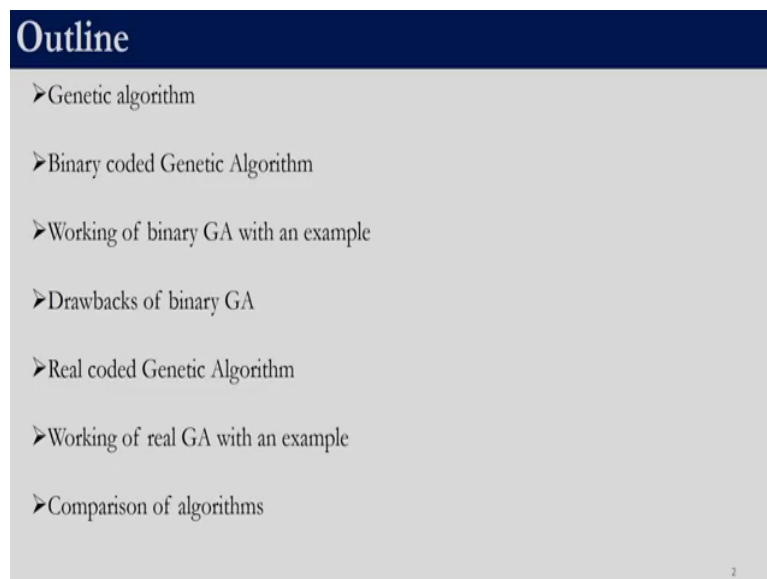


Computer Aided Applied Single Objective Optimization
Dr. Prakash Kotecha
Department of Chemical Engineering
Indian Institute of Technology, Guwahati

Lecture - 14
Genetic Algorithm

Welcome back. In today's session we will be looking at Genetic Algorithm. So, genetic algorithm is the most popular among the heuristic techniques which we have discussed. This session we will broadly divide it into 2 parts, in the first part we will look into something called as binary coded genetic algorithm and in the second part we will look into the real coded genetic algorithm.

(Refer Slide Time: 00:49)

A slide titled "Outline" with a dark blue header. The content is a list of topics with right-pointing arrowheads. The slide number "2" is in the bottom right corner.

Outline
➤ Genetic algorithm
➤ Binary coded Genetic Algorithm
➤ Working of binary GA with an example
➤ Drawbacks of binary GA
➤ Real coded Genetic Algorithm
➤ Working of real GA with an example
➤ Comparison of algorithms

Again in binary coded genetic algorithm, we will first look at the working principle and then as usual we will take an example and try to understand binary coded genetic algorithm. We will

follow it with a brief discussion on the drawbacks of binary coded genetic algorithm and then we will move on to the real coded genetic algorithm.

(Refer Slide Time: 01:12)

Genetic Algorithm

Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence
Author: John H. Holland

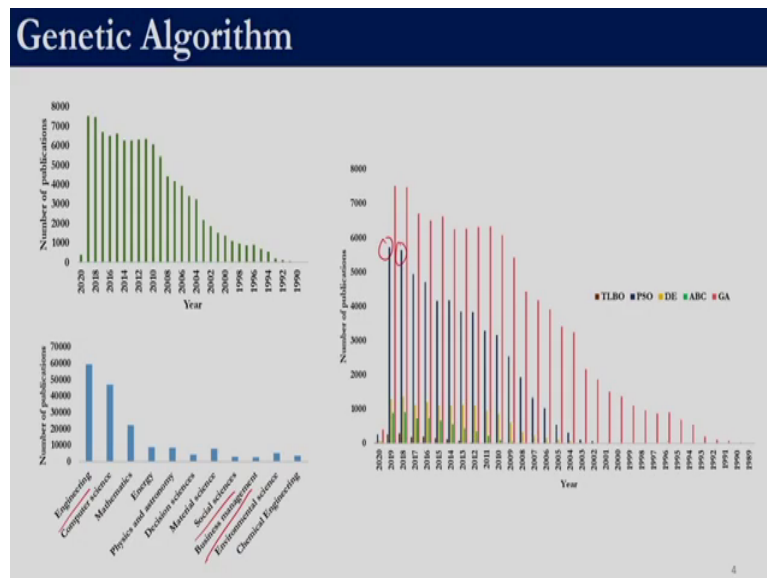
Genetic Algorithms in Search, Optimization and Machine Learning
Author: David E. Goldberg

Multi-Objective Optimization Using Evolutionary Algorithms
Authors: Kalyanmoy Deb, Deb Kalyanmoy

3

These are 3 classical books on genetic algorithm. The first one is by John H Holland, and the other one is by David E Goldberg, and the third one is actually a multi-objective optimization textbook, right, but one or two chapters are dedicated for single objective genetic algorithm. So, what we are going to discuss is a small part from this book, right. So, for additional reading you can look into this book by Professor Kalyanmoy Deb.

(Refer Slide Time: 01:42)



So, this shows the popularity of genetic algorithm over the past two decades. So, as we see there has been an exponential increase in the number of publications which use genetic algorithm. Again like all our previous metaheuristic techniques, genetic algorithm is used in diverse area, right from engineering to social sciences, business management and so on and so forth. This shows the comparison of the metaheuristic techniques which we are learning in as part of this course.

(Refer Slide Time: 02:46)

Genetic Algorithm (GA)

- Inspired by the principles of natural genetics and selection (Darwin's principle of natural evolution).
- Solution vectors are termed as chromosome.
- **Parent:** Solution from which new solutions are generated.
- **Offspring:** Newly generated solutions.
- Offspring are generated through
 - reproduction (selection of good solutions for mating).
 - variation (crossover and mutation).
- Selection of good solutions after variation operator.
- Better candidate has more chance to survive in an environment of limited resources.
 - Good solutions are retained and bad solutions are eliminated.

Genetic Algorithms in search, optimization and machine learning, Addison-Wesley publishing company, 1989 5

So, as we see genetic algorithm has a clear edge over other techniques, right. So, genetic algorithm if we see throughout the years it has been widely used, right, followed by particle swarm optimization, the peak is of particle swarm optimization. And for the other techniques they were recently proposed, in the more in the last decade unlike genetic algorithm which has been there for quite some time. So, this is the status of those 5 metaheuristic techniques that we are discussing as part of this course.

So, genetic algorithm is inspired by the principles of natural genetics and selections. So, it is based on Darwin's principle of natural evolution. So, in this genetic algorithm solution vectors are termed as chromosome, right. There are two types of chromosome, one is parent chromosome and then the other one is the offspring chromosome. So, parent chromosomes

are the solutions from which new solutions would be generated whereas, offspring chromosomes are the newly generated solutions, right.

So, similar to the other techniques this is also a population based technique, right. From the population we will extract few good members and we will term them as parent. So, we will be using these parents to produce the offsprings. So, this offsprings are generated through reproduction and variation reproduction involves selection of good solutions for mating whereas, variation involves crossover and mutation, right.

So, once offsprings are generated we select good solutions after the variation operator. So, initially we will have a population, we will generate a offsprings, we will combine them and we will take the best members in the population, right. So, genetic algorithm is designed such a way that better candidate has more chance to survive in an environment of limited resources. So, that is more like good solutions will be retained and bad solutions will be discarded.

So, we will look into how binary coded GA can be used for real variables. Since, we are going to talk about only binary coded GA in the first half of the session we will look into like how real variables can be handled, right.

(Refer Slide Time: 04:24)

Binary coded GA

- Real variables are encoded into binary variables (0 and 1).
- If the bit length (n) is specified for a variable, then exactly 2^n different solutions are possible between the bounds of the variable.

000
001
010
011
.

6

So, real variables, so are to be encoded into binary variables 0 and 1, right. So, for each real variable we will have to fix the bit length. So, if we specified the bit length n then we can have 2^n different solutions. So, for example, if I say bit length is 3, so that means, our string length is 3. So, with this 3 digits we can generate all the other combination 000, 001, 010, 011 and so on, right. So, like this there will be 2^n different solutions.

(Refer Slide Time: 04:56)

Binary coded GA

- Real variables are encoded into binary variables (0 and 1).
- If the bit length (n) is specified for a variable, then exactly 2^n different solutions are possible between the bounds of the variable.

Let a binary string of length (n) be used to represent a variable x with bounds $[x_{\min}, x_{\max}]$.

String (00000) will map to x_{\min} and (11111) will map to x_{\max} .

Decoded value of a string can be determined as

$$DV = b_n 2^{n-1} + b_{n-1} 2^{n-2} + \dots + b_2 2^1 + b_1 2^0$$

where binary string $s = [b_n \ b_{n-1} \ \dots \ b_2 \ b_1]$

Maximum decoded value of binary string is $(\beta 1)(2^n - 1)$ and the minimum is 0.

