GENETIC ALGORITHMS

Course Code: **CSC4226** Course Title: **Artificial Intelligence and Expert System**

Dept. of Computer Science
Faculty of Science and Technology

Lecture No:	**Five (5)**	**Week No:**	**Five (5)**	**Semester:**	
Lecturer:	*SAZIA SHARMIN*				

GA: BASIC CONCEPT

Genetic algorithms (GAs) are the main paradigm of evolutionary computing. GAs are inspired by Darwin's theory about evolution – the "survival of the fittest". In nature, competition among individuals for scanty resources results in the fittest individuals dominating over the weaker ones.

- GAs are the ways of solving problems by mimicking processes nature uses; ie., Selection, Crosses over, Mutation and Accepting, to evolve a solution to a problem.
- GAs are adaptive heuristic search based on the evolutionary ideas of natural selection and genetics.
- GAs are intelligent exploitation of random search used in optimization problems.
- GAs, although randomized, exploit historical information to direct the search into the region of better performance within the search space.

GA: FLOW-CHART

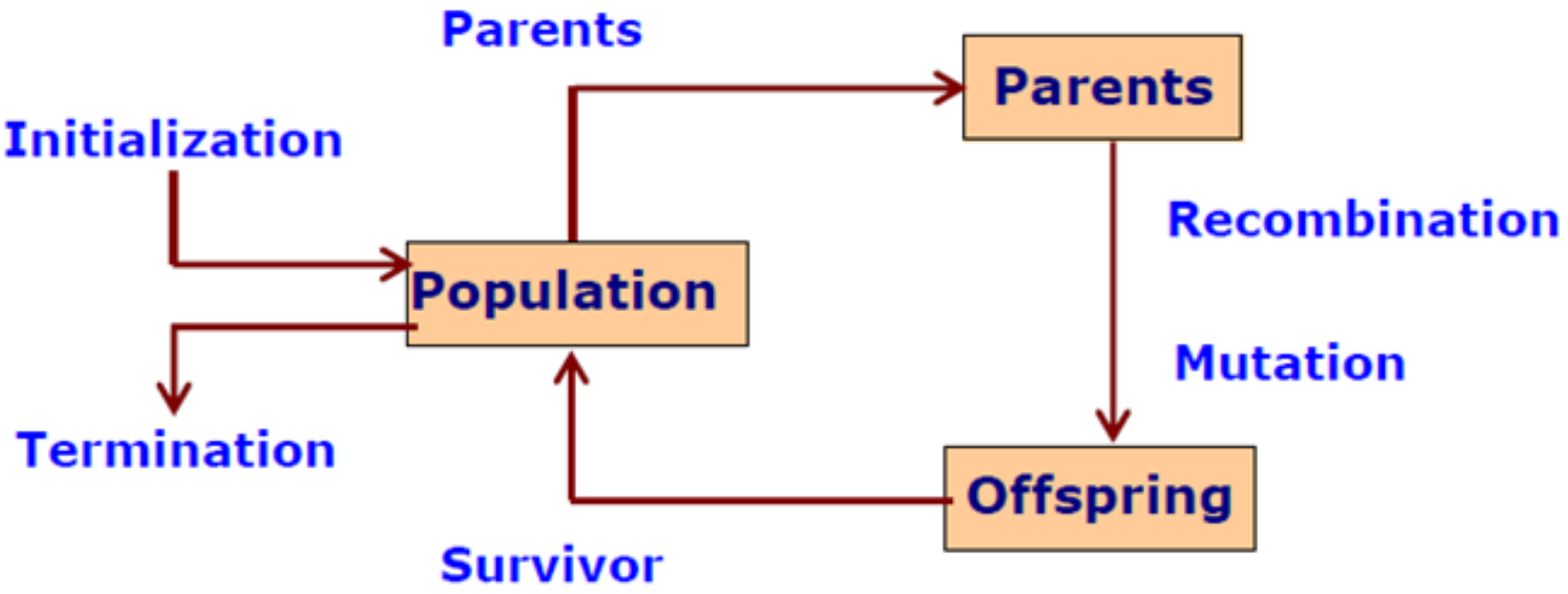

Fig. General Scheme of Evolutionary process

GA: PSEUDO-CODE

BEGIN

INITIALISE population with random candidate solution.

EVALUATE each candidate;

REPEAT UNTIL (termination condition) is satisfied DO

1. SELECT parents;
2. RECOMBINE pairs of parents;
3. MUTATE the resulting offspring;
4. SELECT individuals or the next generation;

END.

