values of all objects in the knapsack if the representation is valid or 0 otherwise. In some problems, it is hard or even impossible to define the fitness expression; in these cases, interactive genetic algorithms are used.

Q.14 Consider the two strings as initial population for genetic algorithm and generate all possible off springs using various operators.
String 1: 11101001000
String 2: 00001010101

☞ **[JNTU : Dec-17, Marks 10]**

Ans. :



4.2 : Genetic Programming

Q.15 What do you mean genetic programming ?

Ans. : Genetic programming is genetic algorithm wherein the population contains programs rather than bit strings.

Q.16 What is genetic programming ?

Ans. :

- Genetic Programming (GP) is a method to evolve computer programs.
- Genetic programming is a model of programming which uses the ideas of biological evolution to handle a complex problem.
- Genetic programming can be viewed as an extension of the genetic algorithm, a model for testing and selecting the best choice among a set of results, each represented by a string.
- Genetic programming is a recent development in the area of evolutionary computation. It was greatly stimulated in the 1990s by **John Koza.**
- According to Koza, genetic programming searches the space of possible computer programs for a program that is highly fit for solving the problem at hand.
- A parse tree is a good representation of a computer program for Genetic Programming.

Q.17 Explain how genetic algorithms are different from evolutionary programming.

Ans. : 1. Genetic algorithms a coded form of the function values i.e. parameter set, rather than with the actual values themselves. For example, if we want to find the minimum of the function $f(x) = x^3 + x^2 + 5$, the GA would not deal directly with x or y values, but with strings that encode these values. For this case, strings representing the binary x values should be used.

2. Genetic algorithms use a set or population of points to conduct a search, not just a single point on the problem space. This gives GAs the power to search noisy spaces littered with local optimum points. Instead of relying on a single point to search through the space, the GAs looks at many different areas of the problem space at once, and uses all of this information to guide it.

3. Genetic algorithms use only payoff information to guide themselves through the problem space. Many search techniques need a variety of information to guide themselves. Hill climbing methods require derivatives, for example. The only information a GA needs is some measure of fitness about a point in the space. Once the GA knows the current measure of "goodness" about a point, it can use this to continue searching for the optimum.

4. GAs are probabilistic in nature, not deterministic. This is a direct result of the randomization techniques used by GAs.

5. GA is inherently parallel. Here lies one of the most powerful features of genetic algorithms. GA's, by their nature, are very parallel, dealing with a large number of points simultaneously.

Parameters	Genetic Algorithms	Traditional Methods
Work with	Coding of parameter set	Parameters directly
Use information	Payoff i.e. objective function	Payoff plus derivatives etc.
Rules	Probabilistic	Fully deterministic
Search	A population of points	A population of points a single point

4.3 : Models of Evolution and Learning

Q.18 State Baldwin effect. ☞ **[JNTU : Dec-16, Marks 2]**

Ans. : The Baldwin effect works in two steps.

1. Phenotypic plasticity allows an individual to adapt to a partially successful mutation, which might otherwise be useless to the individual. If this mutation increases inclusive fitness, it will tend to proliferate in the population. However, phenotypic plasticity is typically costly for an individual. For example, learning requires energy and time, and it sometimes involves dangerous mistakes.
2. Given sufficient time, evolution may find a rigid mechanism that can replace the plastic mechanism. Thus a behavior that was once

learned (the first step) may eventually become instinctive (the second step).

Q.19 What is Lamarckian evolution ?

Ans. :

- According to Lamarckism, the offspring of those giraffes that did succeed in transmitting an acquired extension of their necks to the next generation could obtain more food than other members of their cohort. They would thus be more numerous, which, in turn, would result in an increase of the average neck length in successive generations.
- The currently accepted view is that the genetic makeup of an individual is, in fact, unaffected by the lifetime experience of one's biological parents.

4.4 : Parallelizing Genetic Algorithms

Q.20 Explain parallelizing genetic algorithm.