Bit	Weight	Value
0	2^4	0
0	2^3	0
1	2^2	4
0	2^1	0
0	2^0	0
Total		4

So, this is an example, right. So, if we take a binary string of length n , say 5 and if we use to represent a real variable x within the bounds x_{\min} and x_{\max} , right, so our string will be 0 0 0 0 0, there are 5 elements in this because our bit length is 5. So, all possible combinations of this 5 string is 2^5 , right. So, last number would be 1 1 1 1 1. So, this 0 0 0 0 0 will correspond to x_{\min} , whereas, this 1 1 1 1 1 will correspond to x_{\max} .

So, any value between this let us say this value 1 0 0 0 1 0, so if this is the binary number we can decode it, right. So, here what we will call is decoded value. So, decoded value is nothing but the decimal equivalent of this. So, if we are given this binary number, the way we determine the decimal equivalent of this binary is something that you would have come across, right. So, 0 into 2^0 , 1 into 2^1 , again 0 is the element, so 0 into 2^2 , 0

into 2^3 , for this 0 and 1 into 2^4 . So, this has to be 1, so 1 into 2^4 will be 16. So, this value turns out to be 18, right.

So, this is how we will be decoding a binary number into its decimal value or for today's session we will call this decimal value is as nothing, but decoded value, right. So, this gives the generic expression how to obtain a decoded value from a binary string. So, if the first element from this side is b_1 , the second element is b_2 and so on then this expression gives the decoded value, right.

So, we will start with 2^0 which is 1, 2^1 , 2^2 , 2^3 and all the way up to 2^{n-1} , right. So, maximum decoded value of binary string in this case is 31. So, if you decode this value 1 1 1 1 1 you will get 31, right and the minimum value is 0. So, if we decode this value will get 0 and if we decode this value we will get 31, right. The maximum decoded and the minimum decoded value is 31 and 0 if the bit length is 5. So, if the bit length is varied then obviously the number of possible combinations would be 2^n and the decoded value would be different, right.

(Refer Slide Time: 07:28)

Binary coded GA

➤ For a fixed bit length (n), the value of a real variable x within bounds $[x_{min}, x_{max}]$ is

$$x = x_{min} + \left(\frac{x_{max} - x_{min}}{2^n - 1} \right) DV$$

$$Precision = \frac{x_{max} - x_{min}}{2^n - 1}$$

➤ For evaluating fitness, x is used.

Example: Consider a variable x with bounds $[5, 30]$ to be represented by a 4 bit binary string.

We will be having 16 ($= 2^4$) different solutions between 5 and 30.

Let the binary string be $s = 0110$, then $DV = 6$

$(30-5) / (16-1)$

$$x = 5 + \left(\frac{30-5}{2^4-1} \right) 6 = 15$$

Precision = $25/15 = 1.67$

For higher precision, n should be increased.

$(0 \times 2^3) + (1 \times 2^2) + (1 \times 2^1) + (0 \times 2^0) = 6$

5 5 → 0000

5+1.67 ≈ 0001

5+2x1.67 2x1.67

So, let us see an example. If our bounds are let us say 5 and 30, right. So, there are we want to represent let us say now we have a real variable x , and we want to use binary coded GA. So, in this case first we will have to encode this real variable into a binary equivalent. So, first we will have to decide as to how many bits are we going to represent this real variable with.

So, in this case we have taken the bit size to be 4, right. So, if we have 4, then we have 2 power 4 combinations like 0000, 0001 and so on, if we proceed we will get 16 different solutions, right. So, now, this 16 different solutions are supposed to represent the numbers between 5 and 30, right. So, between 5 and 30 for a real variable there are infinite values, but here we have this 16 values only.

So, this 16 values are supposed to capture this space, right. So obviously, we will not be able to capture every number, right, so that is why we have something called as precision in this

one, right. So, precision is given by $x_{\max} - x_{\min}$ divided by 2^{n-1} . If our bit length is 4, 2^4 would be 16, 16 minus 1, 15, right. So, x_{\max} will be 30, x_{\min} will be 5. So, 30 minus 5, so this is $x_{\max} - x_{\min}$ divided by 16 minus 1. So, which is nothing, but 25 by 15, so the precision is 1.67.

So, we will start with 5 the next number that we can actually capture is 5 plus 1.67 and the next number that we will capture is 5 plus 2 into 1.67 and so on, right. So, we will not be able to capture any number which is greater than 5, but less than 6.67, right. So, that is a drawback of binary coded GA.

So, now, let us see given a binary string how do we construct the actual value. Let us assume that the binary string is 0 1 1 0, right. So, if we determine the decoded value of this, right. So, that would be 0 into 2^3 , so this is 0 into 2^3 , 1 into 2^2 , right, 1 into 2^1 and 0 into 2^0 , right. So, if we calculate this comes out to be 6. So, the decoded value is 6, right. So, we have seen that 0 0 0 0, four 0s would correspond to 5 and four values of 1, right that is the maximum permissible 4 digit string, right, so this is 30, right.

So, this is supposed to represent 5 this is supposed to represent 30. So, this string actually when decoded it comes out to 6, but the value that it represents can be determined from this formula, right. So, this formula says that the actual value of x is the minimum value of x . So, in this case 5 plus $x_{\max} - x_{\min}$. So, in this case it is 30 minus 5 divided by 2^{n-1} .

So, in this case it is 2^4 which is 16, 16 minus 1 into the decoded value. So, the decoded value here is 6, right, so if we do this calculation we will end up with 15. The decimal equivalent of 0 1 1 0 is 6, right that is the decoded value, but the corresponding real variable value for x_{\min} as 5 and x_{\max} as 30 is 15, right.

(Refer Slide Time: 11:00)

Binary to real values

$n = 3, 2^n = 8$ possible solutions

$x_{min} = 5, x_{max} = 30$

precision = $\frac{30-5}{2^3-1} = 3.57$

Binary string	Decoded value	Actual value
000	0	5
001	1	8.57
010	2	12.14
011	3	15.71
100	4	19.29
101	5	22.86
110	6	26.43
111	7	30

$n = 4, 2^n = 16$ possible solutions

precision = $\frac{30-5}{2^4-1} = 1.67$

$x = x_{min} + \left(\frac{x_{max} - x_{min}}{2^n - 1} \right) DV$

Binary string	Decoded value	Actual value	Binary string	Decoded value	Actual value
0000	0	5	1000	8	18.33
0001	1	6.67	1001	9	20
0010	2	8.33	1010	10	21.67
0011	3	10	1011	11	23.33
0100	4	11.67	1100	12	25
0101	5	13.33	1101	13	26.67
0110	6	15	1110	14	28.33
0111	7	16.67	1111	15	30

$5 + 3 \times 1.67$

So, let us see one more example, right. So, in this case x_{min} is 5, x_{max} is 30, right. If n is equal to 3 the total number of combinations which we have is 8, right 2^3 will be 8, so there are 8 possible solutions. So, those are given over here 000, 001, 010, 011, 100, 101, 110 and 111, right.

So, the decoded value of all this if we decode it will be 0, 1, 2, 3, 4, 5, 6, 7. So, the actual value is using this formula, x_{min} plus x_{max} minus x_{min} divided by $2^n - 1$ into decoded value which is given over here. So, this number will actually correspond to 5, three 0s would correspond to 5, three 1s would correspond to 30, 010 the decoded value is 2, right decoded value we need to plug it back into this expression to get the actual value of 12.14, right.

So, the precision in this case if we calculate it is upper bound minus lower bound, so 30 minus 5 divided by 2 power 3 minus 1, so that would be 7. So, we can capture the real variable 5, we can capture the real variable 8.57, we can capture the real variable 12.14 and so on, right. So, now, if we see the precision is very less, right, so we cannot capture some anything between 5 and 8.57 because the binary GA which we will initially steady is going to work only with the binary strings. If we want a higher precision then we will have to increase the bit length.

Suppose, let us say instead of 3 over here if we decide to have a bit length of 4. So, in this case same problem x_{min} is 5 x_{max} is 30. So, in this case we will have 2 power n, so 2 power 4 combinations which is 16 possible solutions. So, the 16 possible solutions are listed here 1, 2 3 and all those if you see these are the 16 possible combinations, right. Once we have this 16 possible combinations we can obtain their decoded value, decoded value or the decimal equivalent, right. So, this will be 0, 1, 2, 3, 4, 5, 6, 7 and so on up to 15, right. So, in this case the precision if you calculate using that formula, right $x_{max} - x_{min}$ divided by 2 power n minus 1 the precision is 1.67.