ENCODING

Before a genetic algorithm can be put to work on any problem, a method is needed to encode potential solutions to that problem in a form so that a computer can process.

- One common approach is to encode solutions as binary strings: sequences of **1's** and **0's**, where the digit at each position represents the value of some aspect of the solution.

- There are many other ways of encoding, e.g., encoding values as integer or real numbers or some permutations and so on.
- The virtue of these encoding method depends on the problem to work on .

BINARY ENCODING

Binary encoding is the most common to represent information contained.

In genetic algorithms, it was first used because of its relative simplicity.

- In binary encoding, every chromosome is a string of bits : **0** or **1**, like

  Chromosome 1: 1 0 1 1 0 0 1 0 1 1 0 0 1 0 1 0 1 1 1 0 0 1 0 1

  Chromosome 2: 1 1 1 1 1 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 1 1 1

- Binary encoding gives many possible chromosomes even with a small number of alleles ie possible settings for a trait (features).

- This encoding is often not natural for many problems and sometimes corrections must be made after crossover and/or mutation.

Example: Binary Encoding

A Gene represents some data (eye color, hair color, sight, etc.).

A chromosome is an array of genes. In binary form

a **Gene** looks like : **(11100010)**

a **Chromosome** looks like: **Gene1 Gene2 Gene3 Gene4**

(11000010, 00001110, 001111010, 10100011)

A chromosome should in some way contain information about solution which it represents; it thus requires encoding. The most popular way of encoding is a **binary string** like :

Chromosome 1 : 1101100100110110

Chromosome 2 : 1101111000011110

Each bit in the string represent some characteristics of the solution.

Value Encoding

The Value encoding can be used in problems where values such as real numbers are used. Use of binary encoding for this type of problems would be difficult.

1. In value encoding, every chromosome is a sequence of some values.
2. The Values can be anything connected to the problem, such as : real numbers, characters or objects.

   Examples :

   Chromosome A 1.2324 5.3243 0.4556 2.3293 2.4545

   Chromosome B ABDJEIFJDHDIERJFDLDFLFEGT

   Chromosome C (back), (back), (right), (forward), (left)

3. Value encoding is often necessary to develop some new types of crossovers and mutations specific for the problem.

Permutation Encoding

Permutation encoding can be used in ordering problems, such as traveling salesman problem or task ordering problem.

1. In permutation encoding, every chromosome is a string of numbers that represent a position in a sequence.

   Chromosome A **1 5 3 2 6 4 7 9 8**

   Chromosome B **8 5 6 7 2 3 1 4 9**

2. Permutation encoding is useful for ordering problems. For some problems, crossover and mutation corrections must be made to leave the chromosome consistent.

Examples: Permutation Encoding

1. The Traveling Salesman problem:

   There are cities and given distances between them. Traveling salesman has to visit all of them, but he does not want to travel more than necessary. Find a sequence of cities with a minimal traveled distance. Here, encoded chromosomes describe the order of cities the salesman visits.

2. The Eight Queens problem :

   There are eight queens. Find a way to place them on a chess board so that no two queens attack each other. Here, encoding describes the position of a queen on each row.

Operators of GA

Genetic operators used in genetic algorithms maintain genetic diversity.

Genetic diversity or variation is a necessity for the process of evolution.

Genetic operators are analogous to those which occur in the natural world:

- **Reproduction** (or Selection) ;
- **Crossover** (or Recombination); and
- **Mutation**.

Selection

Many reproduction operators exists and they all essentially do same thing. They pick from current population the strings of above average and insert their multiple copies in the mating pool in a probabilistic manner.

The most commonly used methods of selecting chromosomes for parents to crossover are :

- Roulette wheel selection,
- Boltzmann selection,
- Tournament selection,
- Rank selection
- Steady state selection.

The Roulette wheel and Boltzmann selections methods are illustrated next.

Example of Selection

Evolutionary Algorithms is to maximize the function $f(x) = x^2$ with x in the integer interval **[0 , 31]**, i.e., **x = 0, 1, . . . 30, 31.**

1. The first step is encoding of chromosomes; use binary representation for integers; 5-bits are used to represent integers up to **31**.
2. Assume that the population size is **4**.
3. Generate initial population at random. They are chromosomes or genotypes; e.g., **01101, 11000, 01000, 10011**.
4. Calculate fitness value for each individual.
   (a) Decode the individual into an integer (called phenotypes),
   01101 → 13; 11000 → 24; 01000 → 8; 10011 → 19;
   (b) Evaluate the fitness according to $f(x) = x^2$,
   13 → 169; 24 → 576; 8 → 64; 19 → 361.