Ans. :

- Genetic algorithms are naturally suited to parallel implementation. Classification relies on the computation/communication ratio.
- Coarse grain approaches to parallelization subdivide the population into somewhat distinct groups of individuals, called demes.
  - Each deme is assigned to a different computational node, and a standard GA search is performed at each node.
  - Fine-grained implementations typically assign one processor per individual in the population.
- Fine-grained parallel GAs are suited for massively parallel computers and consist of one spatially-structured population.
- Selection and mating are restricted to a small neighborhood, but neighborhoods overlap permitting some interaction among all the individuals. The ideal case is to have only one individual for every processing element available.
- Fig. Q.20.1 shows a schematic of a master-slave parallel GA.
- The master stores the population, executes GA operations, and distributes individuals to the slaves.

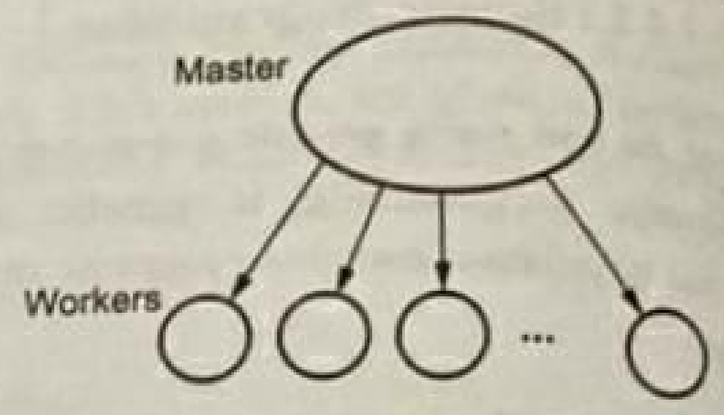


Fig. Q.20.1

The slaves only evaluate the fitness of the individuals.

4.5 : Introduction of Learning Sets of Rules

Q.21 How to learn the target function represented as a set of if-then rules?

Ans. :

- One way to learn sets of rules is to first learn a decision tree, then translate the tree into an equivalent set of rules-one rule for each leaf node in the tree.
- Second method is to use a genetic algorithm that encodes each rule set as a bit string and uses genetic search operators to explore this hypothesis space.

Q.22 What do you mean learning rule ?

Ans. :

- One of the most expressive and human readable representations for learned hypotheses is sets of production rules (if-then rules).
- Rules can be derived from other representations (e.g., decision trees) or they can be learned directly. Here, we are concentrating on the direct method.
- An important aspect of direct rule-learning algorithms is that they can learn sets of first-order rules which have much more representational power than the propositional rules that can be derived from decision trees.
- Learning first-order rules can also be seen as automatically inferring PROLOG programs from examples.

Q.23 What is sequential covering algorithms ?

Ans. :

- A covering algorithm, in the context of propositional learning systems, is an algorithm that develops a cover for the set of positive examples, that is, a set of hypotheses that account for all the positive examples but none of the negative examples.
- This is called sequential covering because it learn one rule at a time and repeat this process to gradually cover the full set of positive examples.
- The algorithm - given a set of examples:

1. Start with an empty Cover
2. Using Learn-One-Rule to find the best hypothesis.
3. If the Just-Learnt-Rule satisfies the threshold then
   - Put Just-Learnt-Rule to the Cover.
   - Remove examples covered by Just-Learnt-Rule.
   - Go to step 2.
4. Sort the Cover according to its performance over examples.
5. Return: Cover.

Q.24 Define First-Order learning problems.

Ans. : In First-Order learning problems, the hypotheses that must be represented involve relational assertions that can be conveniently expressed using first-order representations such as horn clauses.

Q.25 What is difference between Sequential covering and Simultaneous covering ?

Ans. : Sequential covering :

- Learn just one rule at a time, remove the covered examples and repeat the process on the remaining examples.
- Many search steps, making independent decisions to select each precondition for each rule.

Simultaneous covering :

- ID3 learns the entire set of disjunct rules simultaneously as part of a single search for a decision tree.
- Fewer search steps, because each choice influences the preconditions of all rules

Q.26 Explain the algorithm of LEARN-ONE-RULE.

