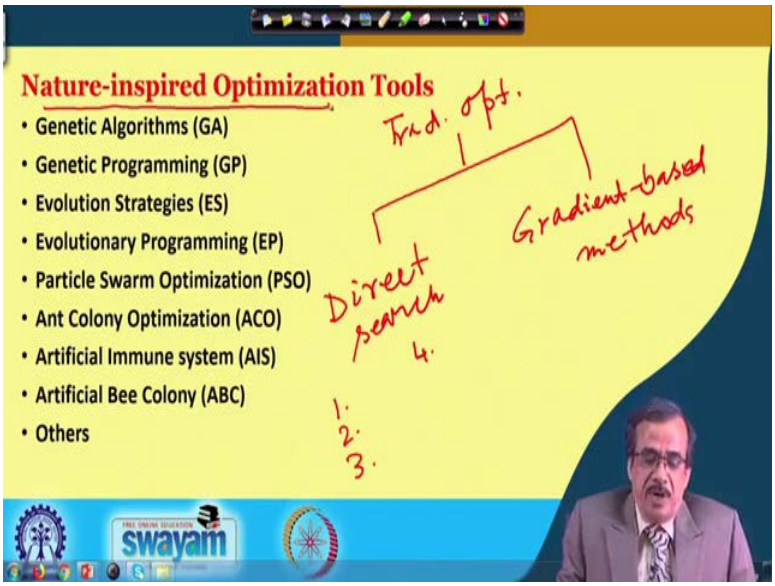
Fuzzy Logic and Neural Networks
Prof. Dilip Kumar Pratihar
Department of Mechanical Engineering
Indian Institute of Technology, Kharagpur

Lecture – 17
Optimization of Fuzzy Reasoning and Clustering Tool

(Refer Slide Time: 00:24)

We are going to discuss on optimal design of fuzzy reasoning and clustering tools. Now, in this lecture, I am just going to concentrate on these topics. At first, I will give a brief introduction to the nature-inspired optimization tools; and after that, I will concentrate on how to optimize the fuzzy reasoning tool; and at the end, we will deal with optimization related to fuzzy clustering. Now, let me start with this optimization or the nature-inspired optimization.

(Refer Slide Time: 01:00)



Now, before I start with this nature-inspired optimization tools, let me give you a very brief introduction to the concept of optimization, and why should you go for this nature-inspired optimization tools. And, these are also known as nontraditional optimization tools. Now, by the term: optimization actually we try to select the best design out of all the possibilities. Supposing that we have got a number of feasible designs, and we try to find out which one is the optimal, that is the task of optimization.

Now, if you see the literature on optimization, a huge literature is available. Now, there are a large number of classical tools or the traditional tools for optimization, these are also known as the conventional methods of optimization. And, there are a few unconventional or nature-inspired optimization tools. Now, if you see the conventional or the traditional optimization tool, so traditional optimization tool if you see, these are broadly classified into two groups. Now, one is called the direct search method, and another is known as your the gradient-based method. So, we have got this gradient based method.

Now, this gradient-based method actually, we try to find out the search direction by knowing the gradient information of the objective function. So, in gradient search method, the algorithm will try to move along the direction or opposite to the direction depending on the problem, whether it is maximization or minimization. On the other hand, in direct search method, the search direction is decided by the value of the

objective function. And, here we do not need the gradient information of the objective function.

Now, here for these traditional methods actually there are a few demerits. I am just going to mention those demerits one after another, and then I will try to explain, why should you go for so this nature-inspired optimization tools. Now, if you see the various demerits of the traditional or the conventional optimization tools, the first one is, in traditional method, there is a chance that the algorithm is going to get stuck at the local optimum solution. So, instead of reaching the globally optimal solution, there is a possibility that we will be getting the locally optimal solution.

Now, if you are going to use the gradient-based method and supposing that I have got one objective function, which is discontinuous, so we cannot find out the gradient. So, it is bit difficult to apply the gradient-based optimization tool. Now, then comes supposing that we have got a few integer variables. Now, if there are integer variables, it becomes bit difficult in traditional tool to tackle that type of optimization problem.

The next is your or if you see the parallel computing like if you want to make it faster, this traditional tool, as it starts with only one solution selected at random, we cannot use the principle of parallel computing. And, if you see, this traditional tool for optimization, so to solve different types of problems, we will have to take the help of various algorithms. And, a particular algorithm may not be suitable to solve a variety of problems. So, it may not be so much robust.