Example of Selection

5. Select parents (two individuals) for crossover based on their fitness in p_i. Out of many methods for selecting the best chromosomes, if **roulette-wheel** selection is used, then the probability of the **i th** string in the population is $p_i = F_i / (\sum_{j=1}^{n} F_j)$, where

F_i is fitness for the string **i** in the population, expressed as **f(x)**

p_i is probability of the string **i** being selected,

n is no of individuals in the population, is population size, **n=4**

$n * p_i$ is expected count

String No	Initial Population	X value	Fitness Fi $f(x) = x^2$	p i	Expected count N * Prob *i*
1	0 1 1 0 1	13	169	0.14	0.58
2	1 1 0 0 0	24	576	0.49	1.97
3	0 1 0 0 0	8	64	0.06	0.22
4	1 0 0 1 1	19	361	0.31	1.23
Sum			1170	1.00	4.00
Average			293	0.25	1.00
Max			576	0.49	1.97

Roulette Wheel Selection

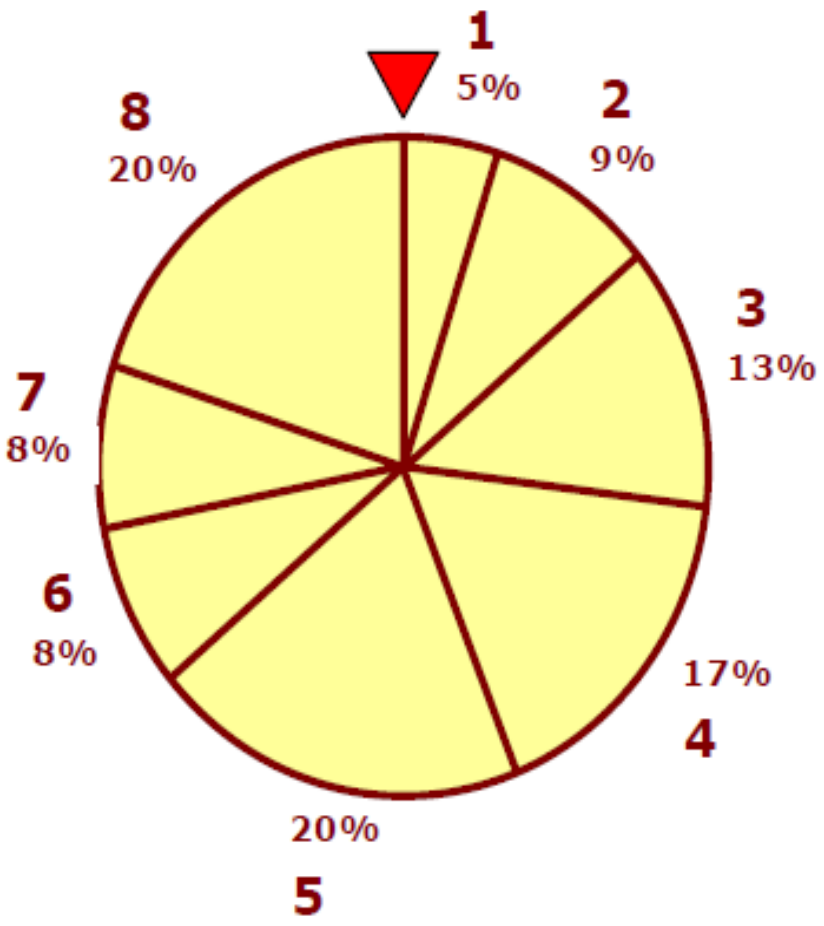

Fig. Roulette-wheel Shows 8 individual with fitness

The Roulette-wheel simulates **8** individuals with fitness values F_i, marked at its circumference; e.g.,

- the **5th** individual has a higher fitness than others, so the wheel would choose the **5th** individual more than other individuals .
- the fitness of the individuals is calculated as the wheel is spun **n = 8** times, each time selecting an instance, of the string, chosen by the wheel pointer.