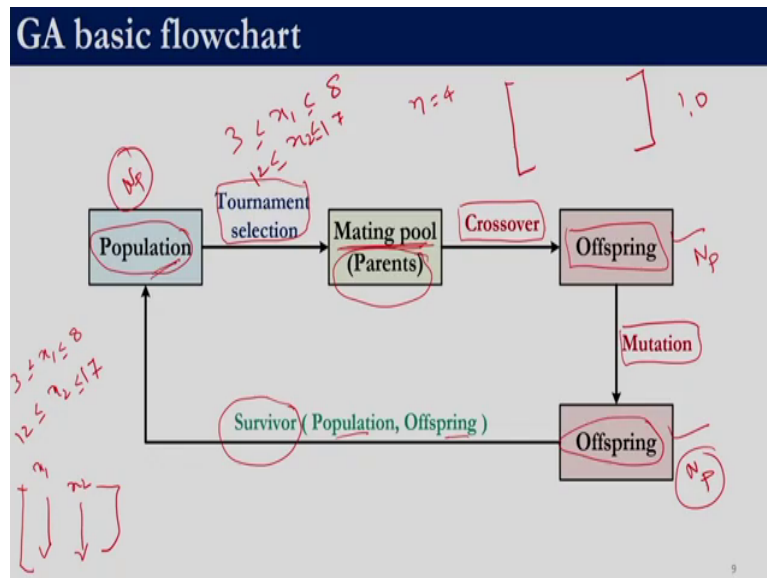
If you take this expression and plug the value of 0, right you will see that you the actual value that you get is 5. If you substitute the value of 1 for decoded value and x_{max} x_{min} as 30 and 5 we will get 6.67. If we substitute 6 as the decoded value x_{max} is 30 x_{min} as 5, right and 2 power 4 minus 1, if we compute that we will get 15. So, here we can see that we are getting an increase precision when compared to here.

So, here we will be able to capture the real variable value 5, will be able to capture the real variable value 6.67 8.33, 10 and so on, right. So, again this number if you want it is nothing but 5 plus 0 into 1.67, this 6.67 is nothing but 5 plus 1 into 1.67. So, if you want a higher precision we will have to use a higher bit length, right. So, if our problem has two real variables, right and if we choose to represent each of the variable with a string of bit length 4, right then 4 plus 4 our total string length will be 8, right.

So, as you can see as your number of variable increases and as we desire more and more precision the length of the population member would be very high, right. So, that is a

drawback of binary coded GA. But for the sake of completion and to aid in our understanding we will still look at binary coded GA first and then we will move on to real coded GA.

(Refer Slide Time: 15:07)



So, this is the basic flow chart of genetic algorithm, right. So, we will start with an initial population. So, for all other techniques let us say we had two variable x_1 and x_2 , right let us say x_1 was between 3 and 8 and x_2 was between 12 and 17, right. So, our initial population whatever we generated we had two columns, column one presenting variable x_1 and column two representing variable x_2 .

So, values of x_1 would be between 3 and 8 and values of x_2 would be 12 and 17, whereas, in the case of genetic algorithm let us say we choose to represent each variable by a bit length of 4, right. In this case we will have our population would consist of only 1s and 0s despite the

fact that our decision variables are between 3 and 8, and 12 and 17, right. So, the population would be binary, it will consist of only 0 and 1, right.

So, how do we move from real variables two binary variables and back from binary variables to real variable? Is something that we have previously seen. So, the population is going to be completely a binary population. For this population we will have something called as tournament selection. So, tournament selection is nothing but a competition between various solutions and the solution which wins, right will constitute what is called as this mating pool, right. So, we will have an initial population, we want to select good solutions from initial population.

So, what we will do is we will conduct tournament that is why we have the name a tournament selection, right. And from there we will select the good solutions which will constitute the mating pool. Solutions which are there in mating pool are also known as parents. Once we determine this mating pool we will employ a operator called as crossover to generate the offsprings, right. So, we have initial population, we will have some tournament, we will get a mating pool.

Mating pool is nothing, but the parents then those parents will undergo crossover to generate offspring and this offsprings again would undergo a process called as mutation, right to get modified offsprings over here, right. So, even this is also known as offspring, this is also known as offspring, right. So, once we have this offsprings we have the initial population and we have the newly generated offsrings. So, we will combine them and we will use this survivor operator, right.

So, right now we are not going into the details of each operator that we will see subsequently. Right now we are only looking at the basic flow chart, right, how genetic algorithm works, right. So, using the survivor we will select the best members. Let us say we start with this N_p members in the population initially our population size is N_p , right. We generate few offsprings let us say here also we have N_p and the same N_p have been modified over here, right.

So, basically we have two N_p solutions, N_p solutions over here and N_p solutions over here. So, out of these two N_p solutions we will select the best N_p solutions, so that will constitute the population for the subsequent generation. So, this is what we will be doing, ok. So, one thing you need to understand is there are various ways to determine this mating pool, we will be looking at only what is called as tournament selection, right there are other ways too.

Similarly, in for this crossover there are various types of crossover. We will be looking at only one type of crossover, right that is single point crossover. Similarly, in mutation there are various mutation strategies we will be again looking at one particular mutation strategy. From this itself you can understand that there are various variants of genetic algorithm, right. So, someone says that they are using genetic algorithm that is actually not the complete information, right.

So, genetic algorithm with which selection strategy, which crossover strategy, which mutation strategy, all those things need to be defined only then we would be able to understand what is the exact version of genetic algorithms someone has implemented.

(Refer Slide Time: 19:11)

Population initialization

- Let population size, $N_p = 6$ and the dimension of the problem, $D = 2$.
- Let the number of bits (n) for representing each variable be 5.
- Size of the population matrix will be $N_p \times nD$, (i.e., $= 6 \times 10$).
- Initial population is generated randomly with 0 or 1.

Equal n

	x_1					x_2				
$P =$	0	0	0	1	0	1	1	1	0	0
	1	0	0	1	1	1	1	0	1	0
	1	0	0	0	0	0	1	0	1	0
	1	0	0	1	1	1	1	1	0	0
	0	0	1	1	1	0	0	0	0	1
	1	0	0	0	1	0	1	0	1	0

Handwritten: 5 bits

Unequal n

	x_1					x_2						
$P =$	0	0	0	0	1	0	1	1	1	0	0	
	0	0	1	0	0	1	1	1	1	0	1	0
	0	0	1	0	0	0	0	0	1	0	1	0
	0	0	1	0	0	1	1	1	1	1	0	0
	0	0	0	0	0	1	1	0	0	0	0	1
	0	0	1	0	0	0	1	0	1	0	1	0

Handwritten: 7 bits, 5 bits

So, if the first step is to generate a population. Let us assume we are working with a problem which has two decision variables. So, D is equal to 2 and our population size is 6, right. So, this is a user defined parameter, right. So, whoever is solving that problem will have to take this call as to how many bits they are going to use to represent each variable. So, in this case let us say we are we choose to use 5 bits for both the variables.

So, x_1 will be represented by 5 bits, x_2 will be represented by 5 bits. These are actually real variables, right, but since binary coded GA works with only 0 and 1 we need to fix the number of bits we will be using for each variable. So, let us say in this case we choose 5, 5, right. So, our size of the population matrix which was previously $N_p \times D$, for all other techniques which we discussed so far the population size would be $N_p \times D$.

For here it will be $N \times p$ cross $n \times D$ because, for each variable we are using a bit length of 5 and there are two variables in this case, so 5×2 and there are 6 members. So, the size of the population matrix will be a 6×10 matrix. So, out of these 10 columns 5 columns will indicate the what is the value of x_1 and 5 columns will be used to determine the value of x_2 , right. So, initial population is generated randomly with 0 or 1, right.

So, this is how the initial population can possibly look, right. So, all the elements if we see are between 0 and 1. So, this has 5 columns column 1, 2, 3, 4, 5 and this also has column 1, 2, 3, 4 and 5. So, we have our n is 5, right 5 columns, we have two variables, so this is 10. Number of chromosomes or the solutions are 6, so the size would be 6×10 , right

Suppose we choose to use unequal bit length, right, so for x_1 let us say we choose a bit length of 7, right 1, 2, 3, 4, 5 6 and 7 and for x_2 let us say we take a bit length of 5 and the number of solutions is still 6, right. So, in this case the dimension of the problem will be 12 columns, 7 columns for the first variable and 5 columns for the second variable, right. So, there will be 12 columns and there will be 6 rows. Despite the fact that we have only two variable problem, we will have to work with 12 columns because, we have converted a real problem into a binary problem.

