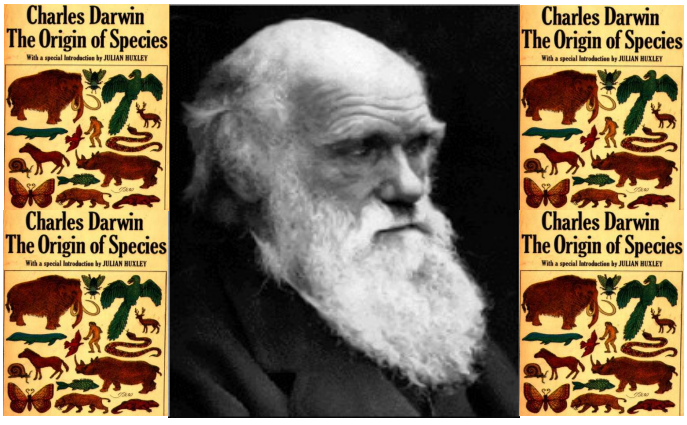
Genetic Algorithm (GA)

M.M. Pedram
pedram@khu.ac.ir
Kharazmi University
(Spring 2020)

1

Charles Darwin 1809 - 1882



"A man who dares to waste an hour of life has not discovered the value of life"

2

Selection Schemes

complete by the following link:

http://www.geatbx.com/docu/algindex-02.html

Or by

.\GA (Link)\CMPE538_L3_I.ppt

And

.\GA (Link)\ 0101.Genetic-Algorithm.ppt

3

Genetic Algorithms

- Life on earth has evolved to be as it is through the processes of natural selection, recombination and mutation.
- **Genetic algorithms** are a part of **evolutionary computing**, which is a rapidly growing area of artificial intelligence.
- As you can guess, genetic algorithms are inspired by Darwin's theory of evolution. Simply said, problems are solved by an evolutionary process resulting in a best (fittest) solution (survivor) - in other words, the solution is evolved.

4

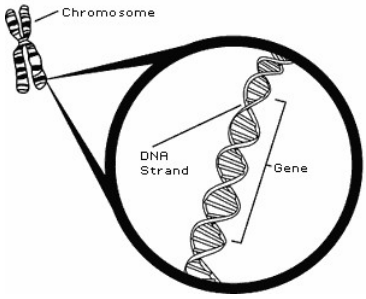
History

- Evolutionary computing was introduced in the 1960s by I. Rechenberg in his work "*Evolution strategies*" (*Evolutions strategy* in original). His idea was then developed by other researchers.
- Genetic Algorithms (GAs) were invented by John Holland and developed by him and his students and colleagues. This lead to Holland's book "*Adaption in Natural and Artificial Systems*" published in 1975.
- In 1992 John Koza has used genetic algorithm to evolve programs to perform certain tasks. He called his method "genetic programming" (GP). LISP programs were used, because programs in this language can expressed in the form of a "parse tree", which is the object the GA works on.

5

Biological Background : Chromosomes

- Genetic information is stored in the ***chromosomes***,
- Each chromosome is build of ***DNA***,
- Chromosomes in humans form pairs,
- There are 23 pairs,
- The chromosome is divided in parts: ***genes***
- Genes code for properties
- The possibilities of the genes for one property is called: ***allele***
- Every gene has an unique position on the chromosome: ***locus***



6

Genetic algorithms (GAs)

- Every organism has a set of rules, describing how that organism is built up from the tiny building blocks of life.
- These rules are encoded in the *genes* of an organism, which in turn are connected together into long strings called *chromosomes*.
- Each gene represents a specific characteristic of the organism, like eye color or hair color, and has several different settings. For example, the settings for a hair color gene may be blonde or black. These genes and their settings are usually referred to as an organism's genotype. The physical expression of the genotype - the organism itself - is called the phenotype.
- When two organisms mate they share their genes. The resultant offspring may end up having half the genes from one parent and half from the other. This process is called *recombination*. Very occasionally a gene may be *mutated*. Normally this mutated gene will not affect the development of the phenotype but very occasionally it will be expressed in the organism as a completely new characteristic.

7

What Are Genetic Algorithms?

- A way to employ evolution in the computer,
- Search and optimization technique based on variation and selection.

8

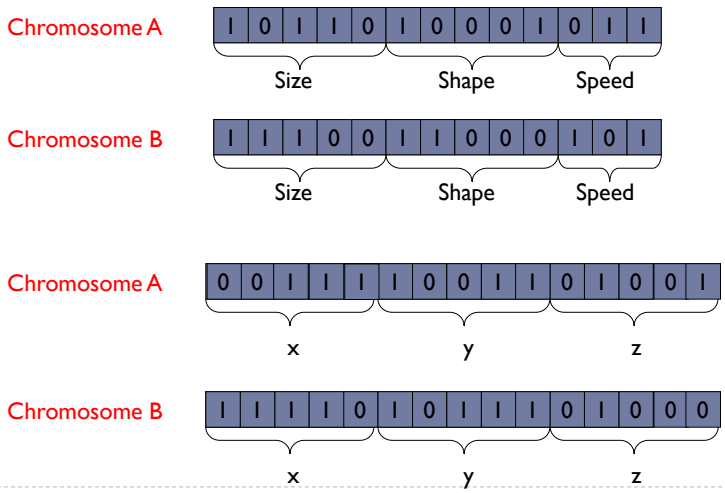
Components of GAs : Encoding

- Encoding of chromosomes is the first question to ask when starting to solve a problem with GA. Encoding depends on the problem heavily.
- This could be as a string of real numbers or, as is more typically the case, a binary bit string.
- *Binary encoding* is the most common one, mainly because the first research of GA used this type of encoding and because of its relative simplicity.
- In binary encoding, every chromosome is a string of bits 0 or 1.

Chromosome A	10110010110010101100101
Chromosome B	11111110000011000001111

Components of GAs : Encoding

Solutions are coded as "chromosomes"



Components of GAs : Encoding

- Given the digits 0 through 9 and the operators +, -, * and /, find a sequence that will represent a given target number. The operators will be applied sequentially from left to right as you read.
- So, given the target number 23, the sequence 6+5*4/2+1 would be one possible solution.
- If 75.5 is the chosen number then 5/2+9*7-5 would be a possible solution.
- First we need to encode a possible solution as a string of bits, i.e. a chromosome.
- So how do we do this? Well, first we need to represent all the different characters available to the solution... that is 0 through 9 and +, -, * and /. This will represent a gene. Each chromosome will be made up of several genes.

11

Components of GAs : Encoding

In this example, 4 bits are required to represent the range of characters used:

0: 0000
1: 0001
2: 0010
3: 0011
4: 0100
5: 0101
6: 0110
7: 0111
8: 1000
9: 1001
+: 1010
-: 1011
*: 1100
/: 1101

The possible genes **1110** & **1111** will remain unused and will be ignored by the algorithm if encountered.

So now you can see that the solution mentioned above for 23, ' 6+5*4/2+1' would be represented by nine genes like so:

0110 1010 0101 1100 0100 1101 0010 1010 0001
6 + 5 * 4 / 2 + 1

These genes are all strung together to form the chromosome:
011010100101111000100110100101010000 1

12

Components of GAs : Decoding

Because the algorithm deals with random arrangements of bits it is often going to come across a string of bits like this:

0010001010101110101101110010

Decoding these bits:

0010	0010	1010	1110	1011	0111	0010
2	2	+	n/a	-	7	2

Which is meaningless in the context of this problem! Therefore, when decoding, the algorithm will just ignore any genes which don't conform to the expected pattern of: number → operator → number → operator … and so on. With this in mind the above 'nonsense' chromosome is read (and tested) as:

0010	0010	1010	1110	1011	0111	0010
2	2	+	n/a	-	7	2

2 + 7

13

Components of GAs: Decoding

0:	0000
1:	0001
2:	0010
3:	0011
4:	0100
5:	0101
6:	0110
7:	0111
8:	1000
9:	1001
+:	1010
-:	1011
*:	1100
/:	1101

0010 1010 0011 1100 0101 1010 0110
2 + 3 * 5 + 6

=30

1000 1100 0011 1101 0011 1010 0010
8 * 3 / 3 + 2

=10

0111 1010 0100 1011 0001 1101 0010
? ? ? ? ? ? ?

=?

14

Components of GAs: Permutation Encoding

- Permutation encoding can be used in ordering problems, such as travelling salesman problem or task ordering problem.
- In **permutation encoding**, every chromosome is a string of numbers that represent a position in a **sequence**.

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

- Permutation encoding is useful for ordering problems. For some types of crossover and mutation corrections must be made to leave the chromosome consistent (i.e. have real sequence in it) for some problems.

Example of Problem: Travelling salesman problem (TSP)
The problem: There are cities and given distances between them. Travelling salesman has to visit all of them, but he does not want to travel more than necessary. Find a sequence of cities with a minimal travelled distance.
Encoding: Chromosome describes the order of cities, in which the salesman will visit them.

Components of GAs: Value Encoding

- Direct value encoding can be used in problems where some more complicated values such as real numbers are used. Use of binary encoding for this type of problems would be difficult.
- In the **value encoding**, every chromosome is a sequence of some values. Values can be anything connected to the problem, such as (real) numbers, chars or any objects.
- *Example of chromosomes with value encoding*

Chromosome A	1.2324 5.3243 0.4556 2.3293 2.4545
Chromosome B	ABDJEIFJDHDIERJFDLDFLFEGT
Chromosome C	(back), (back), (right), (forward), (left)

- Value encoding is a good choice for some special problems. However, for this encoding it is often necessary to develop some new crossover and mutation specific for the problem.

Example of Problem: Finding weights for a neural network.
The problem: A neural network is given with defined architecture. Find weights between neurons in the neural network to get the desired output from the network.
Encoding: Real values in chromosomes represent weights in the neural network.

Selection Schemes

- In selection, the offspring producing individuals are chosen.
  - The first step is fitness assignment. Each individual in the selection pool receives a reproduction probability depending on the own objective value and the objective value of all other individuals in the selection pool.
  - The second step is selection in which fitness is used for the actual selection step afterwards.

- Different Selection Schemes:
  - Rank-based fitness assignment
  - Multi-objective Ranking
  - Roulette wheel selection
  - Stochastic universal sampling
  - Local selection
  - Truncation selection
  - Tournament selection

17

Selection Schemes

Rank-based fitness assignment:

- The population is sorted according to the objective values. The fitness assigned to each individual depends only on its position in the individuals rank and not on the actual objective value.
- Rank-based fitness assignment overcomes the scaling problems of the proportional fitness assignment. (Stagnation in the case where the selective pressure is too small or premature convergence where selection has caused the search to narrow down too quickly.)
- The reproductive range is limited, so that no individuals generate an excessive number of offspring.
- Ranking introduces a uniform scaling across the population and provides a simple and effective way of controlling selective pressure.