Calculate fitness value

Genetic Algorithm Solved Example – Calculate Fitness Value

String No.	Initial Population (Randomly Selected)	X Value	Fitness $f(x) = x^2$	Prob	% Prob	Expected Count	Actual Count
1	01100	12	144	0.1247	12.47	0.4987	1
2	11001	25	625	0.5411	54.11	2.1645	2
3	00101	5	25	0.0216	2.16	0.0866	0
4	10011	19	181	0.3126	31.26	1.2502	1
Sum			1155	1.0	100	4	4
Average			288.75	0.25	25	1	1
Maximum			625	0.5411	54.11	2.1645	

Calculate fitness value

Genetic Algorithm Solved Example – Calculate Fitness Value

String No.	Mating Pool	Crossover Point	Offspring after crossover	X Value	Fitness $f(x) = x^2$
1	01100	4	01101	13	169
2	11001		11000	24	576
3	11001	2	11011	27	729
4	10011		10001	17	289
Sum					1763
Average					440.75
Maximum					729

Mutation

Genetic Algorithm Solved Example – Mutation

String No.	Offspring after crossover	Mutation Chromosome for flipping	Offspring after mutation	X Value	Fitness $f(x) = x^2$
1	01101	10000	11101	29	841
2	11000	00000	11000	24	576
3	11011	00000	11011	27	729
4	10001	00101	10100	20	400
Sum					2546
Average					636.5
Maximum					841

Roulette Wheel Selection

Probability of i^{th} string is $p_i = F_i / (\sum_{j=1}^{n} F_j)$, where

n = no of individuals, called population size; **p_i = probability** of i^{th} string being selected; **F_i = fitness** for i^{th} string in the population.

Because the circumference of the wheel is marked according to a string's fitness, the Roulette-wheel mechanism is expected to make $\frac{F}{\bar{F}}$ copies of the i^{th} string.

Average fitness = $\bar{F} = \sum F_j / n$; Expected count = (n =8) x p_i

Cumulative Probability$_5$ = $\sum_{i=1}^{N=5} p_i$

Crossover

Crossover is a genetic operator that combines (mates) two chromosomes (parents) to produce a new chromosome (offspring). The idea behind crossover is that the new chromosome may be better than both of the parents if it takes the best characteristics from each of the parents. Crossover occurs during evolution according to a user-definable crossover probability. Crossover selects genes from parent chromosomes and creates a new offspring.

The Crossover operators are of many types.

- one simple way is, **One-Point crossover**.
- the others are **Two Point, Uniform, Arithmetic, and Heuristic crossovers**.

The operators are selected based on the way chromosomes are encoded.

One-Point Crossover

One-Point crossover operator randomly selects one crossover point and then copy everything before this point from the first parent and then everything after the crossover point copy from the second parent. The Crossover would then look as shown below.

Consider the two parents selected for crossover.

Parent 1	**1 1 0 1 1 | 0 0 1 0 0 1 1 0 1 1 0**
Parent 2	**1 1 0 1 1 | 1 1 0 0 0 0 1 1 1 1 0**

Interchanging the parents chromosomes after the crossover points -
The Offspring produced are :

Offspring 1	**1 1 0 1 1 | 1 1 0 0 0 0 1 1 1 1 0**
Offspring 2	**1 1 0 1 1 | 0 0 1 0 0 1 1 0 1 1 0**

Note : The symbol, a vertical line, | is the chosen crossover point.

Two-Point Crossover

Two-Point crossover operator randomly selects two crossover points within a chromosome then interchanges the two parent chromosomes between these points to produce two new offspring.

Consider the two parents selected for crossover :

Parent 1	**1 1 0 1 1 | 0 0 1 0 0 1 1 | 0 1 1 0**
Parent 2	**1 1 0 1 1 | 1 1 0 0 0 0 1 | 1 1 1 0**

Interchanging the parents chromosomes between the crossover points - The Offspring produced are :

Offspring 1	**1 1 0 1 1 | 1 1 0 0 0 0 1 | 0 1 1 0**
Offspring 2	**1 1 0 1 1 | 0 0 1 0 0 1 1 | 1 1 1 0**

Uniform Crossover

Uniform crossover operator decides (with some probability – know as the mixing ratio) which parent will contribute how the gene values in the offspring chromosomes. The crossover operator allows the parent chromosomes to be mixed at the gene level rather than the segment level (as with one and two point crossover).

Consider the two parents selected for crossover.