For this population member let us say we want to determine the fitness function value, right. So, what we will do is we will first find out the decoded value. Decoded value is nothing, but the decimal equivalent of this, right. So, the decimal equivalent of this one, right. That is the decoded value of x_1 . Depending upon the bounds of x_1 x_{min} and x_{max} of the first variable we will calculate what is the actual value of the variable x_1 .

Similarly, we will use this 5 elements, right to find the decoded value of the second variable. Once we have the decoded value, we will use that expression $x_{min} + x_{max} - x_{min}$ divided by $2^{power\ n} - 1$ into decoded value to determine the actual value of x_2 . Once we have the actual values of x_1 and x_2 , we will plug it back into the fitness function and determine what is the fitness of this solution 1.

(Refer Slide Time: 22:38)

Reproduction operator

- Identify good (usually above average) solutions in the population.
- Eliminate bad solutions from the population so that multiple copies of good solutions are considered for variation operator.
- Group of solutions selected through reproduction operator comprise the mating pool.
- **Reproduction:** Selection of good solutions before variation operator.
- Reproduction/selection operators reduces the diversity of the population.

Slides of Dr. Deepak Choudhary, MIT 685: Evolutionary computation, IIT Guwahati 11

So, now let us look into the reproduction operator. So, the use of this reproduction operator is to identify good solutions, right. Let us say we have N_p solutions and we still want N_p solutions, right, but the good solutions in this original population need to have multiple copies over here and the bad solutions should get a poor representation. So, this is going to be our mating pool, right.

So, this mating pool is what it is actually going to be used in the crossover operation, the next operation. So, to perform crossover we do not want the entire population, but we are interested only in good solutions, right. So, we will employ this reproduction operator to identify good solutions which will constitute the mating pool and the mating pool will undergo crossover.

So, a property of this reproduction or selection operator is that it reduces the diversity of the population, so we give more importance to the good solution. The solutions which are not good do not find a representation in the mating pool. So, that way it reduces the diversity of the population.

(Refer Slide Time: 23:42)

Tournament selection

- Pool size: number of members required to be in the mating pool (usually N_p).
- Tournament size (k) number of members that participate in a tournament (commonly $k = 2$).
- Tournaments are played between 'k' members and the member with best fitness is selected for mating.
- N_p number of tournaments has to be played.

Let $k = 3$, the solutions selected randomly from the population ($N_p = 6$) be {4, 2, 6} and their fitness $f_4 = 27$, $f_2 = 89$ and $f_6 = 12$.

For maximization problem: Solution 2 will win.

For minimization problem: Solution 6 will win.

12

To one of the very popular reproduction operator is the tournament selection operator. So, in tournament selection operator we need to define something called as pool size, right. So, pool size will tell us how many members do we desire in the mating pool. Let us say if we have N_p population initially. Let us say we desire N_p members in mating pool also, not all the members should come into the mating pool, but the good solution should have multiple copies and the bad solutions should have a poor representation, right.

So, we need to fix the pool size. So, usually the pool size is N_p , right and then we need to fix a tournament size, right. So, usually the tournament size is 2, right. So, what we will do is tournament selection depending upon this number k , let us say we take 2 as tournament size. So, we will pick 2 members, we will see which has better, whichever solution is better copy of that would go into the mating pool, whichever solution is not better that will not be put into the mating pool.

Again, there will be a competition between two other solutions, whoever is the winner will find the place in the mating pool and the loser will not find its copy in the mating pool. So, this is called as a binary tournament because two members are playing. So, in binary tournament since two members are playing we will get one solution at a time, right.

So, if we desire N_p solutions, right then we require N_p tournament. Let us say we have initial population size is 100, we decided that our mating pool should also have 100 solutions, right. So, from each tournament we will get 1. So, we need to conduct 100 such tournaments, so that we get the required number solutions in the mating pool. This can be any number, right. The competition can be between any numbers. So, if k is 3 and let us say we have 6 solutions, right, so and let us say we randomly pick up these 3 solutions 4, 2 and 6.

So, what do we mean by conducting a competition between 4, 2, 6 is, the solution which has the best fitness will be the winner. So, if our problem is maximization, right and the fitness is 27, 89 and 12 then obviously, 89 is the winner, right. So, solution 2 wins and a copy of solution 2 will be placed in the mating pool. If the problem was to be a minimization problem then out of 27, 89, 12, 12 is the minimum value. So, in this case we say that solution 6 is the winner and a copy of the winner would be placed in the mating pool. So, this is how we will prepare the mating pool.

(Refer Slide Time: 26:17)

Binary tournament selection

➤ If tournaments are played systematically, each solution will have two chances to play.

- best solution will have two copies.
- worst solution will never be selected.
- other solutions can have 0, 1 or 2 copies.

➤ As the tournament size is increased, the selection pressure of each solution increases.

k	Remarks
2	Worst solution will not be in the mating pool
3	Worst and second worst solution will not be in the mating pool
n	The worst n-1 solutions will not be in the mating pool

5 9

13

So, a property of binary tournament selection is that if the tournaments are played systematically, systematically meaning each solution getting two chances to play. If you think about it the best solution in the population gets two chances to play. So, if the best solution is getting two chances to play, it is going to win both the times because it is the best solution, right. No matter with whom it is competing it is going to win and it is playing twice. So, the mating pool will definitely have two copies of it.

Now, think about the worst solution. So, despite the fact that we will be playing two competitions with the worst solution also, in both the cases it will lose. So obviously, the mating pool will not have any copies of the worst solution. All other solutions will have a either 0 copies, 1 copy or 2 copy depending with whom they are playing. But the best solution will definitely find 2 copies and the worst solution will find 0 copies, that much is assured.

Rest of the solutions depending upon with whom they are competing they can have either 0 copies, assuming that they fail both the times, both the times they are playing with someone better than them or they can play one tournament with someone who is better than them and one tournament with someone who is not better than them. In that case, they will be in one tournament and it can happen that in both the tournaments the solution though it is not the best solution is competing with solutions which are poorer than itself. So, in that case it will win in both the cases.

So, the other solutions will have 0, 1 or 2 copies depending upon whom they are playing. This solution will definitely have two copies and the worst solution will never be selected, right. So, if we increase the tournament size, right. So, if the tournament size is 2, like if 2 players are to play in a tournament then the worst solution will not be in the mating pool.

If 3 solutions are used to play, right then you if you analyze it worst and second worst solution will never be in the mating pool, because they both will keep losing and every tournament consists of 3 people every time they will lose, right. The worst solution as well as the second worst solution, right that way neither of them will find a copy.

So, if you generalize it if our tournament size is n , right then the worst n minus solution will not be in the mating pool. So, if we take that every competition will require 5 members to play it, right. So, in that case the worst 4 solutions will not find a place in the mating pool, right.

But for the rest of this discussion we will be sticking to only k is equal to 2 that our tournament size is fixed, 2 members will be playing in a particular tournament. So, subsequently after looking at the crossover and mutation operator we will also solve an example, right. So, if things are not clear over here we hope you will be able to pick it up over there, right.

(Refer Slide Time: 29:17)

Variation operator: Single point crossover

- Responsible for generating offspring.
- Two binary strings are chosen randomly from the population to perform crossover.
- Some portion of parent strings are exchanged to generate two offspring.
- Crossover site is a random integer chosen between 1 and nD .
- Crossover of two parents occurs with a probability (p_c).

Let $n = 4$, $D = 2$, crossover site = 3 and the randomly chosen parents be

$parent_1 = [1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1]$	$offspring_1 = [1 \ 0 \ 0 \ 1 \ 0 \ 1 \ 1 \ 1]$
$parent_2 = [0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 1 \ 1]$	$offspring_2 = [0 \ 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1]$

14

So, the variation operator that we will be studying as part of this course is single point crossover, right. So, this variation operator is responsible for generating the offspring. Remember the generic flow chart we had a population, from that population we employed the tournament selection to get a mating pool, right over mating pool we will be applying the crossover to get the offspring. So, two binary strings are chosen randomly from the population to perform crossover.

In single point crossover what will happen is some portion of the parent strings are exchanged to generate two offsprings. So, we need to fix something called as crossover site to employ this crossover. So, crossover site is a random integer chosen between 1 and nD , D is the dimension of the problem, n is the number of bits we have chosen to represent the real variables, right.

So, as we discussed earlier we may have 11 columns. So, in that 11 columns from 1 to 11 we need to randomly decide on a integer that integer is known as crossover site. And crossover of two parents is going to occur with the probability. From the mating pool we will select two members, we will generate a random number, the random number is less than what is called as crossover probability those two solutions will undergo crossover else they will not undergo crossover.