18

Selection Schemes

complete by the following link:

http://www.geatbx.com/docu/algindex-02.html

Or by

.\GA (Link)\CMPE538_L3_I.ppt

And

.\GA (Link)\ 0101.Genetic-Algorithm.ppt

19

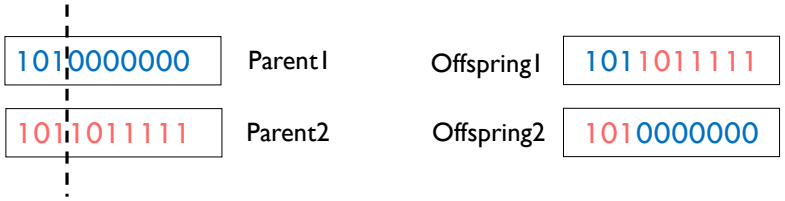
Crossover and Mutation

- **Crossover:** Operate on selected parent chromosomes to create new offspring.
- **Mutation:** Randomly changes the offspring resulted from crossover to prevent falling of all solutions into a local optimum.

20

Crossover

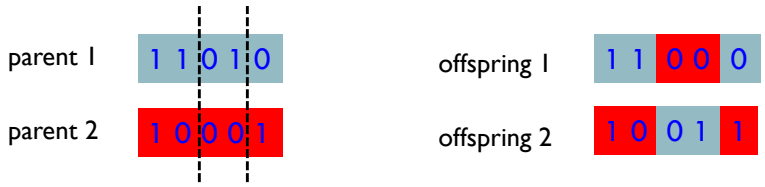
Crossover: single point - random



21

Crossover

Crossover: two-point - random



22

Crossover

- Assign 'heads' to one parent, 'tails' to the other
- Flip a coin for each gene of the first child
- Make an inverse copy of the gene for the second child
- Inheritance is independent of position

Crossover: Uniform

- Parent 1: 00000000000000000
- Parent 2: 11111111111111111
- Offspring 1: 01001011000101100 1
- Offspring 2: 10110100111010011 0

Crossover

Crossover: Single arithmetic crossover

- Parents: $\langle x_1,\ldots,x_n \rangle$ and $\langle y_1,\ldots,y_n \rangle$
- Pick a single gene (k) at random,
- child$_1$ is: $\langle x_1, \ldots, x_{k-1}, \alpha \cdot y_k + (1-\alpha)\cdot x_k, \ldots, x_n \rangle$
- reverse for other child. e.g. with $\alpha = 0.8$

Parent 1:	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Parent 2:	0.3	0.1	0.4	0.5	0.3	0.2	0.6	0.6	0.3

$0.8\times 0.6+(1-0.8)0.8=0.48+0.16=0.64\approx 0.6$

Offspring 1:	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.6	0.9

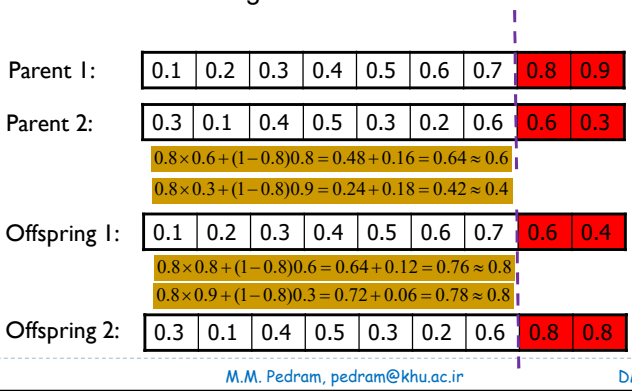
$0.8\times 0.8+(1-0.8)0.6=0.64+0.12=0.76\approx 0.8$

Offspring 2:	0.3	0.1	0.4	0.5	0.3	0.2	0.6	0.8	0.3

Crossover

Crossover: Simple arithmetic crossover

- Parents: $\langle x_1,\ldots,x_n \rangle$ and $\langle y_1,\ldots,y_n \rangle$
- Pick random gene *(k)* after this point mix values
  child$_1$ is: $\langle x_1, \ldots, x_{k-1}, \alpha \cdot y_k + (1-\alpha)\cdot x_k, \ldots, \alpha \cdot y_n + (1-\alpha)\cdot x_n \rangle$
- reverse for other child. e.g. with α = 0.8



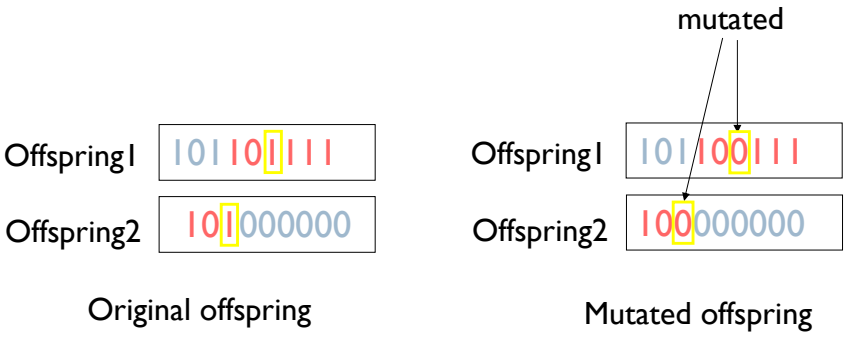
25

Crossover

- Note : no crossover, offspring is exact copy of parents

26

Mutation



With some small probability (the *mutation rate*) flip each bit in the offspring (typical values between 0.1 and 0.001)

Mutation

Order changing mutation - two numbers are selected and exchanged
(1 **2** 3 4 5 6 **8** 9 7) => (1 **8** 3 4 5 6 **2** 9 7)

Adding Mutation: a small number (for real value encoding) is added to (subtracted from) selected values

(1.29 5.68 **2.86** **4.11** 5.55) => (1.29 5.68 **2.73** **4.22** 5.55)

Components of Genetic algorithms : Create initial population (usually random)

1	10110010110010101110010101010011011001011001010110
2	00100010110010101110010101011111011001011000010110
3	00100010110011111110010111011100001001011000010110
4	10000010110010101110111111010011011001000001010110
5	11111111110010101110010101111111011001011001011111
6	00100010110011111111111111100011001001011000010110
7	10000010100110011110010101110111111010011011001001
8	11111111111000000000111110111111010011011111101000
9	00000000000000000000000000000001111111101010101011
10	00000000011001110000101010101010000000000000011111111
11	00011011011011110000101010101010000000000000011110101
12	11111000011001110000101010101010001111100000000000100
13	10101000011001110000101000001010001111100001110010 1
14	10101000011111111110010100001111111111110011111000 01
15	10101000010101010101011100101010101010101010101011 1
16	10101001110101010111011100101010101110010000001010 1
17	00010000111101010111000000101010000110010000001000 0

29

Simple example of Genetic optimization

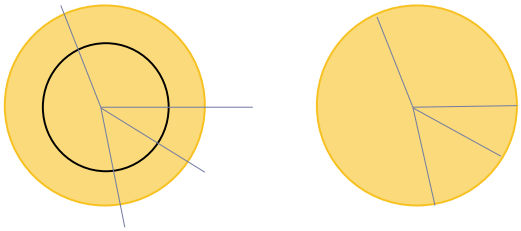
The chosen problem is to minimize the value of a string of 1's and 0's as a binary number. While this is a trivial problem, it will serve to illustrate the operation of the algorithm without the obscuring effects of a complicated objective function.

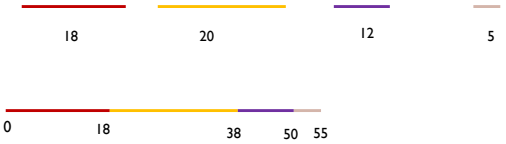
The genetic algorithm performs minimization in the following steps:

1. A random population of chromosomes are generated.
2. The fitness function is applied to each chromosome.
3. The selection operation is applied favoring the fittest chromosomes.
4. The crossover operation is applied on the selected pairs to generate a new population.
5. Mutation is applied to some offspring
6. Repeat steps 2-5

30

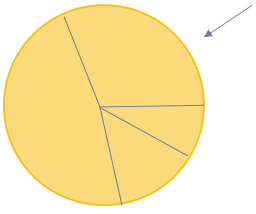
ساز و کار چرخ رولت

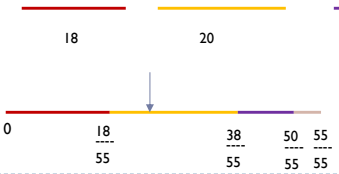




31

ساز و کار چرخ رولت





32

Components of Genetic algorithms : Selection of parents (reproduction)

In this step the current population is used to produce a second generation of population that, on the average, have a higher objective function.

Selection replicates the most successful solutions found in a population at a rate proportional to their relative quality.

J chromosome number	Binary chromosome	Decimal value	O_j = 1/(decimal value) Objective function	$F_j = O_j$ / (Total O_j) Fraction of Total
1	011010	26	0.0385	0.142
2	001011	11	0.0909	0.334
3	110110	54	0.0185	0.068
4	010011	19	0.0526	0.194
5	100111	39	0.0256	0.094
6	010110	22	0.0455	0.168
Total			0.2717	1

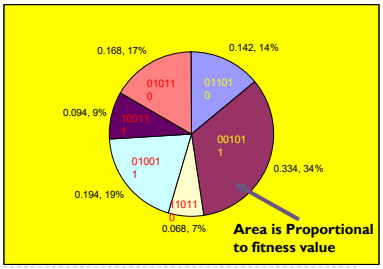
33

Components of Genetic algorithms : Selection of parents (reproduction)

J chromosome number	Binary chromosome	Decimal value	O_j = 1/(decimal value) Objective function	$F_j = O_j$ / (Total O_j) Fraction of Total	Round (F_j*6)= Copy number
1	011010	26	0.0385	0.142	1
2	001011	11	0.0909	0.334	2
3	110110	54	0.0185	0.068	0
4	010011	19	0.0526	0.194	1
5	100111	39	0.0256	0.094	1
6	010110	22	0.0455	0.168	1
Total			0.2717	1	

In the reproduction phase, each chromosome is copied to the second generation a number of times that is proportional to its fractional objective function F_j.