Parent 1	1	1	0	1	1	0	0	1	0	0	1	1	0	1	1	0
Parent 2	1	1	0	1	1	1	1	0	0	0	0	1	1	1	1	0

If the mixing ratio is **0.5** approximately, then half of the genes in the offspring will come from parent **1** and other half will come from parent **2**. The possible set of offspring after uniform crossover would be:

Offspring 1	1_1	1_2	0_2	1_1	1_1	1_2	1_2	0_2	0_1	0_1	0_2	1_1	1_2	1_1	1_1	0_2
Offspring 2	1_2	1_1	0_1	1_2	1_2	0_1	0_1	1_1	0_2	0_2	1_1	1_2	0_1	1_2	1_2	0_1

Note: The subscripts indicate which parent the gene came from.

Mutation

After a crossover is performed, mutation takes place.

Mutation is a genetic operator used to maintain genetic diversity from one generation of a population of chromosomes to the next.

Mutation occurs during evolution according to a user-definable mutation probability, usually set to fairly low value, say **0.01** a good first choice.

Mutation alters one or more gene values in a chromosome from its initial state. This can result in entirely new gene values being added to the gene pool. With the new gene values, the genetic algorithm may be able to arrive at better solution than was previously possible.

Flip Bit Mutation

The mutation operator simply inverts the value of the chosen gene.

i.e. **0** goes to **1** and **1** goes to **0**.

This mutation operator can only be used for binary genes.

Consider the two original off-springs selected for mutation.

Original offspring 1	1	1	0	1	1	1	1	0	0	0	0	1	1	1	1	0
Original offspring 2	1	1	0	1	1	0	0	1	0	0	1	1	0	1	1	0

Invert the value of the chosen gene as **0** to **1** and **1** to **0**

The Mutated Off-spring produced are :

Mutated offspring 1	1	1	0	**0**	1	1	1	0	0	0	0	1	1	1	1	0
Mutated offspring 2	1	1	0	1	1	0	**1**	1	0	0	1	1	0	1	**0**	0

Demonstration

- **Example 1 :**

  Maximize the function $f(x) = x^2$ over the range of integers from **0 . . . 31**.

  Note : This function could be solved by a variety of traditional methods such as a hill-climbing algorithm which uses the derivative.

  One way is to :

  - Start from any integer **x** in the domain of **f**
  - Evaluate at this point **x** the derivative **f'**
  - Observing that the derivative is **+ve**, pick a new **x** which is at a small distance in the **+ve** direction from current **x**
  - Repeat until **x = 31**

  See, how a genetic algorithm would approach this problem ?

GA Approach

Genetic Algorithm Approach to problem - Maximize the function $f(x) = x^2$

1. Devise a means to represent a solution to the problem :

   Assume, we represent **x** with five-digit unsigned binary integers.

2. Devise a heuristic for evaluating the fitness of any particular solution :

   The function **f(x)** is simple, so it is easy to use the **f(x)** value itself to rate the fitness of a solution; else we might have considered a more simpler heuristic that would more or less serve the same purpose.

3. Coding - Binary and the String length :

   GAs often process binary representations of solutions. This works well, because crossover and mutation can be clearly defined for binary solutions. A Binary string of length **5** can represents 32 numbers (0 to 31).

GA Approach

4. Randomly generate a set of solutions :

Here, considered a population of four solutions. However, larger populations are used in real applications to explore a larger part of the search. Assume, four randomly generated solutions as : **01101, 11000, 01000, 10011.** These are chromosomes or genotypes.

5. Evaluate the fitness of each member of the population :

The calculated fitness values for each individual are -

(a) Decode the individual into an integer (called phenotypes),

01101 → 13; 11000 → 24; 01000 → 8; 10011 → 19;

(b) Evaluate the fitness according to $\mathbf{f(x) = x^2}$,

13 → 169; 24 → 576; 8 → 64; 19 → 361.

(c) Expected count = **N * Prob** $_i$, where **N** is the number of individuals in the population called population size, here **N = 4.**

Thus the evaluation of the initial population summarized in table below .

Calculating Fitness & Probability

String No i	Initial Population (chromosome)	X value (Pheno types)	Fitness $f(x) = x^2$	Prob i (fraction of total)	Expected count N * Prob i
1	0 1 1 0 1	13	169	0.14	0.58
2	1 1 0 0 0	24	576	0.49	1.97
3	0 1 0 0 0	8	64	0.06	0.22
4	1 0 0 1 1	19	361	0.31	1.23
Total (sum)			1170	1.00	4.00
Average			293	0.25	1.00
Max			576	0.49	1.97

Thus, the string no 2 has maximum chance of selection.