Ans. :

LEARN-ONE-RULE(Target _attribute, Attributes, Examples, k)

// Returns a single rule that covers some of the Examples. Conducts a general to specific greedy beam search for the best rule, guided by the PERFORMANCE metric.

- Initialize Best _hypothesis to the most general hyothesis ϕ
- Initialize Candidate _hypotheses to the set {Best _hypothesis }
- While Candidate _hypotheses is not empty, Do
  1. Generate the next more specific candidate_hypotheses
     New _Candidate _hypotheses ← new generated and specialized candidates
  2. Update Best_hypotheses
     Best _hypothesis ← h with best PERFORMANCE
  3. Update Candidate_hypotheses
     Candidate _hypotheses ← the k best members of New _Candidate _hypotheses
- Return a rule of the form
  "IF Best _hypothesis THEN prediction"
  where prediction is the most frequent value of Target _attribute among those
  Examples that match Best _hypothesis.

4.6 : Learning First-Order Rules

Q.27 Why Learn First Order rules ?

Ans. :

- Propositional logic allows the expression o individual propositions and their truth-function combination.
- For example : propositions like Tom is a man or A men are mortal may be represented by sing proposition letters such as P or Q.
- Truth functional combinations are built up us connectives, such as $\wedge$, $\vee$, -, – (for example : P$\wedge$(

- Inference rules are defined over propositional forms (For example: P → Q)
- Note that if P is Tom is a man and Q is All men are mortal, then the inference that Tom is mortal does not follow in propositional logic.
- First order logic allows the expression of propositions and their truth functional combination, but it also allows us to represent propositions as assertions of predicates about individuals or sets of individuals.
- For example : propositions like Tom is a man or All men are mortal may be represented by predicate-argument representations such as man(tom) or ∀ x(man(x) → mortal(x))
- Inference rules permit conclusions to be drawn about sets/individuals - e.g. mortal(tom)
- First order logic is much more expressive than propositional logic - i.e. it allows a finer-grain of specification and reasoning when representing knowledge.
- In the context of machine learning, consider learning the relational concept daughter(x, y) defined over pairs of persons x, y, where - persons are represented by attributes: <Name, Mother, Father, Male, Female >
- Training examples then have the form: <person1, person2, target_attribute_value >
- Example: << Name1 = Ann, Mother1 = Sue, Father1 = Bob, Male1 = F, Female1 = T > <Name2 = Bob, Mother2 = Gill, Father2 = Joe, Male2 = T, Female2 = F, Daughter1,2 = T >
- From such examples, a propositional rule learner such as ID3 or CN2 can only learn rules like:
- IF (Father1 = Bob) ∧ (Name2 = Bob) ∧ (Female1 = T)
- THEN Daughter1,2 = T
- This will not be useful in classifying future pairs of persons.

Q.28 Discuss about FOIL.

Ans. :

- FOIL is the natural extension of SEQUENTIAL-COVERING and LEARN-ONE-RULE to first order rule learning.
- FOIL learns first order rules which are similar to Horn clauses with two exceptions :
  1. Literals may not contain function symbols (reduces complexity of hypothesis space)
  2. Literals in body of clause may be negated (hence, more expressive than Horn clauses)
- Like SEQUENTIAL-COVERING, FOIL learns one rule at time and removes positive examples covered by the learned rule before attempting to learn a further rule.
- Unlike SEQUENTIAL-COVERING and LEARN-ONE-RULE, FOIL
  1. Only tries to learn rules that predict when the target literal is true, propositional version sought rules that predicted both true and false values of target attribute
  2. Performs a simple hill-climbing search (beam search of width one)
- FOIL searches its hypothesis space via two nested loops :
  1. The **outer loop** at each iteration adds a new rule to an overall disjunctive hypothesis. This loop may be viewed as a specific-to-general search, starting with the empty disjunctive hypothesis which covers no positive instances and stopping when the hypothesis is general enough to cover all positive examples.