Selected population
011010
001011
001011
010011
100111
010110



34

Components of Genetic algorithms : Crossover (Recombination)

Recombination decomposes two distinct solutions (chromosomes) and then randomly mixes their parts to form novel solutions (chromosomes) .
This process is intended to simulate the exchange of genetic material that occurs during biological reproduction. Here, pairs in the breeding population are mated randomly. For simplicity we will mate adjacent pairs. For each pair, a random integer from 1-6 (1- length of chromosome) is chosen. This indicates how many bits on the right end of each chromosome should be exchanged between the two mated chromosomes.

Before crossover:

s_{P1} = 011010 s_{P3} = 001011 s_{P5} = 100111

s_{P2} = 001011 s_{P4} = 010011 s_{P6} = 010110

After crossover:

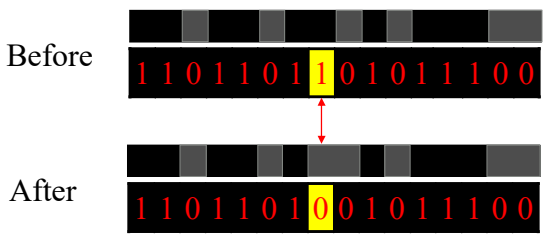
s_{C1} = 011011 s_{C3} = 000011 s_{C5} = 100110

s_{C2} = 001010 s_{C4} = 011011 s_{C6} = 010111

35

Components of Genetic algorithms : Mutation

A surprisingly small role is played by mutation. Biologically, it randomly perturbs the population's characteristics, thereby preventing evolutionary dead ends.
Most mutations are damaging rather than beneficial; therefore, rates must be low to avoid the destruction of species. In the simulation we will assume a mutation rate of on bit per thousand, where a mutation acts to reverse the value of single bit. Since we have only 36 bits in our population, the probability of a mutation is only 0.036; therefore, no bits are reversed in the simulation. By mutation, algorithm explores parts of the space that crossover might miss, and helps prevent premature convergence.



36

Components of Genetic algorithms : Crossover (Recombination)

After One Generation

J chromosome number	Binary chromosome	Decimal value	O_j = 1/(decimal value) Objective function	$F_j = O_j$ / (Total O_j) Fraction of Total
1	**011011**	27	0.037	0.064
2	**001010**	10	0.10	0.174
3	**000011**	3	0.333	0.578
4	**011011**	27	0.037	0.064
5	**100110**	38	0.0263	0.046
6	**010111**	23	0.043	0.075
Total			0.576	1

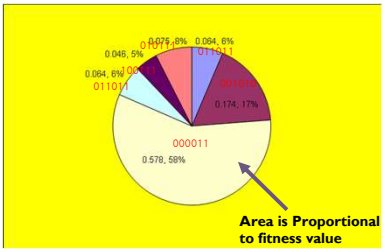
37

Components of Genetic algorithms : Selection of parents (reproduction)

J chromosome number	Binary chromosome	Decimal value	O_j = 1/(decimal value) Objective function	$F_j = O_j$ / (Total O_j) Fraction of Total	Round (F_j*6)= Copy number
1	**011011**	27	0.037	0.064	**0**
2	**001010**	10	0.10	0.174	**1**
3	**000011**	3	0.333	0.578	**4**
4	**011011**	27	0.037	0.064	**0**
5	**100110**	38	0.0263	0.046	**0**
6	**010111**	23	0.043	0.075	**1**
Total			0.576	1	

In the reproduction phase, each chromosome is copied to the second generation a number of times that is proportional to its fractional objective function F_j.

Selected population
001010
000011
000011
000011
000011
010111



38

Components of Genetic algorithms : Crossover (Recombination)

Before crossover:

s_{P1} = 001010 s_{P3} = 000011 s_{P5} = 000011

s_{P2} = 000011 s_{P4} = 000011 s_{P6} = 010111

After crossover:

s_{C1} = 001011 s_{C3} = 000011 s_{C5} = 000011

s_{C2} = 000010 s_{C4} = 000011 s_{C6} = 010111

39

Components of Genetic algorithms : Crossover (Recombination)

After Two Generations

J chromosome number	Binary chromosome	Decimal value	O_j = 1/(decimal value) Objective function	$F_j = O_j$ / (Total O_j) Fraction of Total
1	**001011**	11	0.091	0.0557
2	**000010**	2	0.5	0.306
3	**000011**	3	0.333	0.204
4	**000011**	3	0.333	0.204
5	**000011**	3	0.333	0.204
6	**010111**	23	0.043	0.026
Total			1.633	1

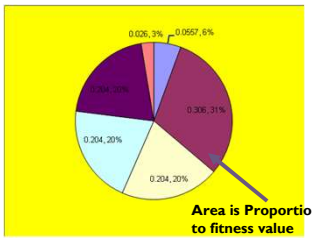
40

Components of Genetic algorithms : Selection of parents (reproduction)

J chromosome number	Binary chromosome	Decimal value	O_j = 1/(decimal value) Objective function	$F_j = O_j$ / (Total O_j) Fraction of Total	Round (F_j*6)= Copy number
1	**001011**	11	0.091	0.0557	**0**
2	**000010**	2	0.5	0.306	**2**
3	**000011**	3	0.333	0.204	**1**
4	**000011**	3	0.333	0.204	**1**
5	**000011**	3	0.333	0.204	**1**
6	**010111**	23	0.043	0.026	**0**
Total			1.633	1	

In the reproduction phase, each chromosome is copied to the second generation a number of times that is proportional to its fractional objective function F_j.

Selected population
000010
000010
000011
000011
000011
000011



41

Components of Genetic algorithms : Crossover (Recombination)

Before crossover:

s_{P1} = 000010 s_{P3} = 000011 s_{P5} = 000011

s_{P2} = 000010 s_{P4} = 000011 s_{P6} = 000011

After crossover:

s_{C1} = 000010 s_{C3} = 000011 s_{C5} = 000011

s_{C2} = 000010 s_{C4} = 000011 s_{C6} = 000011

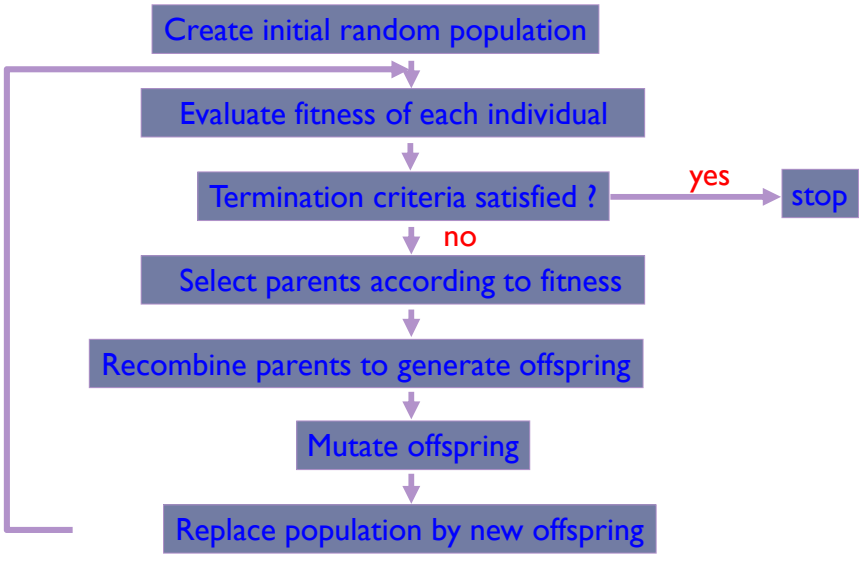
42

Components of Genetic algorithms : Crossover (Recombination)

After Three Generations

J chromosome number	Binary chromosome	Decimal value	O_j = 1/(decimal value) Objective function	$F_j = O_j$ / (Total O_j) Fraction of Total
1	**000010**	2	0.5	0.214
2	**000010**	2	0.5	0.214
3	**000011**	3	0.333	0.143
4	**000011**	3	0.333	0.143
5	**000011**	3	0.333	0.143
6	**000011**	3	0.333	0.143
Total		16	2.332	1

The Evolutionary Cycle



Example: the MAXONE problem

Suppose we want to maximize the number of ones in a string of *l* binary digits

Is it a trivial problem?

It may seem so because we know the answer in advance

- An individual is encoded (naturally) as a string of l binary digits
- The fitness f of a candidate solution to the MAXONE problem is the number of ones in its genetic code
- We start with a population of n random strings. Suppose that l = 10 and n = 6

Genetic algorithms : Example (**initialization**) the MAXONE problem

J chromosome number	Binary chromosome	O_j = number of ones Objective function	$F_j = O_j$ / (Total O_j) Fraction of Total
1	1111010101	7	0.206
2	0111000101	5	0.147
3	1110110101	7	0.206
4	0100010011	4	0.118
5	1110111101	8	0.235
6	0100110000	3	0.088
Total		34	1

Genetic algorithms : Selection of parents (reproduction)

J chromosome number	Binary chromosome	O_j = 1/(decimal value) Objective function	$F_j = O_j$ / (Total O_j) Fraction of Total	Round (F_j*6)= Copy number
1	1111010101	7	0.206	**1**
2	0111000101	5	0.147	**1**
3	1110110101	7	0.206	**1**
4	0100010011	4	0.118	**1**
5	1110111101	8	0.235	**1**
6	0100110000	3	0.088	**0**
Total		34	1	

Selected population
1111010101
0111000101
1110110101
0100010011
1110111101
1110111101

47

Genetic algorithms : Example (crossover I) the MAXONE problem

Next we mate strings for crossover. For each couple we decide according to crossover probability (for instance 0.6) whether to actually perform crossover or not.