Now, to overcome all these drawbacks of traditional tools for optimization, we take the help of the nature-inspired optimization tool or non-traditional optimization tool. Now, let us see the reality, now whenever you face some very difficult problem, the real world problem, we are unable to use the traditional tools for optimization in a very efficient way. And, the moment we felt, we try to see the way our nature has solved the similar type of problems, and we try to copy its principle in the artificial way. And consequently actually a large number of the nature-inspired optimization tools have come into the picture.

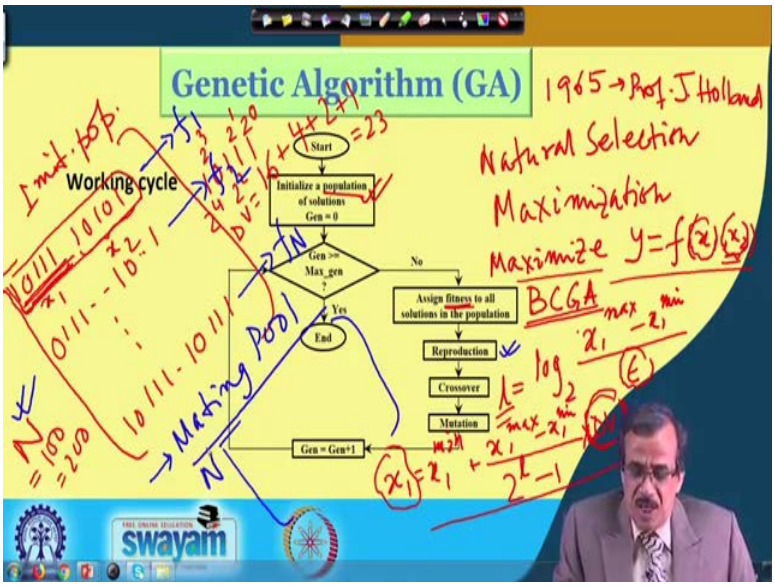
(Refer Slide Time: 06:22)

Now, if you see the literature, a large number of tools are available, the literature is huge. For example, say it starts with say genetic algorithm, in short this is known as GA, which was proposed in the year 1965. There are a few other algorithms like your genetic programming, in short this is known as GP. We have got the evolution strategies, now this is known as ES; then we have got evolutionary programming, it is known as EP. Then, comes your particle swarm optimization, in short this is known as PSO.

Then, we have got ant colony optimization, this is known as ACO; then, we have got artificial immune system, that is AIS. We have got artificial bee colony, in short this is known as ABC, and others. We have got, in fact, a large number of nature-inspired optimization tools. Now, this particular course actually, it is not a course on optimization. So, I will not be able to discuss all such algorithms in details.

Now, what I am going to do is, I am just going to discuss, in brief, the working principle of at least one algorithm and that is the most popular one, that is your the genetic algorithm. And, we are going to use the principle of genetic algorithm just to optimize the fuzzy reasoning tool and clustering algorithms. So, we are going to discuss all such things, in details.

(Refer Slide Time: 08:22)



So, let me start with your the working cycle of a genetic algorithm. Now, as I told that this is not a course on optimization, and I will be discussing in brief the working principle of genetic algorithm. Now, if you want to get more information, you will have to refer to the textbook of this course, that is, Soft Computing: Fundamentals and Applications, or we have already developed another course on Optimization, that is a MOOC course, NPTEL course, that is Traditional and Non-traditional Optimization Tools. So, you will have to attend that particular course

Now, let me start with the working principle or the working cycle of this particular the genetic algorithm. Now, as I told, the genetic algorithm was proposed in the year 1965 by Prof. John Holland. Now, he is from the University of Michigan, USA. Now, Prof. John Holland, he proposed actually the concept of this genetic algorithm. And, genetic algorithm is actually a population-based search and optimization tool, which works based on Darwin's principle of natural selection. So, according to this natural selection, it is the survival of the fittest. So, on principle, this genetic algorithm can solve the maximization problem. But, supposing that you have got a minimization problem, now this minimization problem can be converted into a maximization problem for the purpose of solving.

Now, here, as I told that it starts with a population of solution, and this population of solution is generated at random using the random number generator. Now, let me take

the very simple example, supposing that I am just going to maximize a very simple problem say an optimization problem and this is a function of say two variables: x_1 and x_2. And, of course, x_1 and x_2 are having some range, like the minimum and maximum values for x_1 and x_2. Now, what we do is, here actually I am just going to use the binary-coded GA, that is, BCGA. We have got a few other types of GA like real-coded GAs, then comes here gray-coded GAs and so, but here, we are going to concentrate on the binary-coded genetic algorithm.