2. The **inner loop** works out the detail of each specific rule, adding conjunctive constraints to the rule precondition on each iteration. This loop may be viewed as a general-to-specific search, starting with the most general precondition (empty) and stopping when the hypothesis is specific enough to exclude all negative examples.

Algorithm :

FOIL (Target_predicate, Predicates, Examples)

- Pos ← positive Examples
- Neg ← negative Examples
- Learned_rules ← { }
- while Pos, do

  Learn a NewRule

  NewRule ← most general rule possible (no preconditions)

  NewRuleNeg ← Neg

  while NewRuleNeg, do

  Add a new literal to specialize NewRule

  1. Candidate_literals ← generate candidates based on Predicates
  2. Best_literal ← argmax_{Le} Candidate_literals Foil_Gain(L.NewRule)
  3. Add Best_literal to NewRule preconditions
  4. NewRuleNeg ← subset of NewRuleNeg that satisfies NewRule preconditions

  – Learned_rules ← Learned_rules + NewRule

  – Pos ← Pos – {members of Pos covered by NewRule}

- Return Learned_rules

Q.29 Discuss limitation of FOIL.

Ans. :

1. Search space of literals can become intractable.
2. Requires large extensional background definitions.
3. Hill-climbing search gets stuck at local optima and may not even find a consistent clause.
4. Requires complete examples to learn recursive definitions.
5. Requires a large set of closed-world negatives
6. Inability to handle logical function
7. Background predicates must be sufficient to construct definition, e.g. cannot learn reverse unless given append

4.7 : Induction as Inverted Deduction, Inverting Resolution

Ans. :

- Induction is the inverse of deduction
- Given some data D and some partial background knowledge B, learning can be described as generating a hypothesis h that, together with B, explains D.
- If the training data is a set of examples of the form $<x_i, f(x_i)>$ where x_i denotes the i^{th} training example and $f(x_i)$ denotes its target value.
- Then learning is the problem of discovering h such that

  $(\forall <x_i, f(x_i)> \in D)\ (B \wedge h \wedge x_i)$ entails $f(x_i)$
- Example: Target concept is Child(u, v)
- Single positive example Child(Bob, Sharon) where instance is described by Male(Bob), Female(Sharon), and Father(Sharon, Bob)
- General background knowledge of Parent(u,v) Father(u,v)
- Two of the many hypothesis that satisfy $(B \wedge h \wedge x_i)$ entails $f(x_i)$ are :

  h1: Child(u,v) ← Father(v,u)

  h2: Child(u,v) ← Parent(v,u)
- Induction is, in fact, the inverse operation of deduction, and cannot be conceived to exist without the corresponding operation, so that the question of relative importance cannot arise

Q.31 What is inverting resolution ? Explain.

Ans. :

- The resolution rule is a sound and complete rule for deductive inference in first-order logic.
- Let L be an arbitrary propositional literal, and let P and R be arbitrary propositional clauses.
- The resolution rule is

$$\frac{P \vee L \qquad \neg L \vee R}{P \vee R}$$

- Given the two clauses the line, conclude the clause below the line. Intuitively, the resolution rule is quite sensible.
- Given the two assertions P v L and ¬ L v R, it is obvious that either L or ¬ L must be false. Therefore, either P or R must be true. Thus, the conclusion P v R of the resolution rule is intuitively satisfying.
- This operator used in Cigol
- $C = A \vee B$ and $C_2 = B \vee D$
- Any literal present in C but not in C_1 must be present in C_2
- $C_2 = A \vee \neg D$ or $C_2 = A \vee \neg D \vee B$
- The literal that occurs in C_1 but not in C must be the literal removed by the resolution rule and therefore its negation must occur in C_2.
- Cigol uses inverse resolution with sequential covering butwith first order representations.