Selection

6. Produce a new generation of solutions by picking from the existing pool of solutions with a preference for solutions which are better suited than others:

We divide the range into four bins, sized according to the relative fitness of the solutions which they represent.

Strings	*Prob i*	*Associated Bin*
0 1 1 0 1	0.14	0.0 . . . 0.14
1 1 0 0 0	0.49	0.14 . . . 0.63
0 1 0 0 0	0.06	0.63 . . . 0.69
1 0 0 1 1	0.31	0.69 . . . 1.00

By generating **4** uniform **(0, 1)** random values and seeing which bin they fall into we pick the four strings that will form the basis for the next generation.

Random No	*Falls into bin*	*Chosen string*
0.08	0.0 . . . 0.14	0 1 1 0 1
0.24	0.14 . . . 0.63	1 1 0 0 0
0.52	0.14 . . . 0.63	1 1 0 0 0
0.87	0.69 . . . 1.00	1 0 0 1 1

Crossover

7. Randomly pair the members of the new generation

   Random number generator decides for us to mate the first two strings together and the second two strings together.

8. Within each pair swap parts of the members solutions to create offspring which are a mixture of the parents :

   For the first pair of strings: **0 1 1 0 1 , 1 1 0 0 0**

   – We randomly select the crossover point to be after the fourth digit. Crossing these two strings at that point yields:

   0 1 1 0 1 ⇒ 0 1 1 0 |1 ⇒ 0 1 1 0 0

   1 1 0 0 0 ⇒ 1 1 0 0 |0 ⇒ 1 1 0 0 1

   For the second pair of strings: **1 1 0 0 0 , 1 0 0 1 1**

   – We randomly select the crossover point to be after the second digit. Crossing these two strings at that point yields:

   1 1 0 0 0 ⇒ 1 1 |0 0 0 ⇒ 1 1 0 1 1

   1 0 0 1 1 ⇒ 1 0 |0 1 1 ⇒ 1 0 0 0 0

Mutate & Re-evaluate

9. Randomly mutate a very small fraction of genes in the population :

With a typical mutation probability of per bit it happens that none of the bits in our population are mutated.

10. Go back and re-evaluate fitness of the population (new generation) :

This would be the first step in generating a new generation of solutions. However it is also useful in showing the way that a single iteration of the genetic algorithm has improved this sample.

String No	Initial Population (chromosome)	X value (Pheno types)	Fitness $f(x) = x^2$	Prob i (fraction of total)	Expected count
1	0 1 1 0 0	12	144	0.082	0.328
2	1 1 0 0 1	25	625	0.356	1.424
3	1 1 0 1 1	27	729	0.415	1.660
4	1 0 0 0 0	16	256	0.145	0.580
Total (sum)			1754	1.000	4.000
Average			439	0.250	1.000
Max			729	0.415	1.660

Improved:
Individual & Total Fitness

1. Initial populations : At start step 5 were

   0 1 1 0 1 , 1 1 0 0 0 , 0 1 0 0 0 , 1 0 0 1 1

   After one cycle, new populations, at step 10 to act as initial population

   0 1 1 0 0 , 1 1 0 0 1 , 1 1 0 11 , 1 0 0 0 0

2. The total fitness has gone from **1170** to **1754** in a single generation.
3. The algorithm has already come up with the string 11011 (i.e **x = 27**) as a possible solution.

Books

1. “Artificial Intelligence: A Modern Approach,” by Stuart J. Russell and Peter Norvig.
2. "Artificial Intelligence: Structures and Strategies for Complex Problem Solving", by George F. Luger, (2002)
3. "Artificial Intelligence: Theory and Practice", by Thomas Dean.
4. "AI: A New Synthesis", by Nils J. Nilsson.
5. “Programming for machine learning,” by J. Ross Quinlan,
6. “Neural Computing Theory and Practice,” by Philip D. Wasserman, .
7. “Neural Network Design,” by Martin T. Hagan, Howard B. Demuth, Mark H. Beale, .
8. “Practical Genetic Algorithms,” by Randy L. Haupt and Sue Ellen Haupt.
9. “Genetic Algorithms in Search, optimization and Machine learning,” by David E. Goldberg.
10. "Computational Intelligence: A Logical Approach", by David Poole, Alan Mackworth, and Randy Goebel.
11. “Introduction to Turbo Prolog”, by Carl Townsend.