Now, I am just going to tell you, in short, how to generate this initial population of solution. And, its name indicates the BCGA, that is a binary-coded genetic algorithm that means the variables x_1 and x_2 will be represented with the help of some binary numbers. For example, this is nothing but a collection of 1s and 0s. Now, to represent this x_1 and x_2, the first thing we will have to do is, we will have to select how many bits we are going to assign to represent these x_1 and x_2. Now, if we need the better precision, we will have to assign more number of bits and vice-versa.

Now, depending on the level of the precision we need, if we want to determine, how many bits to be assigned, that can be determined very easily using this particular relationship. So, $l = \log_2 \frac{x_1^{\max} - x_1^{\min}}{\varepsilon}$. And, this ε is actually the level of precision we need. Now, this indicates that if you need more precision or the better precision, so we will have to assign a large number of bits.

Now, let me assume that say we are going to assign 5-bits to represent each of these x_1 and x_2. Now, if you see this initial population or a particular GA-string. So, these particular five bits are going to represent x_1; then comes your 1 0 1 0 1, another five bits, so this is going to represent x_2. So, the GA-string will be actually 10-bits long and this is population-based approach.

So, what we do is, we try to actually generate the whole population, and the population size is denoted by capital N. So, this could be equal to say 100 or say 200 or say 50 depending on the complexity of the problem. And, what you do is, this population of solutions, in fact, we generate at random. So, the whole population of solutions will be generated at random, and this is the way, actually, by using the random number generator, we can generate the initial population of your genetic algorithm.

Now, as this particular population is generated at random, there is no guarantee that we will be able to select all such good solutions in the initial population. So, this is nothing but your initial population of genetic algorithm. And, once you have got, the number of bits and the bits to represent x_1, so very easily, we can find out the real value corresponding to this particular x_1, and which is determined as follows: So, $x_1 = x_1^{\min} + \frac{x_1^{\max} - x_1^{\min}}{2^l - 1} \times DV$.

Now, x_1 minimum, x_1 maximum will be supplied, l is actually the number of bits to be assigned to represent x_1. Now, we will have to find out this decoded value. Now, determining the decoded value is not difficult. Now, let us concentrate on these particular 5-bits. Supposing that I have got the bits like here 1 0 1 1 1, the place value for this particular 1 is 2 raised to the power 0. Here, it is 2 raise to the power 1; the place value for this is 2 raised to the power 2; this is 2 raised to the power 3, and this is 2 raised to the power 4.

Now, its decoded value will be nothing but is your 2 raise to the power 4 is nothing but 16 plus 2 raise to the power 4 is 4 plus 2 raise to the power 1 is 2 plus 2 raise to the power 0 is 1, so 16 plus 4: 20 plus 2: 22 plus 1, that is 23. So, 23 is nothing but is your the decoded value. So, very easily, you can find out the real value for this particular x_1 using this linear mapping rule. Now, this is known as the linear mapping rule.

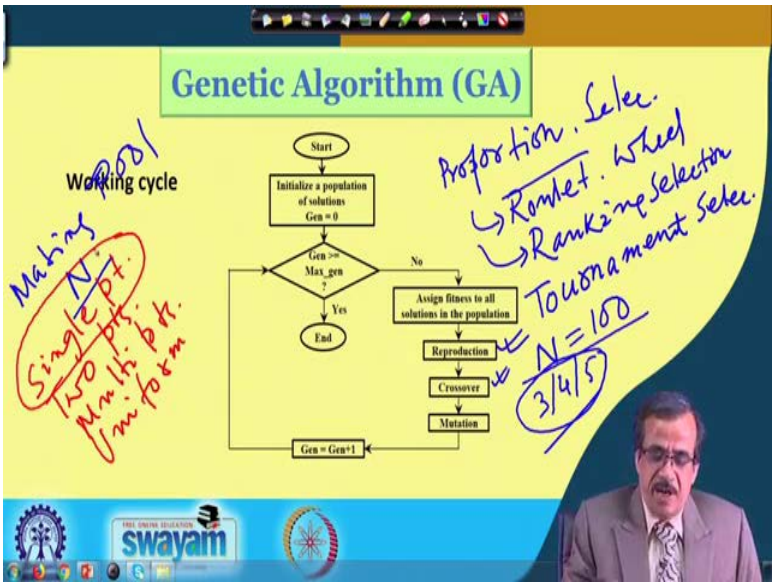
Now, by following the same procedure, I can also find out, what should be the value for your x_2, the real value for this particular x_2. And, once you have got, the real values for these particular your x_1 and x_2, so using the expression of say objective function, so I can find out, what should be the numerical value for this particular objective. And, this is nothing but the fitness of a particular GA-string provided this is a maximization problem.