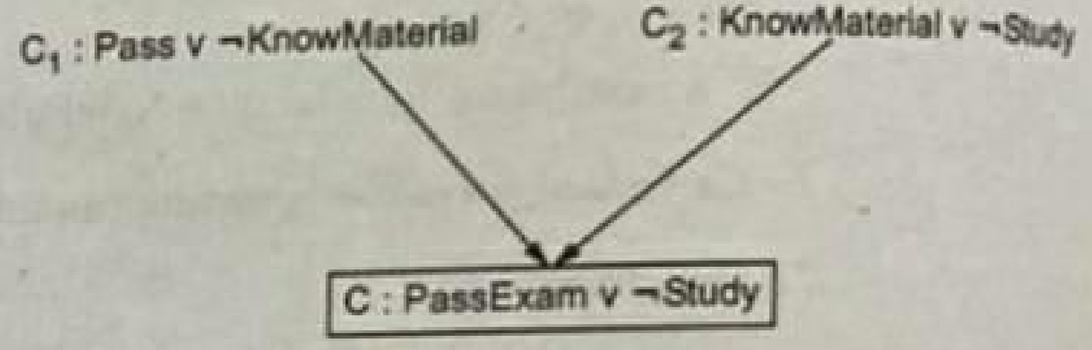


Fig. Q.31.1 Resolution

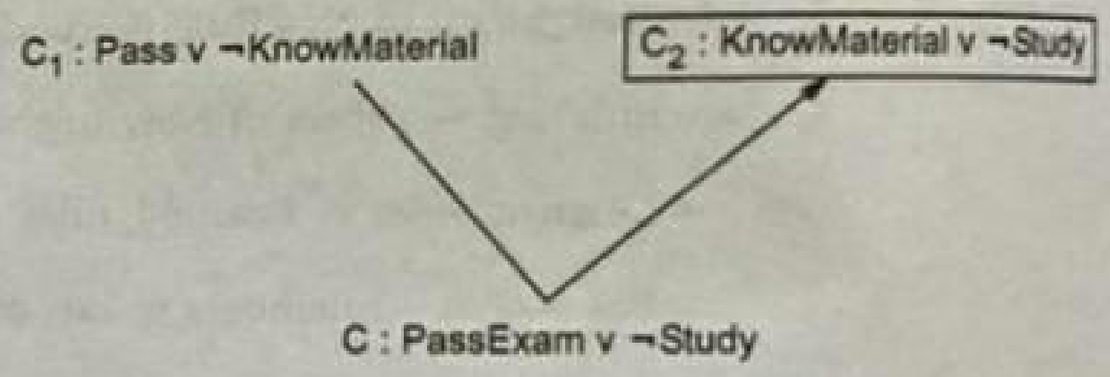


Fig. Q.31.2 Inverse resolution

4.8 : Reinforcement Learning and Q-Learning

Q.32 What is reinforcement learning ?

Ans. : • User will get immediate feedback in supervised learning and no feedback from unsupervised learning. But in the reinforced learning, you will get delayed scalar feedback.

- Reinforcement learning is learning what to do and how to map situations to actions. The learner is not told which actions to take. Fig. Q.32.1 shows concept of reinforced learning.
- Reinforced learning is deals with agents that must sense and act upon their environment. It combines