Suppose that we decide to actually perform crossover only for couples (s_1`, $s_2$`) and (s_5`, $s_6$`). For each couple, we randomly extract a crossover point, for instance 2 for the first and 5 for the second

48

Genetic algorithms : Example (crossover 2)
the MAXONE problem

Before crossover:

$s_{P1} = 1111010101$ $s_{p4} = 0100010011$

$s_{P3} = 1110110101$ $s_{P5} = 1110111101$

After crossover:

$s_{C1} = 1110110101$ $s_{C4} = 0100011101$

$s_{C3} = 1111010101$ $s_{C5} = 1110110011$

Genetic algorithms : Example (Mutation)
the MAXONE problem

The final step is to apply random mutation: for each bit that we are to copy to the new population we allow a small probability of error (for instance 0.1)

Before applying mutation:

Modified population
1110110101
0111000101
1111010101
0100011101
1110110011
1110111101

After applying mutation:

Modified population
1110100101
0111000101
1111110100
0100011101
1110110001
1110101111

Genetic algorithms : Example (End)
the MAXONE problem

J chromosome number	Binary chromosome	O_j = number of ones Objective function	$F_j = O_j$ / (Total O_j) Fraction of Total
1	1110100101	6	0.162
2	0111000101	5	0.135
3	1111110100	7	0.189
4	0100011101	5	0.135
5	1110110001	6	0.162
6	1110101111	8	0.216
Total		37	1

In one generation, the total population fitness changed from 34 to 37, thus improved by ~9%

At this point, we go through the same process all over again, until a stopping criterion is met

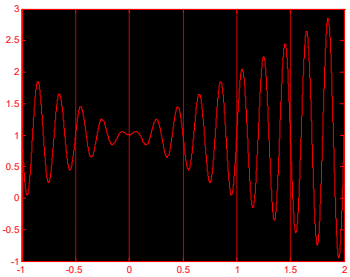
51

Genetic algorithms : Optimization of a simple function

The function is : $f(x) = x.\sin(10\pi.x) + 1.0$

The problem is to find x from the rang [−1 2] which

$f(x_0) \geq f(x), \quad for\ all\ x \in [-1\ 2]$ x_0 such t



It is relatively easy to analyze the function f. The zeros of the first derivative f' should be

$$f'(x) = \sin(10\pi.x) + 10\pi x.\cos(10\pi.x) = 0$$

$$\tan(10\pi.x) = -10\pi.x$$

$$x_i = \frac{2i-1}{20} \quad for\ i = 1,2,\cdots$$

$$x_0 = 0$$

$$x_i = \frac{2i+1}{20} \quad for\ i = -1,-2,\cdots$$

The function f reaches its local max for x_i if i is an odd integer and its local min for x_i if i is an even integer. The function reaches its max for:

$$x_{19} = \frac{37}{20} = 1.85$$

$$f(1.85) = 1.85.\sin(18\pi + \frac{\pi}{2}) + 1.0 = 2.85$$

52

Genetic algorithms : Optimization of a simple function

Assume that we wish to construct a genetic algorithm: to maximize the function f.
Let discuss the major components of such a genetic algorithm in turn.

Genetic algorithms : Presentation (Encoding)

We use a binary vector as a chromosome to present real values of the variable x.
The length of the vector depends on the required precision, which, in this example, is six places after the decimal point.
The domain of the variable x has length 3; the precision requirement implies that the range [−1 2] should be divided into at least 3*1,000,000=3,000,000 equal size ranges. This means that 22 bits are required as a binary vector (chromosome):

$$2097152 = 2^{21} < 3000000 < 2^{22} = 4194304$$

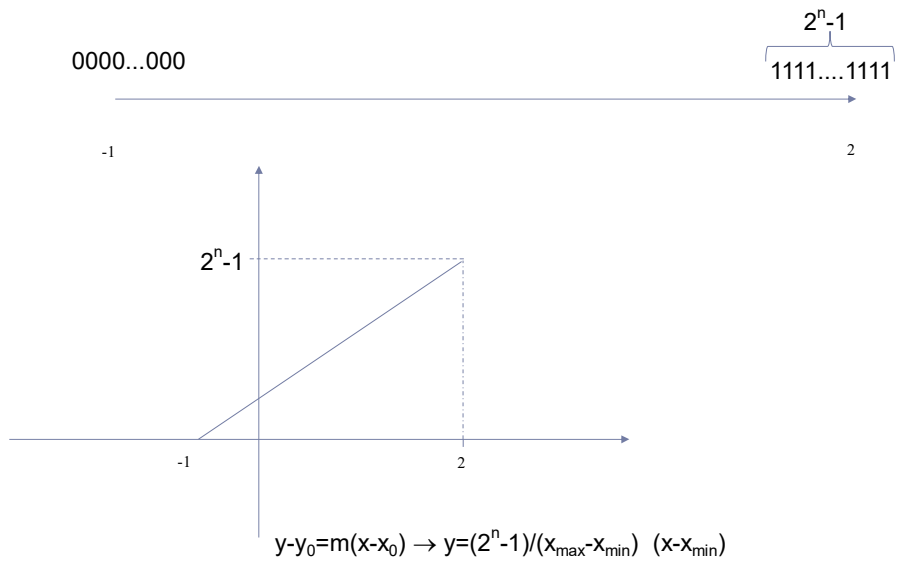
The mapping from a binary vector (chromosome) $(b_{21}b_{20}....b_0)$ into a real number X from the range [−1 2] is straightforward and is completed in to steps:
•Convert the binary chromosome ($b_{21}b_{20}\cdots.b_0$) from the base 2 to base 10:

$$(b_{21}b_{20}\cdots b_0)_2 = \left(\sum_{i=0}^{21} b_i.2^i\right)_{10} = x'$$

- Find a corresponding real number x :

$$x_i = a_i + decimal(10001....10000_2).\frac{b_i - a_i}{2^{m_i} - 1} \qquad x = -1.0 + x'.\frac{3}{2^{22} - 1}$$

Where −1.0 is the left boundary of the domain and 3 is the length of the domain:



$y-y_0=m(x-x_0) \rightarrow y=(2^n-1)/(x_{max}-x_{min})\ (x-x_{min})$

$y-y_0=m(x-x_0) \rightarrow y=(2^m-1)/(b-a)\ (x-a)$

55

Genetic algorithms : Presentation (Encoding)

Chromosome: 10001011101101010001111

y = $x' = (10001011101101010001111)_2 = 2288967$

$$x = -1.0 + x'.\frac{3}{2^{22}-1}$$

$$x = -1.0 + 2288967.\frac{3}{4194303} = 0.637197$$

Of course, the chromosomes (00000000000000000000) and (11111111111111111111)
Represent boundaries of the domain, −1.0 and 2.0

56

Genetic algorithms : Initial population

The initialization process is very simple:
we create a population of chromosomes,
where each chromosome is is a binary vector
of 22 bits. All 22 bits for each chromosome
are initialized
Randomly.

57

Genetic algorithms : Evaluation function

Evaluation function eval(⋆) for binary chromosome v is equivalent to the function f:

$$eval(v) = f(x)$$

As noted earlier, the evaluation plays the role of the environment, rating potential solution
In terms of their fitness. For example, three chromosomes:

$$v_1 = (1000101110110101000111) \; corresponed \; to \; x_1 = 0.637197$$
$$v_2 = (0000001110000000010000) \; corresponed \; to \; x_2 = -0.958973$$
$$v_3 = (1110000000111111000101) \; corresponed \; to \; x_3 = 1.627888$$

$$eval(v_1) = f(x_1) = 1.586345$$
$$eval(v_2) = f(x_2) = 0.078878$$
$$eval(v_3) = f(x_3) = 2.250650$$

Clearly, the chromosome v_3 is the best of the three chromosomes, since its evaluation retur
the highest value.

58

Genetic algorithms : Mutation

Assume that the fifth gene from the v_3 chromosome was selected for mutation:

$v_3 = (11100\underline{0}0000011111000101)$ *corresponed to* $x_3 = 1.627888$

$v'_3 = (11101\underline{1}0000011111000101)$ *corresponed to* $x'_3 = 1.721638$

$eval(v_3) = f(x_3) = 2.250650$

$eval(v'_3) = f(x'_3) = -0.082257$

This means that this particular mutation resulted in a significant decrease of the value o the chromosome v_3. If the 10th gene was selected for mutation in chromosome v_3, the

$v_3 = (1110000000\underline{1}11111000101)$ *corresponed to* $x_3 = 1.627888$

$v''_3 = (1110000001\underline{1}11111000101)$ *corresponed to* $x''_3 = 1.630818$

$eval(v_3) = f(x_3) = 2.250650$

$eval(v''_3) = f(x''_3) = 2.343555$

59

Genetic algorithms : Crossover

Before crossover:

00000011100000000010000

11100000001111111000101

$v_2 = (00000011100000000010000)$ *corresponed to* $x_2 = -0.958973$ $eval(v_2) = f(x_2) = 0.078878$

$v_3 = (11100000001111111000101)$ *corresponed to* $x_3 = 1.627888$ $eval(v_3) = f(x_3) = 2.250650$

After crossover:

00000000001111111000101

11100111000000000010000

$v'_2 = (00000000001111111000101)$ *corresponed to* $x'_2 = -0.998113$ $eval(v'_2) = f(x'_2) = 0.940865$

$v'_3 = (11100011100000000010000)$ *corresponed to* $x'_3 = 1.666028$ $eval(v'_3) = f(x'_3) = 2.459245$

Note that the second offspring has a better evaluation than both of its parents.

60

Genetic algorithms : Parameters

For this particular problem we have used the following parameters: Population size: pop_size = 50, probability of crossover P_c =0.25, probability of mutation P_m = 0.01.

In table, we provide the generation number for which we noted an improvement in the evaluation function, together with the value of function. The best chromosome after 150 generations was:

$$v_{max} = (11110011010001000000101)$$
$$corresponed\ to\ x_{max} = 1.850773$$

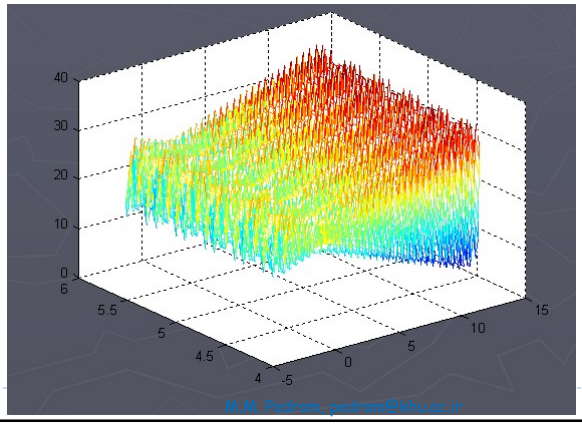
Generation number	Evaluation function
1	1.441942
6	2.250003
8	2.250283
9	2.250284
10	2.250363
12	2.328077
39	2.344251
40	2.345087
51	2.738930
99	2.849246
137	2.850217
145	2.850227

61

Genetic algorithms : Optimization of a two variables function

The function is defined as: $f(x_1, x_2) = 21.5 + x_1.\sin(4\pi.x_1) + +x_2.\sin(20\pi.x_2)$

Where $-3 \le x_1 \le 12.1 \quad and \quad 4.1 \le x_2 \le 5.8$



62

Genetic algorithms : Presentation (Encoding)

We use a binary vector as a chromosome to present real values of the variable x.
The length of the vector depends on the required precision, which, in this example, is four places after the decimal point.