Now, once you have got this particular fitness value for the whole population, we are in a position to use the other operators, so that we can further modify it. Now, supposing that your so this particular fitness information is something like this. Now, supposing that the fitness for the first GA-string is your f_1, then comes your second GA-string is f_2, and your the last GA-string, so its fitness is nothing but f_n. And, once you have got the

fitness information for the whole population, we are in a position to use actually the operator and that is called the reproduction operator.

Now, the purpose of this reproduction operator is to actually select the good solutions from the initial population just to make the mating pool. Now, the population size for this mating pool will once again be equal to your N, that is N could be say 100. So, by using the reproduction scheme actually, we are going to select all such good strings from the initial population, and we will try to make one population of size capital N, that is nothing but the mating pool. Now, if you see the literature, so we have got, in fact, the different types of your the reproduction scheme.

(Refer Slide Time: 18:51)



Now, if you see the literature, we have got the reproduction scheme like the proportionate selection. So, we are just going to discuss the reproduction scheme. So, we have got the proportionate selection. So, this proportionate selection, actually proportionate selection actually, what we do is, we try to select the GA-string based on its fitness value. So, the higher the fitness or the more the fitness, so higher will be the probability of being selected in the mating pool and vice-versa. Now, this proportionate selection can be implemented with the help of your the roulette wheel selection, which is actually the very old roulette wheel selection or we can go for some sort of your the ranking selection. So, this roulette wheel selection or the ranking selection, what we do

is, we try to select the good solutions from the initial population and its probability of being selected is proportional to the fitness.

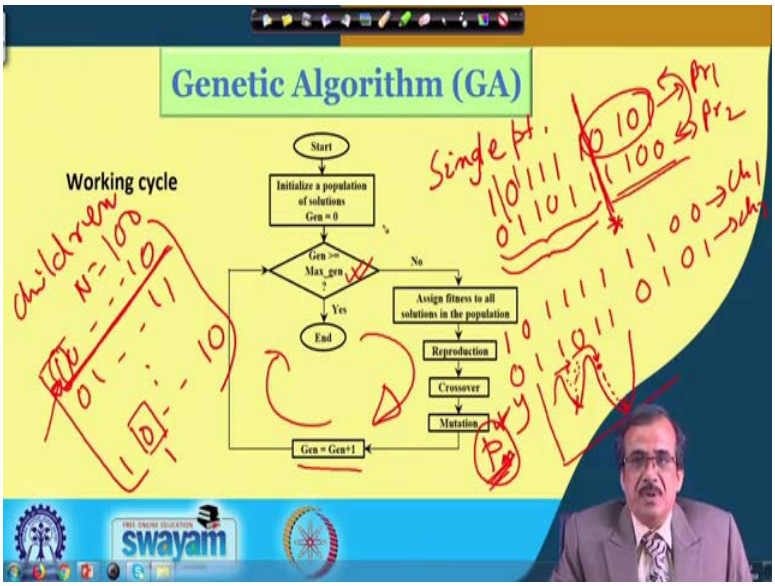
Now, next, we have got another very popular reproduction scheme and that is known as your the tournament selection. Now, here actually, what I do is, in tournament selection, we choose a tournament size. Now, if the population size that is denoted by capital N, if it is 100, so the tournament size, we select as say might be 3 or say 4 or 5, generally not more than 5.

And, depending on this particular tournament size, what we do is, we select a few solutions at random from the initial population, and we compare in terms of their fitness values. And, if it is a maximization problem, we select the GA-string which corresponds to your the maximum fitness and we put it in the mating pool. And, if the population size is 100, so 100 times you will have to play this particular tournament, and each time, we are going to select a particular string.

Now, supposing that we have got the mating pool. Now, the size of the mating pool once again will be made equal to your population size, that is nothing but your capital N. And, now we just go for the next operator and that is nothing but the cross over operator. Now, in crossover operator actually, there will be an exchange of properties and the two parents are going to participate in crossover. And, due to the exchange of properties, some good children solutions will be created.