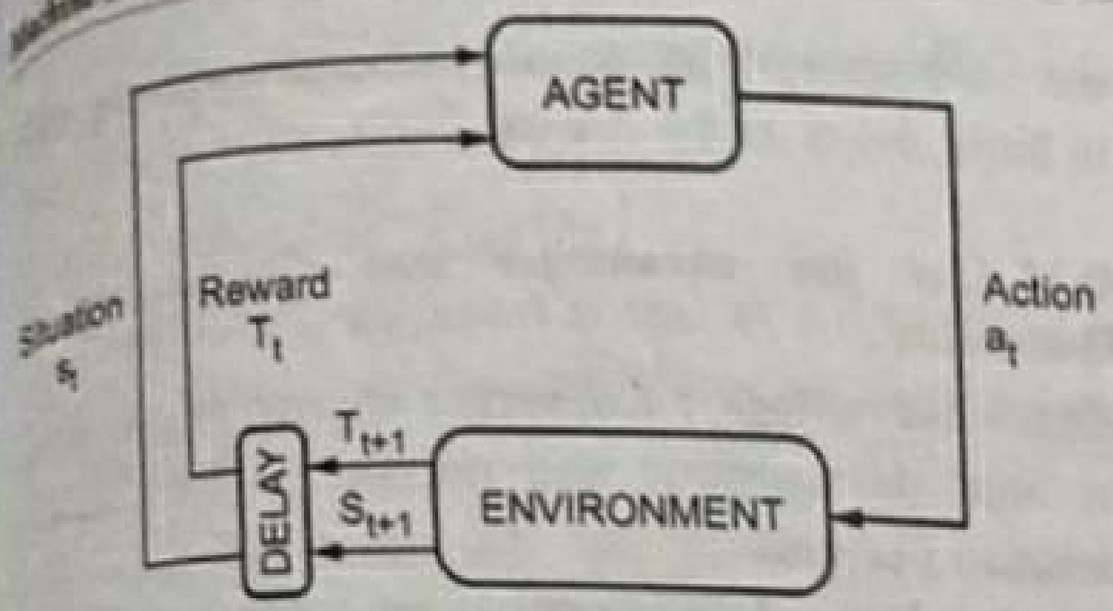


Fig. Q.32.1 Reinforced learning

classical Artificial Intelligence and machine learning techniques.

- It allows machines and software agents to automatically determine the ideal behavior within a specific context, in order to maximize its performance. Simple reward feedback is required for the agent to learn its behavior; this is known as the reinforcement signal.
- Two most important distinguishing features of reinforcement learning is trial-and-error and delayed reward.
- With reinforcement learning algorithms an agent can improve its performance by using the feedback it gets from the environment. This environmental feedback is called the reward signal.
- Based on accumulated experience, the agent needs to learn which action to take in a given situation in order to obtain a desired long term goal. Essentially actions that lead to long term rewards need to reinforced. Reinforcement learning has connections with control theory, Markov decision processes and game theory.
  - **Example of Reinforcement Learning :** A mobile robot decides whether it should enter a new room in search of more trash to collect or start trying to find its way back to its battery recharging station. It makes its decision based on how quickly and easily it has been able to find the recharger in the past.

Q.33 Explain elements of reinforcement learning ?

Ans. : • Reinforcement learning elements are as follows :

1. Policy
2. Reward Function
3. Value Function
4. Model of the environment

- Fig. Q.33.1 shows

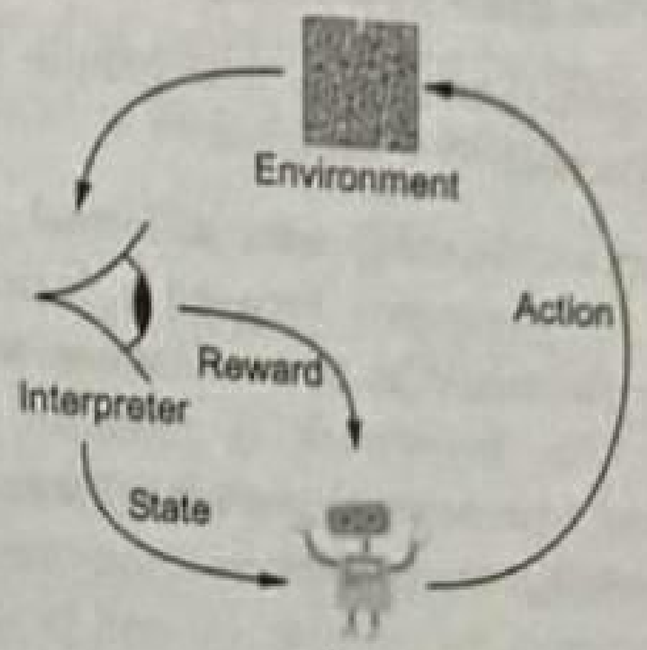


Fig. Q.33.1 : Elements of reinforcement learning

- **Policy :** Policy defines the learning agent behavior for given time period. It is a mapping from perceived states of the environment to actions to be taken when in those states.
- **Reward Function :** Reward function is used to define a goal in a reinforcement learning problem. It also maps each perceived state of the environment to a single number.
- **Value function :** Value functions specify what is good in the long run. The value of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state.
- **Model of the environment :** Models are used for planning
- Credit assignment problem : Reinforcement learning algorithms learn to generate an internal value for the intermediate states as to how good they are in leading to the goal.
- The learning decision maker is called the agent. The agent interacts with the environment that includes everything outside the agent.
- The agent has sensors to decide on its state in the environment and takes an action that modifies its state.
- The reinforcement learning problem model is an agent continuously interacting with an environment. The agent and the environment interact in a sequence of time steps. At each time step t, the agent receives the state of the environment and a