The domain of the variable x_1 has length 15.1; the precision requirement implies that the range [−3 12.1] should be divided into at least 15.1*10000=151000 equal size ranges. This means that 18 bits are required as a binary vector (chromosome):

$$131072 = 2^{17} < 151000 < 2^{18} = 262144$$

The domain of the variable x_2 has length 1.7; the precision requirement implies that the range [4.1 5.8] should be divided into at least 1.7*10000=17000 equal size ranges. This means that 15 bits are required as a binary vector (chromosome):

$$16384 = 2^{14} < 17000 < 2^{15} = 32768$$

63

Genetic algorithms : Presentation (Encoding)

The total length of a chromosome (solution vector) is then m = 18 + 15=33 bits; the first 18 bits code x_1 and remaining 15 bits (19−33) code x_2.
Let us consider an example chromosome:

010001001011010000111110010100010

$$x_i = a_i + decimal(10001....10000_2).\frac{b_i - a_i}{2^{m_i} - 1}$$

The first 18 bits: 010001001011010000 represents

$$x_1 = -3.0 + decimal(010001001011010000_2).\frac{12.1-(-3.0)}{2^{18}-1} = -3.0 + 70352.\frac{15.1}{262143} = 1.052426$$

The first 18 bits: 111110010100010 represents:

$$x_2 = 4.1 + decimal(111100101000102).\frac{5.8-4.1}{2^{15}-1} = 4.1 + 31906.\frac{1.7}{32767} = 5.755330$$

So the chromosome: 010001001011010000111110010100010 corresponds to (x_1, x_2)= (1.052426,5.755330). The fitness value for this chromosome is:

f(1.052426,5.755330)=20.252640.

64

Genetic algorithms : Initial population

To optimize the function f using a genetic algorithm, we create a population of pop_size = 20 chromosomes. All 33 bits in all chromosomes are initialized andomly.

$v_1 = 10011010000000111111101001101111$
$v_2 = 11100010010011011100101010011010$
$v_3 = 00001000001100100000101011011101$
$v_4 = 10001100010110100111100000111010$
$v_5 = 00011101100101001101011111000101$
$v_6 = 00010100001001010100101011111011$
$v_7 = 00100010000011010111011011111011$
$v_8 = 10000110000111010001011010100111$
$v_9 = 01000000010110001011000000111100$
$v_{10} = 00000111100011000001101000011101 1$
$v_{11} = 01100111111011010110000110111000$
$v_{12} = 11010001011110110100010101000000$
$v_{13} = 11101111101000100011000000100011 0$
$v_{14} = 01001001100000101010011100101001$
$v_{15} = 11101110110111000010001111011110$
$v_{16} = 11001111000001111110000110100101 1$
$v_{17} = 01101011111100111010001101111101$
$v_{18} = 01110100000000111010011110101101$
$v_{19} = 00010101001111111111000011000110 0$
$v_{20} = 10111001011001111001100010111111 0$

65

During the evaluation phase we decode each chromosome and calculate the fitness Function values from (x_1, x_2) values just decoded. We get:

$v_1 = 10011010000000111111101001101111 \mid x_1 = 6.0844 \quad x_2 = 5.6522 \mid eval(x_1, x_2) = f(x_1, x_2) = 26.0196$
$v_2 = 11100010010011011100101010011010 \mid x_1 = 10.3484 \quad x_2 = 4.3802 \mid eval(x_1, x_2) = f(x_1, x_2) = 17.5800$
$v_3 = 00001000001100100000101011011101 \mid x_1 = -2.5166 \quad x_2 = 4.3903 \mid eval(x_1, x_2) = f(x_1, x_2) = 19.5263$
$v_4 = 10001100010110100111100000111010 \mid x_1 = 5.2786 \quad x_2 = 5.5934 \mid eval(x_1, x_2) = f(x_1, x_2) = 17.4067$
$v_5 = 00011101100101001101011111000101 \mid x_1 = -1.2551 \quad x_2 = 4.7344 \mid eval(x_1, x_2) = f(x_1, x_2) = 25.3411$
$v_6 = 00010100001001010100101011111011 \mid x_1 = -1.8117 \quad x_2 = 4.3919 \mid eval(x_1, x_2) = f(x_1, x_2) = 18.1004$
$v_7 = 00100010000011010111011011111011 \mid x_1 = -0.9914 \quad x_2 = 5.5802 \mid eval(x_1, x_2) = f(x_1, x_2) = 16.0208$
$v_8 = 10000110000111010001011010100111 \mid x_1 = 4.9106 \quad x_2 = 4.7030 \mid eval(x_1, x_2) = f(x_1, x_2) = 17.9597$
$v_9 = 01000000010110001011000000111100 \mid x_1 = 0.7954 \quad x_2 = 5.3814 \mid eval(x_1, x_2) = f(x_1, x_2) = 16.1277$
$v_{10} = 00000111100011000001101000011011 \mid x_1 = -2.5548 \quad x_2 = 4.7937 \mid eval(x_1, x_2) = f(x_1, x_2) = 21.2784$
$v_{11} = 01100111111011010110000110111000 \mid x_1 = 3.1300 \quad x_2 = 4.9960 \mid eval(x_1, x_2) = f(x_1, x_2) = 23.4106$
$v_{12} = 11010001011110110100010101000000 \mid x_1 = 9.3561 \quad x_2 = 4.2394 \mid eval(x_1, x_2) = f(x_1, x_2) = 15.0116$
$v_{13} = 11101111101000100011000000100110 \mid x_1 = 11.1346 \quad x_2 = 5.3786 \mid eval(x_1, x_2) = f(x_1, x_2) = 27.3167$
$v_{14} = 01001001100000101010011100101001 \mid x_1 = 1.3359 \quad x_2 = 5.1513 \mid eval(x_1, x_2) = f(x_1, x_2) = 19.8762$
$v_{15} = 11101110110111000010001111011110 \mid x_1 = 11.0889 \quad x_2 = 5.0545 \mid eval(x_1, x_2) = f(x_1, x_2) = 30.0602$
$v_{16} = 11001111000001111110000110100101 1 \mid x_1 = 9.2115 \quad x_2 = 4.9937 \mid eval(x_1, x_2) = f(x_1, x_2) = 23.8672$
$v_{17} = 01101011111100111010001101111101 \mid x_1 = 3.3675 \quad x_2 = 4.5713 \mid eval(x_1, x_2) = f(x_1, x_2) = 13.6961$
$v_{18} = 01110100000000111010011110101101 \mid x_1 = 3.8430 \quad x_2 = 5.1582 \mid eval(x_1, x_2) = f(x_1, x_2) = 15.4141$
$v_{19} = 00010101001111111111000011000110 0 \mid x_1 = -1.7466 \quad x_2 = 5.3955 \mid eval(x_1, x_2) = f(x_1, x_2) = 20.0959$
$v_{20} = 10111001011001111001100010111111 0 \mid x_1 = 7.9359 \quad x_2 = 4.7573 \mid eval(x_1, x_2) = f(x_1, x_2) = 13.6669$

Total Fitness 387.7768

It is clear, that the chromosome v_{15} is the strongest one, and the chromosome v_2 the weakest.

66

Now we constructs a roulette wheel for the selection process. The total fitness of the population is: $F = \sum_{i=1}^{20} eval(v_i) = 387.7768$

The probability of a selection p_i and the cumulative probabilities q_i for each chromosome v_i (i=1,…20) are:

$p_1 = eval(v_1)/F = 26.0196/387.6822 = 0.0671 \quad q_1 = 0.0670$
$p_2 = eval(v_2)/F = 7.5800/387.6822 = 0.0195 \quad q_2 = 0.0866$
$p_3 = eval(v_3)/F = 19.5263/387.6822 = 0.0504 \quad q_3 = 0.1370$
$p_4 = eval(v_4)/F = 17.4067/387.6822 = 0.0449 \quad q_4 = 0.1819$
$p_5 = eval(v_5)/F = 25.3411/387.6822 = 0.0654 \quad q_5 = 0.2472$
$p_6 = eval(v_6)/F = 18.1004/387.6822 = 0.0467 \quad q_6 = 0.2939$
$p_7 = eval(v_7)/F = 16.0208/387.6822 = 0.0413 \quad q_7 = 0.3352$
$p_8 = eval(v_8)/F = 17.9597/387.6822 = 0.0463 \quad q_8 = 0.3815$
$p_9 = eval(v_9)/F = 16.1277/387.6822 = 0.0416 \quad q_9 = 0.4231$
$p_{10} = eval(v_{10})/F = 21.2784/387.6822 = 0.0549 \quad q_{10} = 0.4780$
$p_{11} = eval(v_{11})/F = 23.4106/387.6822 = 0.0604 \quad q_{11} = 0.5384$
$p_{12} = eval(v_{12})/F = 15.0116/387.6822 = 0.0387 \quad q_{12} = 0.5771$
$p_{13} = eval(v_{13})/F = 27.3167/387.6822 = 0.0775 \quad q_{13} = 0.6475$
$p_{14} = eval(v_{14})/F = 19.8762/387.6822 = 0.0513 \quad q_{14} = 0.6988$
$p_{15} = eval(v_{15})/F = 30.0602/387.6822 = 0.0775 \quad q_{15} = 0.7763$
$p_{16} = eval(v_{16})/F = 23.8672/387.6822 = 0.0615 \quad q_{16} = 0.8379$
$p_{17} = eval(v_{17})/F = 13.6961/387.6822 = 0.0353 \quad q_{17} = 0.8732$
$p_{18} = eval(v_{18})/F = 15.4141/387.6822 = 0.0397 \quad q_{18} = 0.9129$
$p_{19} = eval(v_{19})/F = 20.0959/387.6822 = 0.0518 \quad q_{19} = 0.9648$
$p_{20} = eval(v_{20})/F = 13.6669/387.6822 = 0.0352 \quad q_{20} = 1.0000$

67

Now we are ready to spin the roulette wheel 20 times: each time we select a single chromosome for a new population. Let us assume that a (random) sequence of 20 numbers fom range [0 1] is:

$r_1 = 0.5138$	$q_1 = 0.0670$	$v_1 = 1$	$v'_1 = v_{11}$
$r_2 = 0.1758$	$q_2 = 0.0866$	$v_2 = 0$	$v'_2 = v_4$
$r_3 = 0.3087$	$q_3 = 0.1370$	$v_3 = 0$	$v'_3 = v_7$
$r_4 = 0.5345$	$q_4 = 0.1819$	$v_4 = 2$	$v'_4 = v_{11}$
$r_5 = 0.9476$	$q_5 = 0.2472$	$v_5 = 1$	$v'_5 = v_{19}$
$r_6 = 0.1717$	$q_6 = 0.2939$	$v_6 = 1$	$v'_6 = v_4$
$r_7 = 0.7022$	$q_7 = 0.3352$	$v_7 = 1$	$v'_7 = v_{15}$
$r_8 = 0.2226$	$q_8 = 0.3815$	$v_8 = 1$	$v'_8 = v_5$
$r_9 = 0.4948$	$q_9 = 0.4231$	$v_9 = 1$	$v'_9 = v_{11}$
$r_{10} = 0.4247$	$q_{10} = 0.4780$	$v_{10} = 1$	$v'_{10} = v_{10}$
$r_{11} = 0.7039$	$q_{11} = 0.5384$	$v_{11} = 3$	$v'_{11} = v_{15}$
$r_{12} = 0.3890$	$q_{12} = 0.5771$	$v_{12} = 0$	$v'_{12} = v_9$
$r_{13} = 0.2772$	$q_{13} = 0.6475$	$v_{13} = 1$	$v'_{13} = v_6$
$r_{14} = 0.3681$	$q_{14} = 0.6988$	$v_{14} = 0$	$v'_{14} = v_8$
$r_{15} = 0.9834$	$q_{15} = 0.7763$	$v_{15} = 4$	$v'_{15} = v_{20}$
$r_{16} = 0.0054$	$q_{16} = 0.8379$	$v_{16} = 1$	$v'_{16} = v_1$
$r_{17} = 0.7656$	$q_{17} = 0.8732$	$v_{17} = 0$	$v'_{17} = v_{15}$
$r_{18} = 0.6446$	$q_{18} = 0.9129$	$v_{18} = 0$	$v'_{18} = v_{13}$
$r_{19} = 0.7671$	$q_{19} = 0.9648$	$v_{19} = 1$	$v'_{19} = v_{15}$
$r_{20} = 0.7802$	$q_{20} = 1.0000$	$v_{20} = 1$	$v'_{20} = v_{16}$

$q_{10} = 0.4780 < r_1 = 0.5138 < q_{11} = 0.5384$ *Select* v_{11} *chromosome*
$q_3 = 0.1370 < r_2 = 0.1758 < q_4 = 0.1819$ *Select* v_4 *chromosome*
$q_6 = 0.1717 < r_3 = 0.3087 < q_7 = 0.1819$ *Select* v_7 *chromosome*
$q_{10} = 0.4780 < r_4 = 0.5345 < q_{11} = 0.5384$ *Select* v_{11} *chromosome*
$q_{18} = 0.9129 < r_5 = 0.9476 < q_{19} = 0.9648$ *Select* v_{19} *chromosome*
$q_3 = 0.1370 < r_6 = 0.1717 < q_4 = 0.1819$ *Select* v_4 *chromosome*
$q_{14} = 0.6988 < r_7 = 0.7022 < q_{15} = 0.7763$ *Select* v_{15} *chromosome*
$q_4 = 0.1819 < r_8 = 0.2226 < q_5 = 0.2472$ *Select* v_5 *chromosome*
$q_{10} = 0.4780 < r_9 = 0.4948 < q_{11} = 0.5384$ *Select* v_{11} *chromosome*
$q_9 = 0.4231 < r_{10} = 0.4247 < q_{10} = 0.4780$ *Select* v_{10} *chromosome*
$q_{14} = 0.6988 < r_{11} = 0.7039 < q_{15} = 0.7763$ *Select* v_{15} *chromosome*
$q_8 = 0.3815 < r_{12} = 0.3890 < q_9 = 0.4231$ *Select* v_9 *chromosome*
$q_5 = 0.2472 < r_{13} = 0.2772 < q_6 = 0.2939$ *Select* v_6 *chromosome*
$q_7 = 0.3352 < r_{14} = 0.3681 < q_8 = 0.3815$ *Select* v_8 *chromosome*
$q_{19} = 0.9648 < r_{15} = 0.9834 < q_{20} = 1.0000$ *Select* v_{20} *chromosome*
$q_0 = 0.0000 < r_{16} = 0.0054 < q_1 = 0.0670$ *Select* v_1 *chromosome*
$q_{14} = 0.6988 < r_{17} = 0.7656 < q_{15} = 0.7763$ *Select* v_{15} *chromosome*
$q_{12} = 0.5771 < r_{18} = 0.6446 < q_{13} = 0.6475$ *Select* v_{13} *chromosome*
$q_{14} = 0.6988 < r_{19} = 0.7671 < q_{15} = 0.7763$ *Select* v_{15} *chromosome*
$q_{15} = 0.7763 < r_{20} = 0.7802 < q_{16} = 0.8379$ *Select* v_{16} *chromosome*

68

Genetic algorithms : Crossover

Now we are ready to apply the crossover operation to the individuals in the new population (vector v'_i). The probability of crossover P_c=0.25, so we expect that on average 25% of chromosomes (i.e., 5 out of 20) undergo crossover.
We proceed in the following way: Let us assume that a (random) sequence of 20 numbers from range [0 1] is:

$r_1 = 0.8230$
$r_2 = 0.1519$
$r_3 = 0.6255$
$r_4 = 0.3115$
$r_5 = 0.3490$
$r_6 = 0.9172$
$r_7 = 0.5198$
$r_8 = 0.4012$
$r_9 = 0.6068$
$r_{10} = 0.7854$
$r_{11} = 0.0315$
$r_{12} = 0.8699$
$r_{13} = 0.1665$
$r_{14} = 0.6745$
$r_{15} = 0.7584$
$r_{16} = 0.5819$
$r_{17} = 0.3892$
$r_{18} = 0.2002$
$r_{19} = 0.3557$
$r_{20} = 0.8269$

This mean that the chromosomes v'_2, v'_{11}, v'_{13} and v'_{18} were selected for crossover. For each of these two pairs, we generate a random integer number pos from range [1 32] (33 is the total length-number of bits-in chromosome). The number pos indicates the position of the crossing point. The first pair of chromosomes is:

$v'_2 = 10001100010110100111100000 1110010$
$v'_{11} = 11101110101110000100011111011110$

And the generated number
pos=9

$v''_2 = 10001100010110000100011111011110$
$v''_{11} = 11101110110110100111100000 1110010$

69

Genetic algorithms : Crossover

The second pair of chromosomes is:

And the generated number
pos=9.

$v'_{13} = 00010100001001010100 1010111111011$
$v'_{18} = 11101111101000100011 0000001000110$

$v''_{13} = 00010100001001010100 0000001000110$
$v''_{18} = 11101111101000100011 1010111111011$

The current version of the population

$v'_1 = 0110011111\ 1011010110\ 0001101111\ 000$
$v'_2 = 1000110001\ 0111000010\ 0011111011\ 110$
$v'_3 = 0010001000\ 0011010111\ 1011011111\ 011$
$v'_4 = 0110011111\ 1011010110\ 0001101111\ 000$
$v'_5 = 0001010100\ 1111111111\ 0000110001\ 100$
$v'_6 = 1000110001\ 0110100111\ 1000001110\ 010$
$v'_7 = 1110111011\ 0111000010\ 0011111011\ 110$
$v'_8 = 0001110110\ 0101001101\ 0111111000\ 101$
$v'_9 = 0110011111\ 1011010110\ 0001101111\ 000$
$v'_{10} = 0000011110\ 0011000001\ 1010000111\ 011$
$v''_{11} = 1110111011\ 0110100111\ 1000001110\ 010$
$v'_{12} = 0100000001\ 0110001011\ 0000001111\ 100$
$v''_{13} = 0001010000\ 1001010100\ 0000001000\ 110$
$v'_{14} = 1000011000\ 0111010001\ 0110101100\ 111$
$v'_{15} = 1011100101\ 1001111001\ 1000101111\ 110$
$v'_{16} = 1001101000\ 0000111111\ 1010011011\ 111$
$v'_{17} = 1110111011\ 0111000010\ 0011111011\ 110$
$v''_{18} = 1110111110\ 1000100011\ 1010111111\ 011$
$v'_{19} = 1110111011\ 0111000010\ 0011111011\ 110$
$v'_{20} = 1100111100\ 0001111110\ 0001010011\ 011$

70

Genetic algorithms : Mutation

The next operator, mutation, is performed on a bit-by-bit basis. The probability of mutation p_m=0.01, so we expect that (on average) 1% of bits would undergo mutation. There are
m x pop_size =33 x 20 = 660 bits in the whole population; we expect (on average) 6.6 mutations per generation. Every bit has an equal chance to be mutated, so, for every bit in the population, we generate a random number r from the range [0 1]; if r<0.01, we mutate the bit.
This means that we have to generate 660 random numbers. In a sample run, 5 of these numbers were smaller than 0.01; the bit number and the random number are listed below:

Bit position	Random number
112	0.0002
349	0.0099
418	0.0088
429	0.0054
602	0.0028

Bit position	Chromosome position	Bit number within chromosome
112	int(112/33) = 4	112-3x33 = 13
349	int(349/33) = 11	349-10x33 = 19
418	int(418/33) = 13	418-12x33 = 22
429	int(429/33) = 13	429-12x33 = 33
602	int(602/33) = 19	602-18x33 = 8

71

Genetic algorithms : Mutation

The current version of the population before mutation

v'_1 = 0110011111 1011010110 0001101111 000
v''_2 = 1000110001 0111000010 0011111011 110
v'_3 = 0010001000 0011010111 1011011111 011
v'_4 = 0110011111 1011010110 0001101111 000
v'_5 = 0001010100 1111111111 0000110001 100
v'_6 = 1000110001 0110100111 1000001110 010
v'_7 = 1110111011 0111000010 0011111011 110
v'_8 = 0001110110 0101001101 0111111000 101
v'_9 = 0110011111 1011010110 0001101111 000
v'_{10} = 0000011110 0011000001 1010000111 011
v''_{11} = 1110111011 0110100111 1000001110 010
v'_{12} = 0100000001 0110001011 0000001111 100
v''_{13} = 0001010000 1001010100 0000001000 110
v'_{14} = 1000011000 0111010001 0110101100 111
v'_{15} = 1011100101 1001111001 1000101111 110
v'_{16} = 1001101000 0000111111 1010011011 111
v'_{17} = 1110111011 0111000010 0011111011 110
v''_{18} = 1110111110 1000100011 1010111111 011
v'_{19} = 1110111011 0111000010 0011111011 110
v'_{20} = 1100111100 0001111110 0001101001 011