Now, if you see the literature, we have got different types of crossover schemes. For example, say in binary-coded genetic algorithm, so we have got the single-point cross over, then we have got the two-point crossover, then we have got the multi-point crossover, then comes your uniform crossover, and so on. So, we have got the different types of the crossover. Now, as the time is short and this is not a course on optimization, so I am just going to discuss say might be I am discussing only the single point crossover. Now, just to see like how does it work, now let me try to concentrate on the single-point crossover only.

(Refer Slide Time: 22:52)



Now, if I see the principle of your single-point crossover, the principle is very simple. Now, supposing that I have got say the two parents, which are going to participate in crossover. Now, one parent could be your 1 0 1 1 1 1 0 1 0 1, and another parent could be 0 1 1 0 1 1 1 1 0 0. Supposing that we have got so these two parents, and these two parents, so this is parent_1 and this is your parent_2. So, these two parents are going to participate in say a single-point crossover.

Now, what you do is there are 10 bits. So, there are 9 places for the crossover. Now, supposing that I select a particular crossover site at random and supposing that the cross over site is selected here. Now, if this is the cross over site, which is selected at random, then the bits which are lying on the left hand side will be kept intact, and there will be swapping of the bits, which are lying on the right side of this particular crossover site.

So, if you just do this particular the single-point crossover, we will be getting the children solution as follows: So, the children solution will be 1 0 1 1 1 1, then 0 1 1 0 1 1. So, this will remain the same. And, there will be swapping here. So, these particular bits will go up. So, this will become 1 1 0 0, and this will come down. So, this will become 0 1 0 1. So, this is nothing but child_1 and this is nothing but child_2. So, in children solution, there is an exchange of properties. Now, if the parents are good, the children solutions are expected to be good, but there is no guarantee.

So, using this crossover operator, there is an exchange of properties. And due to this particular exchange, there is a possibility that some new properties will be added and which is going to give some sort of diversification to the solution. Now, once you have got the children solution, the size of the children solution will become once again equal to N, so N is the population size. So, I will be getting all such children solutions here. So, all such children solutions will be getting here. And, the population size is once again is kept equal to your capital N, and might be capital N is equals to 100.

Next, we go for your the mutation operator. Now, before I discuss the principle of this mutation operator, let me try to understand the reason behind going for this particular mutation operator. Now, in biology, so by mutation, we mean a sudden change of properties. Now, if you see a particular GA-string, GA-string can be compared to our chromosome, and on the chromosome, you will find some gene values and the properties of a human being depends on the gene values. Similarly, the property of this particular GA-string will depend on the bit values.

Now, what we do is in biological mutation, we change, there is a sudden change of parameter; the same thing has been copied here in the artificial mutation. Now, what we do is we try to check with each of the bits present in the population, whether it is going to participate in mutation with a small probability value, which is denoted by the p_m. So, p_m indicates actually the probability of mutation.

So, what we do is, we try to check at each of the bit positions, whether there will be mutation or not, how to implement. It is very simple. So, what we do is, at each of the bit position, we do the coin tossing for appearing head with a small probability p_m. So, if head appears, this is a success, so we go for the mutation; otherwise the bit will remain the same, that means, it will remain as the intact one.

So, what you do is, at each of the bit positions, we check with the probability p_m. And, if there is a success, if head appears in coin-tossing, so what you will do is, a particular bit at which we get that success of the coin-tossing, if there is 0 here, that will be replaced by 1 and vice-versa. And, by doing that, so by doing this bit-wise mutation, we can bring some sort of a local change. And, this particular mutation is going to help us just to avoid actually this local minimum problem.

Now, let me take a very simple example. So, we will understand the concept of so this local minima, and how can this particular mutation help to overcome this local minima problem? Let me take a very simple example, the objective function is a function of only one variable. So, y is a function of x. And, supposing that I have got a plot like this, and I am solving a minimization problem. And, this is nothing but the globally minimum solution, and this is your locally optimal or the locally minimum solution. Supposing that, all the population solutions are lying in this local basin. So, if you run genetic algorithm for a large number of iteration, there is no guarantee that we are going to hit the globally optimal solution.

Now, the function of this mutation is just to push one solution to the global basin. And, if you can push one solution to the global basin, then iteratively it is going to reach that globally optimal solution. So, this is the way, actually mutation is going to help. Now, with the application of this reproduction, crossover and mutation, that completes one iteration or one generation of the GA.

Now, this particular process will continue for a large number of iterations. The moment it reaches this termination criterion, we declare that is the end of generation, and we try to find out the optimal solution. Now, this is the way actually, one genetic algorithm, one binary-coded genetic algorithm works to determine the optimal solution.

Thank you.