scalar numerical reward for the previous action, and then the agent then selects an action.

- Reinforcement Learning is a technique for solving Markov Decision Problems.
- Reinforcement learning uses a formal framework defining the interaction between a learning agent and its environment in terms of states, actions, and rewards. This framework is intended to be a simple way of representing essential features of the artificial intelligence problem.

Q.34 Explain learning task.

Ans. : • Here we define one quite general formulation of the problem, based on Markov decision processes.

- In a Markov decision process (MDP) the agent can perceive a set S of distinct states of its environment and has a set A of actions that it can perform.
- A Markov decision process is a tuple $(S, A, \{P_{sa}\}, \gamma, R)$

where :

- S is a set of actions. (For example, in autonomous helicopter flight. S might be the set of all possible positions and orientations of the helicopter.)
- A is set of actions. (For example, the set of all possible directions in which you can push the helicopter's control sticks).
- P_{sa} are the state transition probabilities. For each state $s \in S$ and action $a \in A$, P_{sa} is a distribution over the state space. We'll say more about this layer, but briefly, P_{sa} gives the distribution over what states we will transition to if we take action a in state s.
- $\gamma \in (0,1)$ is called the **discount factor**.
- $R : S \times A \rightarrow \Re$ is the **reward function**. (Rewards are sometimes also written as a function of a state S only, in which case we would have $R : S \rightarrow \Re$.

Q.35 Define Q-learning.

Ans. : Q-learning is a form of model-free reinforcement learning. It can also be viewed as a method of asynchronous dynamic programming (DP). It provides agents with the capability of learning to act optimally in Markovian domains by experiencing the consequences of actions, without requiring them to build maps of the domains

Q.36 List the advantages and disadvantage of Q-learning.

Ans. : Advantage : Converges to an optimal policy in both deterministic and nondeterministic Markov decision process

Disadvantage : Only practical on a small number of problems.

Q.37 What is Q-Learning ? Explain

Ans. : • Q-learning is a reinforcement learning technique used in machine learning. The goal of Q-Learning is to learn a policy, which tells an agent which action to take under which circumstances.

- In Q-learning , an agent tries to learn the optimal policy from its history of interaction with the environment.
- It has the ability to compute the utility of the actions without a model for the environment. It takes the help of action-value pair and the expected reward from the current action.
- During this process the agent learns to move around the environment and understand the current state which is the optimal policy by taking the action with the highest reward. Let us look at an example of this technique.
- The value Q(s,a) is defined to be the expected discounted sum of future payoffs obtained by taking action a from state s and following an optimal policy thereafter.
- Once these values have been learned, the optimal action from any state is the one with the highest Q-value.
- Consider a computational agent moving around some discrete, finite world, choosing one from a finite collection of actions at every time step. The world constitutes a controlled Markov process with the agent as a controller.
- At step n, the agent is equipped to register the state $x_n (\in X)$ of the world, an can choose its action an $(\in 2)$ accordingly.
- The agent receives a probabilistic reward r_n, whose mean value depends only on the state and action

and the state of the world changes probabilistically to y_n according to the law :

$$Prob[y_n = y | x_n, a_n] = P_{x_n y}[a_n]$$

- The task facing the agent is that of determining an optimal policy, one that maximizes total discounted expected reward.
- By discounted reward, we mean that rewards received s steps hence are worth less than rewards received now, by a factor of $y^s (0 < y < 1)$.
- The state values of x :

$$Q^{\pi}(x, a) = R_x(a) + \gamma \sum_y P_{xy}[\pi(x)] V^{\pi}(y).$$

4.9 : Non-deterministic Rewards and Actions

Q.38 Discuss about non-deterministic rewards and actions of Q-learning.