Chromosome position	Bit number within chromosome
4	13
11	19
13	22
13	33
19	8

The current version of the population after mutation

v'_1 = 0110011111 1011010110 0001101111 000
v''_2 = 1000110001 0111000010 0011111011 110
v'_3 = 0010001000 0011010111 1011011111 011
v'_4 = 0110011111 1001010110 0001101111 000
v'_5 = 0001010100 1111111111 0000110001 100
v'_6 = 1000110001 0110100111 1000001110 010
v'_7 = 1110111011 0111000010 0011111011 110
v'_8 = 0001110110 0101001101 0111111000 101
v'_9 = 0110011111 1011010110 0001101111 000
v'_{10} = 0000011110 0011000001 1010000111 011
v''_{11} = 1110111011 0110100101 1000001110 010
v'_{12} = 0100000001 0110001011 0000001111 100
v''_{13} = 0001010000 1001010100 0100001000 111
v'_{14} = 1000011000 0111010001 0110101100 111
v'_{15} = 1011100101 1001111001 1000101111 110
v'_{16} = 1001101000 0000111111 1010011011 111
v'_{17} = 1110111011 0111000010 0011111011 110
v''_{18} = 1110111110 1000100011 1010111111 011
v'_{19} = 1110111111 0111000010 0011111011 110
v'_{20} = 1100111100 0001111110 0001101001 011

72

We drop primes for modified chromosomes. the population is listed as new vector v_i:

$v_1 = 0110011111\ 1011010110\ 0001101111\ 000$	$eval(v_1) = f(3.1301, 4.9961) = 23.4107$
$v_2 = 1000110001\ 0111000010\ 0011111011\ 110$	$eval(v_2) = f(5.2790, 5.0545) = 18.2011$
$v_3 = 0010001000\ 0011010111\ 1011011111\ 011$	$eval(v_3) = f(-0.9915, 5.6803) = 16.0208$
$v_4 = 0110011111\ 1001010110\ 0001101111\ 000$	$eval(v_4) = f(3.1282, 4.9961) = 23.4126$
$v_5 = 0001010100\ 1111111111\ 0000110001\ 100$	$eval(v_5) = f(-1.7466, 5.3956) = 20.0959$
$v_6 = 1000110001\ 0110100111\ 1000001110\ 010$	$eval(v_6) = f(5.2787, 5.5935) = 17.4067$
$v_7 = 1110111011\ 0111000010\ 0011111011\ 110$	$eval(v_7) = f(11.0890, 5.0545) = 30.0602$
$v_8 = 0001110110\ 0101001101\ 0111111000\ 101$	$eval(v_8) = f(-1.2552, 4.7345) = 25.3412$
$v_9 = 0110011111\ 1011010110\ 0001101111\ 000$	$eval(v_9) = f(3.1301, 4.9961) = 23.4107$
$v_{10} = 0000011110\ 0011000001\ 1010000111\ 011$	$eval(v_{10}) = f(-2.5549, 4.7937) = 21.2784$
$v_{11} = 1110111011\ 0110100101\ 1000001110\ 010$	$eval(v_{11}) = f(11.0886, 4.7434) = 33.3518$
$v_{12} = 0100000001\ 0110001011\ 0000001111\ 100$	$eval(v_{12}) = f(0.7954, 5.3815) = 16.1278$
$v_{13} = 0001010000\ 1001010100\ 0100001000\ 111$	$eval(v_{13}) = f(-1.8118, 4.2099) = 22.6925$
$v_{14} = 1000011000\ 0111010001\ 0110101100\ 111$	$eval(v_{14}) = f(4.9106, 4.7030) = 17.9597$
$v_{15} = 1011100101\ 1001111001\ 1000101111\ 110$	$eval(v_{15}) = f(7.9360, 4.7573) = 13.6670$
$v_{16} = 1001101000\ 0000111111\ 1010011011\ 111$	$eval(v_{16}) = f(6.0844, 5.6522) = 26.0196$
$v_{17} = 1110111011\ 0111000010\ 0011111011\ 110$	$eval(v_{17}) = f(11.0890, 5.0545) = 30.0602$
$v_{18} = 1110111110\ 1000100011\ 1010111111\ 011$	$eval(v_{18}) = f(11.1346, 5.6670) = 27.5911$
$v_{19} = 1110111111\ 0111000010\ 0011111011\ 110$	$eval(v_{19}) = f(11.0595, 5.0545) = 27.6084$
$v_{20} = 1100111100\ 0001111110\ 0001101001\ 011$	$eval(v_{20}) = f(9.2116, 4.9937) = 23.8672$
	$Total = 447.0497$

73

Now, we are ready to run the selection process again and apply the genetic operator, evaluate the next generation, etc. After 100 generations the population is:

$v_1 = 1110111101\ 1001101110\ 0101010111\ 011$	$eval(v_1) = f(11.12, 5.0925) = 30.2985$
$v_2 = 1110011001\ 1000010001\ 0101010111\ 000$	$eval(v_2) = f(10.5888, 4.6674) = 26.8697$
$v_3 = 1110111101\ 1101101110\ 0101010111\ 011$	$eval(v_3) = f(11.1246, 5.0925) = 30.3166$
$v_4 = 1110011000\ 1000011000\ 0101010111\ 001$	$eval(v_4) = f(10.5741, 4.2442) = 31.9331$
$v_5 = 1110111101\ 1101101110\ 0101010111\ 011$	$eval(v_5) = f(11.1246, 5.0925) = 30.3166$
$v_6 = 1110011001\ 1000010000\ 0100010100\ 001$	$eval(v_6) = f(10.5888, 4.2146) = 34.3561$
$v_7 = 1101011000\ 1001001000\ 1100010110\ 000$	$eval(v_7) = f(9.6311, 4.4279) = 35.4586$
$v_8 = 1111011000\ 1000101000\ 1101010010\ 001$	$eval(v_8) = f(11.5181, 4.4528) = 23.3090$
$v_9 = 1110011000\ 1001001000\ 1100010110\ 001$	$eval(v_9) = f(10.5748, 4.4279) = 34.3938$
$v_{10} = 1110111101\ 1101101110\ 0101010111\ 011$	$eval(v_{10}) = f(11.1225, 5.0925) = 30.3166$
$v_{11} = 1101011000\ 0001001000\ 1100010110\ 000$	$eval(v_{11}) = f(9.6237, 4.4279) = 35.4779$
$v_{12} = 1101011000\ 1001001000\ 1100010110\ 001$	$eval(v_{12}) = f(9.6311, 4.42779) = 35.4561$
$v_{13} = 1110111101\ 1101101110\ 0101010111\ 011$	$eval(v_{13}) = f(11.1246, 5.0925) = 30.3166$
$v_{14} = 1110011001\ 1000010000\ 0101010111\ 011$	$eval(v_{14}) = f(10.5888, 4.2425) = 32.9321$
$v_{15} = 1110011010\ 1011100101\ 0100110110\ 001$	$eval(v_{15}) = f(10.6066, 4.6537) = 30.7468$
$v_{16} = 1110011001\ 1000010100\ 0100010100\ 001$	$eval(v_{16}) = f(10.5888, 4.2146) = 34.3595$
$v_{17} = 1110011001\ 1000010000\ 0101010111\ 011$	$eval(v_{17}) = f(10.5888, 4.2425) = 32.9321$
$v_{18} = 1110011001\ 1000010000\ 0101010111\ 001$	$eval(v_{18}) = f(10.5888, 4.2424) = 32.9567$
$v_{19} = 1111011000\ 1000101000\ 1110000010\ 001$	$eval(v_{19}) = f(11.5181, 4.4728) = 19.6697$
$v_{20} = 1110011001\ 1000010000\ 0101010111\ 001$	$eval(v_{20}) = f(10.5888, 4.2424) = 32.9567$
	$Total = 625.3728$

74

Schema Theorem

Schema Theorem (1975,1989)

- *Schemata that are short, low-order, and above average are given exponentially increasing numbers of trials in subsequent generations of a genetic algorithm.*

- theoretical foundations of genetic algorithms
- Introduced by *Holland* and popularized by *Goldberg*

Schema(ta)

- A schema is a similarity template which describes a subset of strings(chromosomes) with similarities at some string positions; i.e., it defines a subset of the search space.
  - For binary strings, it Consists of series of 1's, 0's, and ∗'s (don't care)
    - 1** Matches
      (100, 101, 110, 111)
    - ∗10** Matches
      (01000, 01001, 01010, 01011, 11000, 11001, 11011, 11011)
- A template allowing exploration of similarities among chromosomes.
- The plural of the word *schema* is *schemata*

77

Number of Schemata

- A binary string(chromosome) of length l can have membership in 2^l different schemata.
  - E.g., 101 has membership in schemata
    (101, 10∗, 1∗1, 1**, ∗01, ∗0∗, **1, ***)
- The number of Schemata for a chromosome of length l with alphabet of distinct characters k, is $(k+1)^l$:
  - Each position may contain 0, 1, … , k or ∗

$$\binom{l}{0}k^{l-0}+\binom{l}{1}k^{l-1}+\binom{l}{2}k^{l-2}+\cdots+\binom{l}{l}k^{l-l}=(k+1)^l$$

  - E.g., 3^2 schemata for binary strings (k=2) of length 2
    (00, 01, 0∗, 10, 11, 1∗, ∗0, ∗1, **)

78

Number of Schemata

- The Number of schemata for a population of N binary chromosomes range from 2^l (all strings are the same) to $N2^l$ (all strings have different schemata)
  - Actually, it will be always < Nk^l, because some schemata will not be represented and others will overlap(share) with other schemata
- In general, we want schemata whose instances have higher average fitnesses (even just in the current population in which they're instanced) to get more chance to reproduce. That's how we make the fittest survive!

  i.e. we eventually want to create a population that is full of fitter schemata and we will have lost weaker schemata.

Properties of Schemata

Schemata
S_a = 1***
S_b= 1**0
S_c= *001
S_d= 0*1*11*

- Order ≡O: number of fixed values(non-* symbols),
  - reflecting how large the covering regions of space
  - Affects the probability of a schema destroyed by a *mutation*
- Length ≡L: the difference between the first and the last fixed values (non-* symbols),
  - Affects the probability of a schema destroyed by a *crossover*

Properties of Schemata

- Fitness ≡F: the average fitness of all strings(choromosomes) in the population matched by a schema, S, at time t; is denoted by F(S,t), i.e.

$$F(S,t) = \frac{\sum_{i=1}^{\Phi} f(v_i)}{\Phi}$$

$\{v_1, v_2, \ldots, v_\Phi\}$: Φ strings in a population matched by S

What does Crossover do to Schemata

- Closer together loci are, less likely to be disrupted by crossover. A "compact representation" tends to keep alleles together under a given form of crossover (minimizes probability of disruption).