Let us look at an example. So, if our bit length is 4 our decision variable is 2, then the string length will be n into D , right. So, in this case it has to be 8, our string length will be 8, and if we decide that the crossover site is 3 again between 1 and 8 we are supposed to generate a random number. So, if we decide that random number to be 3, in this case if the parents are this, right. So, there are 8 strings 1, 2, 3, 4, 5 6, 7, 8, these two parents are randomly selected from the mating pool.

So, now, we have a mating pool, from the mating pool we have randomly selected to parents. So, these are the two parents and we have decided the crossover to be 3. So that means, we will make a cut over here, right. So, if we call this as head of parent 1 and this as tail of parent 1, this as head of parent 2 and the tail of parent 2, then these two tails will be swapped. That will help us to create new solutions. So, those are called as offspring, offspring 1 and offspring 2.

So, here if we see head of parent 1, 100 and the tail 10111, so 100, 10111. So, this is nothing, but swapping the tails. Similarly, the second offspring is 010 which is nothing, but the first part of the parent and the tale of the first parent 00101, right. So, this is called as single point crossover. As you can see it is very straightforward, right we choose a location at which we want to cut the strings and swap the tails, so that will help us to generate new solutions. So, this is single point crossover, right.

(Refer Slide Time: 32:10)

Variation operator: Bit-wise mutation

- Change the bit 1 to 0 and vice versa with a mutation probability (p_m).
- Single solution is involved in mutation.
- Every member in the population can potentially undergo mutation.
- For each bit in a string, probability for mutation is checked.

Let $p_m = 0.3$ and the parent string be

→ offspring = [1 0 1 0 1 0 0 0]

Let the random number for each bit be $r = [0.2 \ 0.5 \ 0.6 \ 0.8 \ 0.7 \ 0.1 \ 0.4 \ 0.9]$.

Mutation occurs for 1st ($r_1 < p_m$) and 6th ($r_6 < p_m$) variable and the offspring is

offspring = [0 0 1 0 1 1 0 0]

15

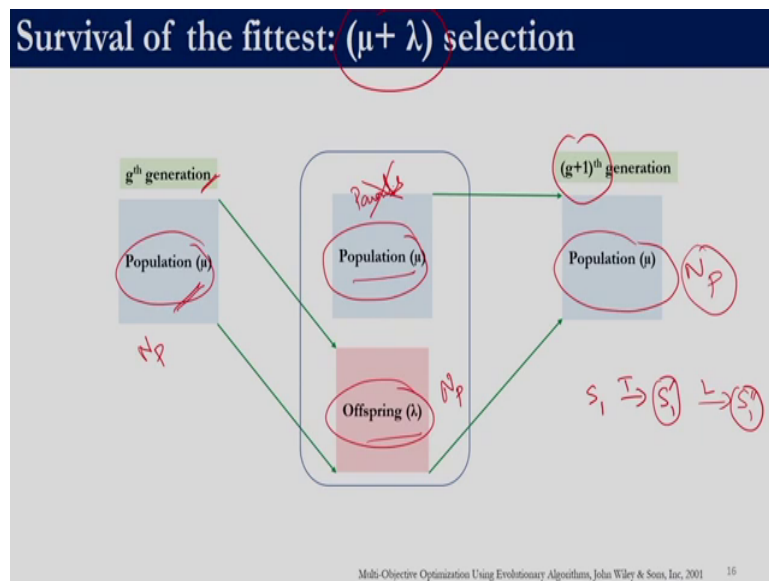
In mutation what we will do is we will change the bit 1 to 0 and vice versa. So, for every bit we will generate a random number. If that random number is less than the mutation probability, right mutation probability is to be again set by the user. So, if the random number is less than mutation probability we will mutate that particular bit. So, if that particular bit has value of 0 we will convert it into 1, and if it has a value of 1 we will convert it into 0.

So, this shows an example if the mutation probability is 0.3 and if our offspring, right. So, we are calling this offspring, because this has come from the crossover. After crossover what we get is offspring, right. So, mutation is to be performed on the offspring, right. So, here we have 1, 2, 3, 4, 5, 6, 7, 8 our string length is 8. So, here we need generate 8 random numbers between 0 and 1, right except for the first one and the sixth one all the other random numbers are greater than the mutation probability, right.

So, we need to identify the ones which are less than mutation probabilities. So, in this case it happens to be 1 and 6, right. So, in this bit length except for 1 and 6 we will retain all those things. So, these two 0s are to be retained this 101 is to be retained, this 0 is also to be retained, right and these two elements this first and sixth element, if it is 1 it has to be replaced by 0, and if it is 0 it has to be replaced by 1. So, this is a new offspring, right. So, again it is a very simple operation to understand as well as to implement.

So, now, we have the initial population which underwent tournament selection. We got mating pool, the mating pool underwent crossover we obtained the offspring, the offsprings underwent mutation, we have offspring.

(Refer Slide Time: 34:16)



So, now we have the initial population not the parents, but the initial population and we have the offspring. Let us say in any particular generation we had the initial population, right and

then we performed crossover and mutation to get this offspring, right. So, let us call the initial population as μ and the offsprings we generated as λ . Now, what we need to do is we need to combine the population and the offspring, right

Remember, this is not the parents, right this is not the parents. Parents is what we got at the end of selection which was there in the mating pool, right. So, we obtain the parents, we did crossover, we did mutation, we got cause offspring, right. This offspring which we generated in the current generation and the population which was the basis on which these offsprings got generated, those are to be combined, right, so we need to combine them and we need to select the best members, right.

So, if our population size is N_p , let us say we generated N_p new solution. So, totally we have two N_p solutions. So, out of these $2N_p$ solutions we need to select the best N_p solutions, right that will be the population for the subsequent $g + 1$ th generation, right. So, this has to be N_p .

In the strategy that we will see both for binary coded GA and the real coded GA here we will have N_p offsprings, right, but it is not necessary that we generate always N_p offsprings. We maybe we may generate $3N_p$ offsprings or N_p by 2 offsprings depending upon the crossover and mutation strategies that we employ, right. So, but ultimately for steady state GA we need to still select the best N_p solutions, right. So, that way the population for every generation is of the same size, right.

So, this strategy is more widely known as μ plus λ strategy, right. So far if you see we were employing greedy selection strategy. Let us say in t_1 we had solution one let us say integer face we were able to get a better solution S_1 prime, right. Then we compared if S_1 prime is better than S_1 or S_1 is greater than S_1 prime, depending upon which ever one is better that underwent the learner phase, right.

And let us say we obtained S_1 double prime, then it replaces, if this is better it replaces this solution, right. So, for every solution we were deciding then and there whether it has to be included in the population or not. Over here we generate the entire set of offsprings, then

combined with the population the initial population, right and then we pick up the best N p solutions. This is unique to genetic algorithm.

There are various other materialistic techniques which employ mu plus lambda, but among the materialistic techniques discussed so far in this course it is only genetic algorithm which employs this mu plus lambda selection strategy, ok.

(Refer Slide Time: 37:10)

Working of genetic algorithm: Sphere function

Consider $\min f(x) = \sum_{i=1}^2 x_i^2; 0 \leq x_i \leq 30, i=1,2$ $f(x) = x_1^2 + x_2^2$

- Decision variables: x_1 and x_2
- Step 1: Fix the population size, maximum iterations, bit length, crossover probability, mutation probability
 $N_p = 4, T = 10, n = 4, p_c = 0.8, p_m = 0.3$

$x = x_{min} + \frac{x_{max} - x_{min}}{2^n - 1} (DV)$
 $x = 0 + \frac{30 - 0}{2^4 - 1} (DV) = 2 \times DV$
- Step 2: Generate random binary solutions

x_1 x_2

1	0	0	1	1	1	0	0
0	0	1	1	0	1	1	1
0	1	0	0	0	1	0	1
0	1	1	0	1	0	1	0

$P =$

$P_D =$

9	12
3	7
4	5
6	10

$P_A =$

18	24
6	14
8	10
12	20

$f =$

900
232
164
544

17

So, now let us look into an example, right. So, is the same sphere function, right instead of four variables we will take a two variable problem. We will take we are taking two variable problem because as you would have realized once we fix the bit length the number of columns in the population is going to increase substantially, right. So, that is why we are just showing it for two variable problem, so that it is much easier to calculate things, but whatever we are discussing can be implemented for any number of decision variable.

Let us take the bounds to be 0 and 30. So, far we were having 0 and 10, we have taken 0 and 30 so that we get a better representation of the solutions. So, the objective function is $x_1^2 + x_2^2$, the decision variables are x_1 and x_2 , and the bounds are 0 to 30, right.

So, first for binary coded GA we need to fix the population size, the maximum number of iterations, these two are similar to all the 5 techniques which we have discussed. We need to fix the bit length, right. So, in this case we are taking a bit length of 4 that each variable will be represented by a 4 bit binary string. We need to fix the crossover probability to be used in single point crossover and we need to fix mutation probability, right.