Ans. :

- In the nondeterministic case reward function r(s,a) and action transition function δ (s,a) have probabilistic outcomes.
- For example, in robot problems with noisy sensors and effectors it is often appropriate to model actions and rewards as nondeterministic.
- In such cases, the functions δ (s, a) and r(s, a) can be viewed as first producing a probability distribution over outcomes based on s and a, and then drawing an outcome at random according to this distribution.
- When these probability distributions depend solely on s and a, then we call the system a nondeterministic Markov decision process.
- In the nondeterministic case, we must first restate the objective of the learner to take into account the fact that outcomes of actions are no longer deterministic.
- The obvious generalization is to redefine the value of a policy n to be the expected value V^{π} of the discounted cumulative reward received by applying this policy

$$V^{\pi}(S_t) = E\left[\sum_{i=0}^{\infty} \gamma^i r_{t+i}\right]$$

- The optimal policy n^* to be the policy n that maximizes V"(s) for all states s. Next we generalize definition of Q again by taking its expected value

$$Q(s, a) = E[r(s, a) + \gamma V^*(\delta(s, a))]$$

$$= E[r(s, a)] + \gamma E[V^*(\delta(s, a))]$$

$$= E[r(s, a)] + \gamma \sum_{s'} P(s'|s, a) V^*(s')$$

Where $P(s'|s, a)$ is the probability that taking action a in state s will produce the next state s'. Note we have used $P(s'|s, a)$ here to rewrite the expected value of $V^*(\delta(s, a))$ in terms of the probabilities associated with the possible outcomes of the probabilistic δ

4.10 : Temporal Difference Learning

Q.39 What is Temporal Difference Learning ? Explain in detail.

Ans. : • Temporal-difference (TD) learning is a combination of Monte Carlo ideas and dynamic programming (DP) ideas.

- Like Monte Carlo methods, TD methods can learn directly from raw experience without a model of the environment's dynamics.
- Like DP, TD methods update estimates based in part on other learned estimates, without waiting for a final outcome.
- The relationship between TD, DP, and Monte Carlo methods is a recurring theme in the theory of reinforcement learning.
- TD learning, which is a model-free learning algorithm, has two important properties:
  1. It doesn't require the model dynamics to be known in advance
  2. It can be applied for non-episodic tasks as well
- The algorithm takes the benefits of both the Monte Carlo method and dynamic programming (DP) into account :
  1. Like the Monte Carlo method, it doesn't require model dynamics, and
  2. Like dynamic programming, it doesn't need to wait until the end of the episode to make an estimate of the value function.

- We try to predict the state values in temporal difference learning, TD learning does not need model of the environment unlike DP.
- TD learning using something called a TD update rule for updating the value of a state :

$$V(S_t) \leftarrow V(S_t) + \alpha\,[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

- value of a previous state = value of previous state + learning_rate*(reward + discount_factor (value of current state) — value ofprevious state)
- This is the difference between the **actual reward (r + Gamma * V(s'))** and the expected reward **V(s)** multiplied by the learning rate alpha.
- The learning rate, also called step size, is useful for convergence.

Fill in the Blanks for Mid Term Exam

Q.1 The chromosome are divided into several parts called _____ .

Q.2 Genetic algorithms are inspired by _____ theory about evolution.

Q.3 The mutation depends on the encoding as well as the _____.

Q.4 A genetic algorithm is a _____ technique used in computing to find true or approximate solutions to optimization and search problems.

Q.5 _____ is the process of taking two parent solutions and producing from them a child.

Multiple Choice

Q.1 Genetic Algo

a evolution

b inspired - "surviv

c are adap based on selection

d All of th

Q.2 Which GA expensive ?

a Initial p

b Selectio

c Reprodu

d Conver

Q.3 Which of th algorithms

a It is a p

b It is gu solutio

c If an o one sol solutio

d It is an progra