Example

- convincing you that probability of disruption by "1-point" crossover of schema *S* of length *L*(*S*) is *L*(*S*) /(*l*-1):

1****01**1

The Schema Theorem

- It is The Fundamental Theorem of Genetic Algorithms.
- It provides *lower bound* on change in sampling rate of a single schema from generation t to t+1. We'll consider it in several steps:
  - the change caused by selection alone,
  - the change caused by crossover,
  - the change caused by mutation,

GA - Schema Theorem - Implicit Parallelism

- If a given chromosome is of length l then it contains 2^l schemata (as each position can have the value 0, 1 or *).
- In theory, this means that for a population of N individuals we are evaluating up to $N2^l$ schemata.
- But, bear in mind that some schemata will not be represented and others will overlap with other schemata.
- This is exactly what we want. We eventually want to create a population that is full of fitter schemata and we will have lost weaker schemata.
- It is the fact that we are manipulating N individuals but $N2^l$ schemata that gives genetic algorithms what has been called ***implicit parallelism***.

GA - Schema Theorem - Intuition about length

- The longer the length of the schema, the more chance there is of the schema being disrupted by a crossover operation
- This implies that shorter schemata have *a better chance of surviving* from one generation to the next
- In turn, this implies that if we know that certain attributes of a problem fit well together then these should be placed as close as possible together in the coding

GA - Schema Theorem - Intuition about order

- This observation is also true for the order of the chromosome. If we are not worried about the number of defined positions (i.e. we allow as many '*' as possible) then a crossover operation has less chance of disrupting good schemata
- Intuitively, it would seem better to have *short*, *low-order* schema.
- This is only based on empirical evidence but it is widely believed that these assumptions are true and the following theory makes some sense of this.

GA - Schema Theorem

- Using a technique where we choose parents relative to their fitness (e.g. roulette wheel selection), fitter schema should find their way from one generation to another.
- Intuitively, if a schema is fitter than average then it should not only survive to the next generation but should also increase its presence in the population.
- If Φ is the number of instances of any particular schema S within the population at time t, then at $t+1$ we would expect

$$\Phi(S, t+1) > \Phi(S, t)$$

to hold for above average fitness schemata.

GA - Schema Theorem - Number of Schema

- Going one stage further we can estimate the number of schema present at $t+1$:

$$\Phi(S,t+1) = \Phi(S,t)\, n \frac{F(S,t)}{\sum f_i}$$

n : the size of the population

$f(S)$: the fitness of the schema

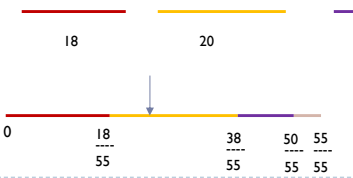
Σf_i : the fitness of the population

$$\Phi(S,t+1) = \Phi(S,t)\frac{F(S,t)}{f_{avg}}$$

f_{avg} : the average fitness of the population

ساز و کار چرخ رولت

$$P(v_i \in S \text{ is selected}) = \Phi(S,t) \quad \frac{F(S,t)}{\sum f_i}$$



Probability of non-disruption through crossover

- Given a schema, what is the probability of it ***not*** being disrupted by a crossover operation?

$$P_{NC} = 1 - \frac{L(S)}{l-1} P_C$$

P_C : the probability of crossover,

$L(s)$: the length of the schema,

l : the length of the chromosome,

Probability of non-disruption through crossover

- $L(s) = 4$ and $l = 11$
- Assume $P_C = 1$

The probability of the schema being not disrupted by a crossover operation is:

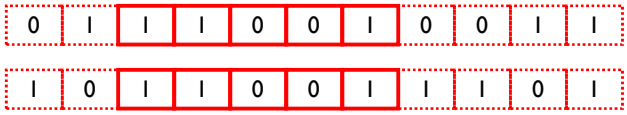
$$1 - 1 \times 4 / 10 = 0.6$$

$$P_{NC} = 1 - \frac{L(S)}{l-1} P_C$$

- We can easily confirm this by seeing that there are six crossover positions, of a possible ten (we assume we do not pick crossover points at the "outside") that will not disrupt the schema.

Probability of non-disruption through crossover

- But what if we crossover this schema with one that is the same?



Probability of non-disruption through mutation

For Mutation:

- We do not need worry about:
  - the length of the chromosome,
  - the length of the schema,
- We are concerned with the *order*.
  - For example, a schema of length 5 but only of order 3. It is only the bits that are defined within the schema that are of concern to us. The "don't care" (*'s) can be mutated without affecting the schema

Probability of non-disruption through mutation

- The probability of a single bit within a schema surviving mutation is:

$$1 - P_m$$

- The probability of surviving mutation is

$$(1 - P_m)^{O(S)}$$

which can be approximated to:

$$1 - P_m{}^{O(S)} \approx 1 - O(S).P_m$$

Probability of non-disruption through mutation

- Assume P_m = 0.01 then the probability of the below schema surviving is:

$$(1 - P_m)^{O(S)} = (1 - 0.01)^3 = 0.97$$

- If the schema had a higher order, say $O(S)$ = 100, then the probability of the schema surviving:

$$(1 - P_m)^{O(S)} = (1 - 0.01)^{100} = 0.366$$

demonstrating that short schema have a better chance of surviving.

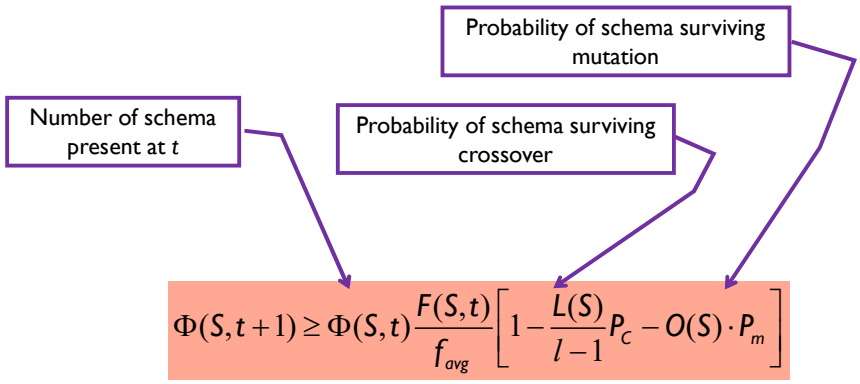
0	1	1	*	*	0	1	0	0	1	1

95

$$\Phi(S,t+1) \geq \Phi(S,t)\frac{F(S,t)}{f_{avg}}\left(1-\frac{L(S)}{l-1}P_c\right)(1 - O(S).P_m)$$

96

Schema Theory



97

Schema Theorem

- *Schemata that are short, low-order, and above average are given exponentially increasing numbers of trials in subsequent generations of a genetic algorithm.*

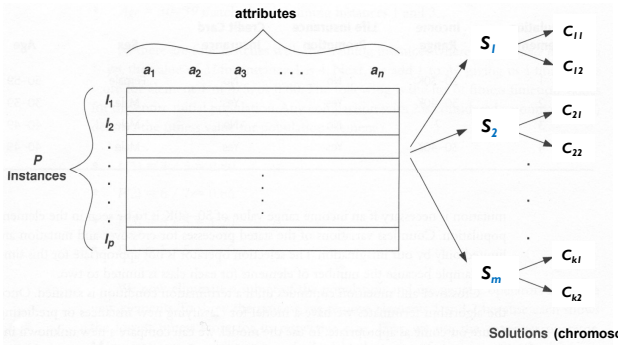
98

GA Algorithm

Input:
$D = \{t_1, t_2, ..., t_n\}$ *// Set of elements*
k *// Number of desired clusters.*
Output:
K *// Set of clusters.*
GA Clustering Algorithm:
randomly create an initial solution;
repeat
use crossover to create a new solution;
until *termination criteria is met;*

GA and Clustering

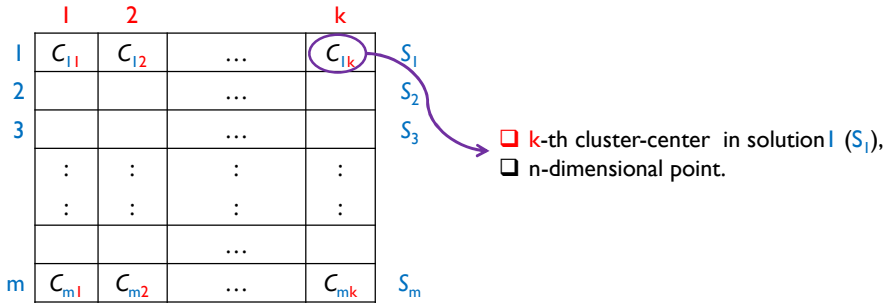
- Suppose
  - There are *P* data instances, where each one consists of *n* attributes.
  - *k* clusters are desired,
  - The method generates *m* possible solutions (chromosomes).
  - Each solution contains *k* n-dimensional point.



m solutions or chromosomes (population size =*m*):
k cluster-centers in each solution (or chromosome)

GA and Clustering

- Suppose
  - k clusters are desired,
  - the method generate m possible solutions (chromosomes).
  - Each solution contains k n-dimensional points (cluster-centers).
  - There are P data instances, where each one consists of n attributes.



101

- Crossover
  - Moving n-dimensional point from solution S_i to Solution S_j.
- Mutation
  - Swapping one or more points of the elements within S_i
- Fitness value for S_i
  - Inverse of Average Euclidean distance of P instances from their closest elements within S_i (or inverse of clustering error)

102

K-means input data (P=6 instances)

Instance	X	Y
1	1.0	1.5
2	1.0	4.5
3	2.0	1.5
4	2.0	3.5
5	3.0	2.5
6	5.0	6.0

103

	S_1	S_2	S_3
Solution elements (initial population)	(1.0,1.0) (5.0,5.0)	(3.0,2.0) (3.0,5.0)	(4.0,3.0) (5.0,1.0)
Fitness score	$(11.31)^{-1}$	$(9.78)^{-1}$	$(15.55)^{-1}$
Solution elements (second generation)	(5.0,1.0) (5.0,5.0)	(3.0,2.0) (3.0,5.0)	(4.0,3.0) (1.0,1.0)
Fitness score	$(17.96)^{-1}$	$(9.78)^{-1}$	$(11.34)^{-1}$
Solution elements (third generation)	(5.0,5.0) (1.0,5.0)	(3.0,2.0) (3.0,5.0)	(4.0,3.0) (1.0,1.0)
Fitness score	$(13.64)^{-1}$	$(9.78)^{-1}$	$(11.34)^{-1}$

$m = 3$ (no of solutions)
$k = 2$ (no of clusters)
$P = 6$ (no of instances)

104