So, in this case we have fixed crossover probability to be 0.8 and mutation probability to be 0.3. So, crossover probability is usually set to a high value as compared to a mutation value. So, we want more solutions to undergo crossover and only few solutions to undergo mutation.

So, the first step is to generate a binary population, right. So, for example, here the length would be 8, right because the number of columns n into D . We have two decision variables and for each decision variable we decided to use 4 bit string, right. So, these 4 values are going to correspond to x_1 , these 4 values are going to correspond to x_2 . Similarly, for the rest of the population number, right.

So, here we have taken a population size of 4, so that is why we have 4 rows and we have 8 columns because of two variables and each variable represented by 4 bit string, right. So, now, how do we calculate the fitness of this? To calculate the fitness of this first we need to find out what is that decoded value. Decoded value is nothing, but the decimal equivalent of this. So, if we find the decimal equivalent of this is $0 \times 2^0 + 0 \times 2^1 + 1 \times 2^2 + 1 \times 2^3$. So, this is $8 + 4$, so this works out to be 12.

So, the decoded value is 12. So, the decoder value is nothing but the decimal equivalent which all of us should be comfortable with. So, once the decoded values are found out, remember

this decoded values are not to be directly plugged into the fitness function, right. So, from this decoded value we need to find out the actual values which correspond to the real variables, right. So, for that we need to use this expression.

So, in this case this expression if we plug in all those values. So, $x_{\min} = 0$ plus $x_{\max} = 30$, $x_{\min} = 0$ divided by 2^4 because our n is 4 minus 1, so into DV. So, here if we simplify this we get this nice expression $2^{\text{into DV}}$ and this is valid for both variables, right. It is valid for both the variables because both the variables are in the same bound 0 to 30, and both variables are represented by 4 bit string, right. So, if that was not the case then you would get a different expression like this for both the variables, for x_1 you will have a different expression, for x_2 you will have a different expression.

So, now, that we have decoded the values over here and found the actual values over here this is what is to be plugged into this fitness function $x_1^2 + x_2^2$ and we need to determine the fitness of all the 4 members. So, we had an initial binary population, we converted into decoder population, we found the actual value and then we found the fitness. So, once we have found the fitness next step is to implement the tournament selection.

We implement the tournament selection to determine the mating pool, right. So, now, we need to fix the pool size. So, in this case we will fix the pool size to be the same as population size which is 4, right.

(Refer Slide Time: 41:28)

Selection: Tournament selection

- Step 3: Select two random candidates for tournament.
Let the two candidates be
 $P_2 = [6 \ 14] \quad f_2 = 232$
 $P_4 = [12 \ 20] \quad f_4 = 544$
- Step 4: Compare fitness to select winner.
 $f_2 < f_4 \rightarrow P_2$
- Step 5: After four tournaments
Best solution has two copies.
Worst solution has zero copies.

$P =$

1	0	0	1	1	1	0	0
0	0	1	1	0	1	1	1
0	1	0	0	0	1	0	1
0	1	1	0	1	0	1	0

$P_1 =$

18	24
6	14
8	10
12	20

$f =$

900
232
164
544

Mating pool

$P_2(232)$	$P_4(544)$	$P_1(900)$
$P_2(232)$	$P_4(544)$	$P_4(544)$
$P_1(900)$	$P_3(164)$	$P_2(232)$
$P_3(164)$	$P_3(164)$	$P_3(164)$

So, this is what we have so far, right. So, we need to randomly select two solutions, right. So, let us say we select the second solution and the fourth solution, right P 2 and P 4 and their corresponding fitness function f 2 and f 4. So, these two solutions are going to compete and the winner is going to be put into the mating pool.

So, this is our mating pool, right. So, between the competition between P 2 and P 4 the fitness function for P 2 is 232, the fitness function for P 4 is 544. So, the winner obviously, would be P 2 because we are looking at a minimization problem, the solution with minimum fitness will go into the mating pool. So, P 2 is our first member in the mating pool, right so but we want 4 members, right. And P 2 and P 4 have competed once, so let us have another competition between P 1 and P 3, right.

So, P 1 has a fitness function of 900 and P 3 has a fitness function of 164. So, the winner obviously, would be P 3, right. So, this is our second competition, right. So, similarly if we conduct two more competitions between P 1 and P 4 and P 2 and P 3, the winner would be P 4 and P 3 respectively because P 4 has 544 and P 1 has 900. So, P 1 would lose out similarly P 3 has 164, P 2 has 232, so P 3 will find a place in the mating pool. So, our initial population size was 4, we wanted 4 members in the mating pool. So, we conducted for competitions 1, 2, 3 and 4, right.

So, here if we see each member got participated in the two competitions. So, for example, P 1 this is the first competition this is the second competition for P 1. For P 2 this is the first competition this is the second competition. For P 3 this is the first competition second competition. P 4 this is the first competition and this is the second competition. So, this we have played systematically, right each member participates in two tournaments, right and the winner will be placed in the mating pool.

And you need to make sure that let us say the first tournament was between P 1 and P 2, so the next tournament should not be between P 1 and P 2. It will still satisfy the criteria that each member plays two tournaments, but that has to be randomly selected, right. So, at the end of the mating pool here if we see some among these fitness function 164 was the best solution, right and 900 was the worst solution, right. So, P 3 is the best solution.

So, here if we see we have two copies of P 3, right. Whereas P 1 which is the worst solution because it has the highest fitness function value 900 does not find a place in this, right. So, that is what we had studied earlier. That the best solution will definitely occur twice, the worst solution will definitely not occur. Depending upon many other parameters the rest of the solution can occur either once, twice or they may not even occur, right.

(Refer Slide Time: 44:32)

Crossover: Single point crossover

- Step 6: Randomly select a pair of parents for crossover.

Let the two parents be

Parent₁ = [0 0 1 1 0 1 1 1]

Parent₂ = [0 1 0 0 0 1 0 1]

$p_c = 0.8$

Parent = $\begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}$

P₂
P₃
P₄
P₃
- Step 7: Generate a random number to check if crossover is to be performed.

Let $r = 0.2$

$r < p_c \rightarrow$ perform crossover ✓
- Step 8: Select a random crossover site.

Let $r = 3$

Parent₁ = [0 0 1 | 0 1 1 1]

Parent₂ = [0 1 0 | 0 0 1 0]

offspring₁ = [0 0 1 | 0 0 1 0]

offspring₂ = [0 1 0 | 1 0 1 1]

So, now we have completed the tournament selection let us say implement the crossover operation, right. So, this is our mating pool P 2, P 3, P 4, P 3, right. So, there are two copies of P 3 and one copy of P 2 and one copy of P 4, right. So, here if we see a particular solution would be repeated twice. So, for example, this one and this one are identical. So, this is actually corresponding to this P 3. So, two copies are there.

So, this is not the population. Remember this is not the population, this is the parent or the mating pool, right. And the rest of this two solutions are the P 2 P 4, so we are we have taken a crossover probability of 0.8. So, after crossover we require 4 solutions, right. So, each crossover will give us two solutions. So, if you have looked at the single point crossover carefully, one crossover operation will give us two solutions, so here we will implement only two crossovers, but we will still get 4 solutions.

So, let say we randomly selected parent 1, parent 2, right and we need to first generate a random number, right. If this random number is less than the crossover probability we need to perform cross over. So, in this case 0.2 is less than 0.8, so we will perform cross over. If we have to perform crossover we need to select a random crossover site.

So, in this case let the random number be 3. So, we will be cutting at the third string, after the third string and then swapping it, right. So, 0 0 1, 0 0 1, 0 0 1 0 1, 0 0 1 0 1, right. So, this is part A over here and part B over here. That is what is our first offspring. And this lower case a and the lower case b will form the second offspring, right 0 1 0, 0 1 0, 1 0 1 1 1, 1 0 1 1 1. So, now, we have determined the offspring, right. We do not need to evaluate the fitness function of this yet because the mutation operator is still pending, right.

(Refer Slide Time: 46:28)

Crossover: Single point crossover

- Step 9: Randomly select a pair of parents for crossover

Let the two parents be

Parent₁ = [0 1 1 0 1 0 1 0]

Parent₂ = [0 1 0 0 0 1 0 1]

$p_c = 0.8$

0	0	1	1	0	1	1	1
0	1	0	0	0	1	0	1
0	1	1	0	1	0	1	0
0	1	0	0	0	1	0	1
- Step 10: Select a random number to check if crossover has to be performed

Let $r = 0.6$

$r < p_c \rightarrow$ perform crossover
- Step 11: Select a random crossover site

Let $r = 5$

0	1	1	0	1		0	1	0
0	1	0	0	0		1	0	1

offspring₁ = [0 1 1 0 1 | 1 0 1]

offspring₂ = [0 1 0 0 0 | 0 1 0]

$o_3 = p_3$

$o_4 = p_4$

20

So, now if we go for the second crossover. We had to do two crossovers, right because we wanted 4 solutions, we have got two offerings, we need two more offspring. Previously, we had taken one and two now we are taking 3 and 4. So, this has to be done randomly, right. So, let us say we select now parent 3 and parent 4. Again, we need to generate a random number 0.6, let us say in this case this is still less than p_c . So, we need to perform the crossover, right.

So, since we need to pass on the crossover, we need to determine the random site, right. So, in this case let the random site be 5, right 1, 2, 3, 4, 5. So, after the fifth position we cut the solution and then swap the tails, right. So, the first part would be identical, right in both the cases, right. This two would be interchanged, right so 1 0 1, 0 1 0.

Now, we have 4 offspring. So, previously we generated two offspring now we have generated two offspring. So, we have 4 offspring. So in this example for both the crossover we happen to do a crossover, right. Like the random number which we generated happened to be less than crossover probability. So, we did a crossover single point crossover. If it had happened, let say this number this random number 0.6 instead of 0.6 if it had happened to be 0.9, right then it is not less than the crossover probability.

So, we do not need to do a crossover, right in that case offspring 3 will be nothing but parent 3 and offspring 4 will be nothing, but parent 4. So, if the random number is less than crossover probability we need to do the single point crossover, if the random number is greater than or equal to the crossover probability then we need to copy the parent themselves as offspring.

(Refer Slide Time: 48:17)

Mutation: bit-wise mutation

- Step 12: Select first offspring for mutation
 $\text{offspring}_1 = [0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 1]$
- Step 13: Select random numbers to check if mutation is to be performed
Let $r = [0.6 \ 0.1 \ 0.5 \ 0.4 \ 0.9 \ 0.7 \ 0.3 \ 0.8]$
 $r < p_m \rightarrow$ perform mutation

$\text{offspring}_1 = [0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 1] \Rightarrow \text{offspring}_1 = [0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 1]$

0	0	1	0	0	1	0	1
0	1	0	1	0	1	1	1
0	1	1	0	1	1	0	1
0	1	0	0	0	0	1	0

21

So, now that we have completed crossover the next step is bitwise mutation, right. So, every member has to undergo mutation, right. So, for the first member, so this is the first member we need to generate 8 random numbers because the string length is 8, right. So, let us say these are our random numbers and our mutation probability is 0.3. So, in this vector we need to identify all the elements which are less than 0.3, right. So, in this case it happens that only this is less than 0.3.

So, except for the second element, right, this is the original offspring, this is the new offspring. So, except for the second element all the rest of the strings all these strings have to be copied as such and this second string has to be converted into 1. So, if it is 0, since it is 0 we converted it to 1, if it had been 1 we would convert it into 0. That is what is mutation, right.

(Refer Slide Time: 49:09)

Mutation: bit-point mutation

$p_m = 0.3$

▪ Step 14: Perform mutation for all offspring

	Offspring	Random number for mutation	New offspring
O ₂	[0 0 1 0 1 1 1 1]	[0.5 0.6 0.3 0.5 0.4 0.6 0.3]	[0 0 1 0 1 1 1 1]
O ₃	[0 1 0 1 1 0 0 0]	[0.4 0.5 0.8 0.5 0.7 0.4 0.2]	[0 1 0 1 1 0 0 0]
O ₄	[0 1 0 0 0 1 0 0]	[0.8 0.6 0.3 0.9 0.7 0.5 0.4 0.6]	[0 1 0 0 0 1 0 0]

x_1

0	1	1	0
0	0	0	1
0	1	0	0
0	1	0	0

x_2

0	1	0	1
0	1	1	1
1	1	0	0
0	0	1	0

$x = x_{min} + \frac{x_{max} - x_{min}}{2^n - 1} (DV)$
 $\Rightarrow x = 2 \times DV$

$n = 4$
 $x_{min} = 0$
 $x_{max} = 30$

O_1

12	10
2	14
8	24
8	4

$f(x) = x_1^2 + x_2^2$

244
200
640
80

$12^2 + 10^2 + 2^2 + 14^2 + 8^2 + 24^2 + 8^2 + 4^2 = 244 = 6$

Similarly, for the other 3 strings O 2, O 3, O 4, these are the actual offspring which we obtained at the end of crossover, right. So, we have this random numbers for each of this, right. So, we need to generate this random numbers, for each offspring we need to generate 8 random numbers and then wherever it is less than 0.3 our mutation probability which is a user defined value wherever it is less than 0.3 we need to change those bits.

So, here in this case we have these 3. So, this would get changed to a 0, right, this would get changed to a 0 and this would get changed to a 0. So, here if you see this is 1, this is 0; this is 1, so this is getting changed to 0 and the last one is 1,so this will get changed to 0. None of the elements in this are less than 0.3, right. So, none of this strings would undergo mutation, so that entire string would get repeated, right.

So, now, we have this offspring, right. So, now we need to find the decimal equivalent of this, right. So, decimal equivalent of this you we need to find which is nothing, but the decoded value. Once we have the decoded value we need to plug it into this expression, right and then calculate the actual value, so that should be straightforward, right. So, the decimal equivalent of this would be $0 \times 2^0 + 1 \times 2^1 + 1 \times 2^2 + 0 \times 2^3$, right. So, this is nothing, but $2 + 4$, so this will be 6, right.

So, the decoder value of this is 6, right. In this case because the lower bound is x_{\min} is 0 and x_{\max} is 30 and n is equal to 4, right if we plug this into this expression we will get this two into decoded value. So, the decoded value is 6. So, the actual value is 12. So, if you decode this you should get 5 because we have a 10 over here, right. So, now, we have the actual values denoted by O suffix A, O stands for the offspring, A stands for the actual values not the decoded values, right. So, once we have this actual values we can calculate the offspring, right.

So, we had initial population, we employed tournament selection to get the mating pool, over the mating pool we employed single point crossover they got offsprings, on the offsprings we employed mutation to get the mutated offsprings or for simplicity we just call that also offspring. So, we have now the initial population and this offspring, right.

(Refer Slide Time: 51:52)

Survival

▪ Step 15: Combine all the solutions and select best N_p ($N_p = 4$) solutions

$$P = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

$$f = \begin{bmatrix} 900 \\ 232 \\ 164 \\ 544 \end{bmatrix}$$

$$O = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$f_o = \begin{bmatrix} 244 \\ 200 \\ 640 \\ 80 \end{bmatrix}$$

$$P_{combined} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$f_{combined} = \begin{bmatrix} 900 \\ 232 \\ 164 \\ 544 \\ 244 \\ 200 \\ 640 \\ 80 \end{bmatrix}$$

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

$$f = \begin{bmatrix} 80 \\ 164 \\ 200 \\ 232 \end{bmatrix}$$

Population for the next iteration

So, this was our initial population which we had started with and this is now our current offspring. So, we are supposed to combine both of this. So, this is the combined population and this is the combined fitness function value. And then we need to sort this vector, sort this vector to determine the best 4, right. Because our population sizes 4, right in this case we generated 4 offspring, so this will be 8 values = 1, 2, 3, 4, 5, 6, 7, 8. So, these are 8 values, so but we require only 4 of them, right.

So, what we will do is we will select the best 4, right. So, the best 4 in this case happens to be this 80, 164, 200 and 232. So, only those 4 solutions should be retained, right 232, 164, 200, 80 and 200, 80 and 200. Remember when you are finding out the best solution over here you need to pick the corresponding solution vector, right. So, these 4 have to be taken and these 4 are to be discarded, right. So, this is the new population, right.

So, for again for this population we will do tournament selection. Once we do tournament selection will do crossover, after crossover mutation, after mutation we will employ a mu plus lambda strategy where you will combine the offspring as well as the original population to determine the population for the next iteration or generation. This is the working of binary coded genetic algorithm. So, before we conclude let us quickly look at the pseudo code of it.

(Refer Slide Time: 53:28)

Pseudocode

Input: Fitness function, lb, ub, N_p , T, p_c , p_m , k

1. Initialize a random population (P) of binary string (size: $N_p \times nD$)
2. Evaluate fitness (f) of P ← FE = N_p

for t = 1 to T

Perform tournament selection of tournament size, k

for i = 1 to $N_p/2$

Randomly choose two parents

if $r < p_c$

Select the crossover site

Generate two offspring using single-point crossover

else

Copy the selected parents as offspring

end

end

end

for i = 1 to N_p

Generate nD random numbers between 0 and 1

Perform bit-wise mutation of i^{th} offspring Addition

Evaluate the fitness of offspring ← Max FE = N_p

end

Combine population and offspring to perform $(\mu + \lambda)$ XPP

end

In one iteration, max #FE = N_p

For T iterations, max #FE = $N_p + N_p T$

crossover site = 3

parent ₁ =	0 0 0 1
parent ₂ =	0 1 0 1
Offspring ₁ =	0 0 0 1
Offspring ₂ =	0 1 0 0

Generation

8

D=2
n=4

$r_i < p_m$

parent =	1 0 1 0 1 1 0 0
offspring =	1 0 1 0 1 0 0 0

Survival of fittest

So, we need to provide the fitness function lower bound, upper bound, which comes from the problem itself. We need to decide on the number of population that we are going to employ and a termination criteria, usually it is the number of iterations, right. We need to decide how many bits are we going to represent a variable with, right. For simplicity we have given n over here, but each variable can have different weights.

So, for example, variable 1 can be represented by 5 bit string and variable 2 can be represented by a 10 bit string, right. And then we have a mutation and crossover probability which is to be given and the tournament size k , right. So, the first step is to initialize a random population of binary string, remember this binary string. This is different from other metaheuristic techniques which we discussed.

The size will be $N \times p$ into n into D , right because for each decision variable we have n bits, right. So, that is why it is $N \times p$ into $n \times D$. Then we need to evaluate the fitness. So, to evaluate the fitness remember from binary string we cannot directly evaluate the fitness, from binary string we need to determine the decoded values, from the decoded values we need to determine the actual values, it is then the actual is can be taken and plugged into the fitness function and we can obtain the fitness function value, ok.

So, then we have this iteration loop for t is equal to 1 to capital T , right. Then we need to perform the tournament selection, in tournament selection we will take two members have a competition between them and the winner would be placed in the mating pool, right. So, for our discussion we took k is equal to 2, but k can be any number, right. So, once we have that mating pool we need to do $N \times p$ by 2 crossovers, right. So, because each crossover is going to give us 2 solutions, so we do $N \times p$ by 2 crossovers.

So, in that case we need to randomly choose two parents, not population member parents, right parents which we obtained at the end of tournament selection, right, which are there in the mating pool, right. And then we need to generate a random number for this case whether i equal to 1 will undergo a crossover or not. So, if this condition happens where p_c is crossover probability, again user defined it, then we need to randomly select a crossover site and generate two offspring using single point crossover.

So, quickly here if you say single point crossover it is just that if you have these two parents 1 0 0, 0 0 1 0 1 we cut over here and then swap it. So, this part would come here and this part would come here. Similarly, this part and this part would come over here, so that is how we

generate to offsprings over here. Similarly, we need to do for all the pairs, right. And if this condition fails, right we need to copy the selected parents as offspring, right.

So, at the end of this offspring we will definitely have N_p solutions, though we are doing N_p by 2 crossover each crossover gives us two solutions, so at the end of it we will have N_p solutions. And then we need to perform mutations. So, in the next step is to generate $n \cdot D$ random numbers, $n \cdot D$ because, if we have two variables and we have taken as n as 4 then our string length is actually 8, right.

So, we need to generate $n \cdot D$ random numbers which are between 0 and 1, right and then we need to check if a particular random number is less than mutation probability, then we need to perform the mutation. So, here if it is less than the mutation probability, only then we need to perform bitwise mutation, right. Once we have performed bitwise mutation then we have all the offspring.

Again, offsprings cannot be directly used to determine the fitness function value we need to convert them into their decimal equivalents or what we call decoded value, from that we need to find out their actual values using that expression $x_{\min} + (x_{\max} - x_{\min}) \cdot \frac{\text{decoded value}}{2^{n-1}}$ into decoder value and then we will get the actual value which can be used to determine the fitness of the offspring, right. Once we are done with that then we need to combine the population, not the parent, it is not parent it is population, right.

A parent will have multiple copies of population number. Parent is used only for crossover. Once crossover is done and we have the offspring we need to combine the offspring with the population, right, with the population that this generation had in its beginning, right. So, then we combine both of them and do a $\mu + \lambda$ strategy to select N_p members, right. So once we have the N_p members we need to keep doing this generations till capital T times, right, so because that is our termination criteria. So, that is binary coded genetic algorithm, right.

So, over here if we see this is the generation phase the in crossover. And in the mutation we are generating new solution, so that is the generation phase and from $\mu + \lambda$ we are

doing a survival of the fitness. So, out of two N_p solutions we are only selecting N_p solutions, right. So, the best get survived, right. So, if we see about the functional evaluations, right. So, initially N_p times we need to do the functional evaluation and over here also we need to do N_p times the functional evaluation, right.

So, over here remember we had this condition. So, just like if you go back and look at the fourth string for which we did mutation all the numbers which we generated randomly were greater than the mutation probability. So, we did not have to mutate any string. So, if we are not mutating then that means, that it is the same population member, right. So, if it is the same population member we did not evaluate the fitness again because we already would have its fitness, right.

So, in the worst case assuming that every time we do mutation we get a new solution N_p functional evaluation will be required, right. So, that is why the max is written over here. That the maximum number of functional evaluation there is N_p it can be less than that also. So, if the crossover did not happen that is r was not less than equal to p_c then you would have not done a crossover, right, you would have merely copied the parents.

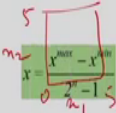
So, now, if you have merely copied the parents there is no need to evaluate their fitness because that is something which we already have, right. So, one can optimally utilize the fitness function evaluation, but in the worst case it will have N_p evaluations, right assuming that everything N_p members which we are generating are actually the new ones and not the parents are not being copied in the crossover and mutation actually results in mutating at least one of the bits, right.

So, if this is N_p times then that has to be performed T times over here, right. So, the total number of fitness function value will be N_p plus $N_p T$. Before we conclude binary GA, let us quickly look at the drawbacks of binary GA.

(Refer Slide Time: 60:29)

Drawbacks of binary GA

- Binary GA makes the search space discrete.
- Unable to achieve any arbitrary precision.
 - If n bits are used to represent a decision variable, then 2^n different values are possible between the lower and upper bound of the variable.
 - To increase the precision, n has to be increased.
 - Increase in n results in larger dimension and population size.



25

So, binary GA actually makes the search space discrete. So, if your search spaces between let us say the lower and upper bounds are 0 and 5 for x_1 and x_2 , right, it makes this the search space which is continuous it actually makes it discrete because it locates only at certain points because it works with the binary strings, right.

(Refer Slide Time: 60:50)

Drawbacks of binary GA

- Binary GA makes the search space discrete.
- Unable to achieve any arbitrary precision.
 - If n bits are used to represent a decision variable, then 2^n different values are possible between the lower and upper bound of the variable.
 - To increase the precision, n has to be increased.
 - Increase in n results in larger dimension and population size.
- Hamming cliffs: transition to neighboring solution (in real space) needs change in multiple bits (Example: 01111 (=15) to 10000 (=16)).

$$x = \frac{x^{\max} - x^{\min}}{2^n - 1}$$

25

It cannot achieve any arbitrary precision, so the precision is governed by this one this expression which we have seen previously, right, so if you want to increase the precision we will have to increase the size of the bit length, right. And if we increase the bit length, if we increase the bit length then the dimension of the problem increases. Like dimension in the sense your optimization problem might have only two variables, but because you are choosing to represent it by a larger string the size of the population, the number of columns in the population would go up, right.

So, that increases the computational load, right. So, that is a problem with binary GA. And this is a classical example what is called as Hamming cliffs. So, let us say we are at a solution 15, right or let us say we are at this solution 01111, let us say that corresponds to this 15 that

decoded value of this is actually 15 you can calculate that, right. So, from 15 if I want to move to 16, right.

So, 15 and 16 if you think about in real space they are close, right whereas, for this solution to become this solution, right this is the solution for which the decoded value is 16. For this solution to turn out to be this solution changes need to happen on 4 strings, right only then this solution which is at 15 can actually come to this solution representing 16. So, that is very difficult to happen.

So, if this has to happen in mutation then we are expecting all the 5 random numbers chosen for all these 5 elements to be actually less than the mutation probability only then this will get converted into this or you are relying on a crossover operator where in you get two such solutions for which if you do a crossover you will end up with this solution, right.

So, points which are even closer by it might be difficult for binary GA to move from point a to point b despite the fact that they are actually closer in the real space, but they are not closer in the binary space or at least because of the operators that we employ, right. So, that is a drawback of binary coded GA.

So, in the next session we will look into real coded GA, right.

Thank you.