Intelligent Control Systems (0640734)

Lecture (9)

Genetic Algorithms; an Introduction

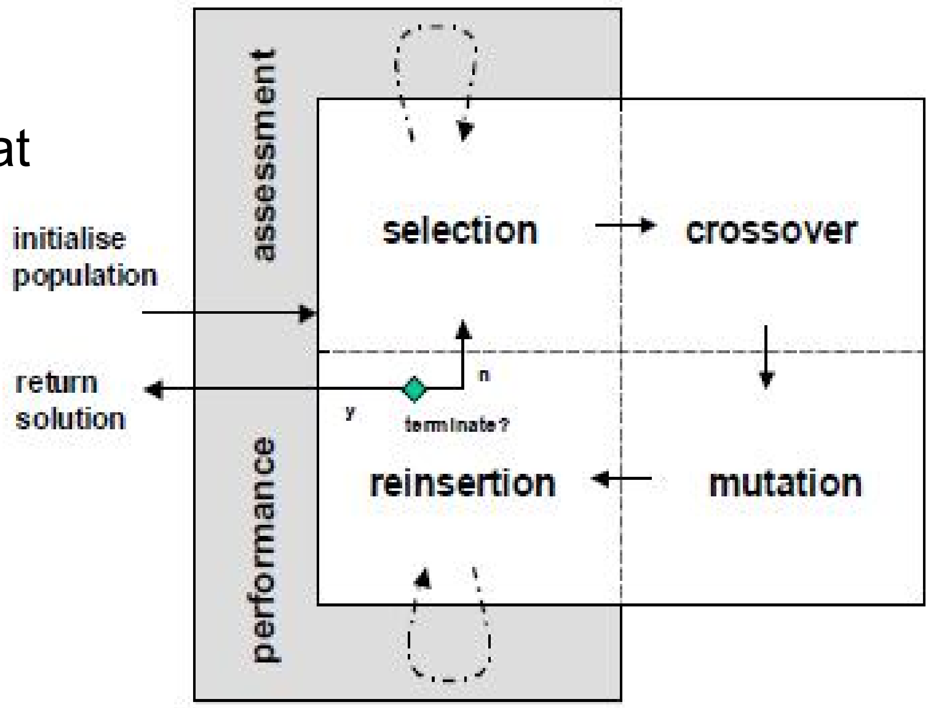
Prof. Kasim M. Al-Aubidy
Philadelphia University-Jordan

Introduction:

1. The genetic information for the construction of the individual is stored in the DNA. The human DNA genome consists of 46 chromosomes. There are about three billion nucleotides. These can be structured in genes, which carry one or more pieces information about the construction of the individual. However, it is estimated that only 3% of the genes carry meaningful information, the vast majority of genes is not used.
2. In the original idea, proposed by John Holland in 1975, the genetic information is encoded in a bit string of fixed length, called the parameter string or individual. Each parameter string represents a possible solution to the examined problem.
3. A Genetic Algorithm (GA) is a data mining technique. They are used to winnow relevant data from large data sets to produce the fittest solution.
4. GA is a computer program that simulates characteristics of evolution, natural selection, and genetics. It is an optimization technique that performs a parallel (i.e., candidate solutions are distributed over the search space) and stochastic but directed search to evolve the most fit population.

What are Genetic Algorithms?

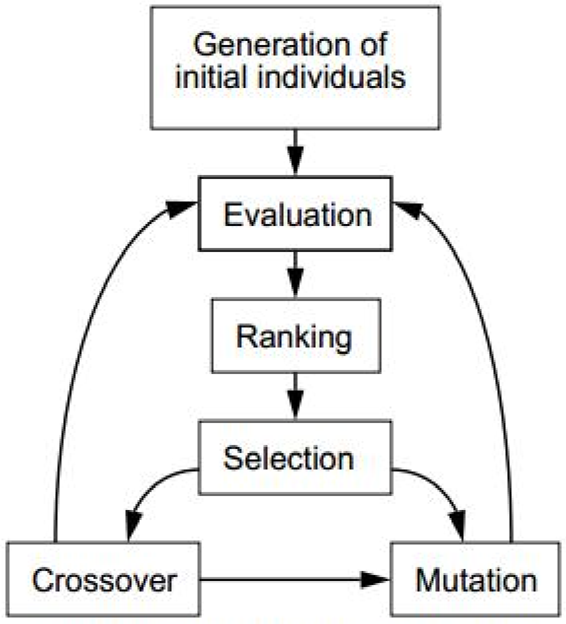
- Genetic algorithms are global, parallel, search and optimization methods, founded on Darwinian principles.
- GAs work with a population of potential solutions to a problem. Each individual within the population represents a particular solution to the problem, generally expressed in some form of genetic code. The population is evolved, over generations, to produce better solutions to the problem.
- Each individual within the population is assigned a **fitness value**, which expresses how good the solution is at solving the problem.
- The fitness value determines how successful the individual will be at propagating its genes (its code) to subsequent generations.
- Better solutions are assigned higher values of fitness than worse performing solutions.



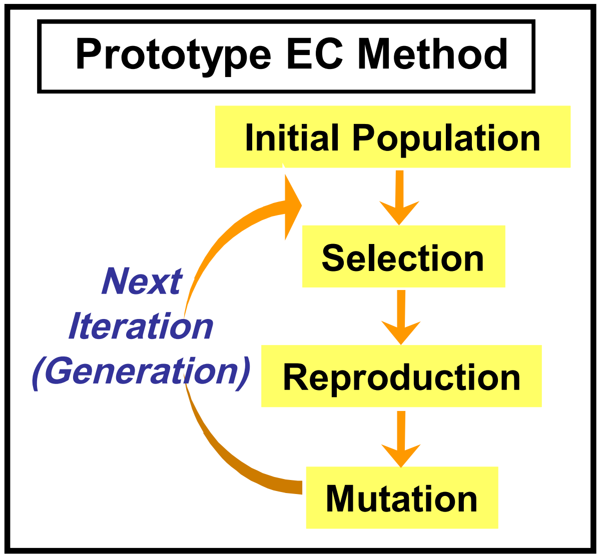
GA Structure:

The basic GA operators are **crossover**, **selection** and **mutation**.

- The GA starts with the random generation of an initial set of individuals (initial population).
- The individuals are evaluated and ranked. Since the number of individuals in each population is kept constant, for each new individual an old one (with the worst fitness value) has to be discarded.
- There are two basic operators to generate new individual: mutation and crossover.
- During mutation, a couple of bits of the parameter string are flipped at random. Mutation may be applied to offspring produced by crossover or, as an independent
- operator, at random to any individual in the population.

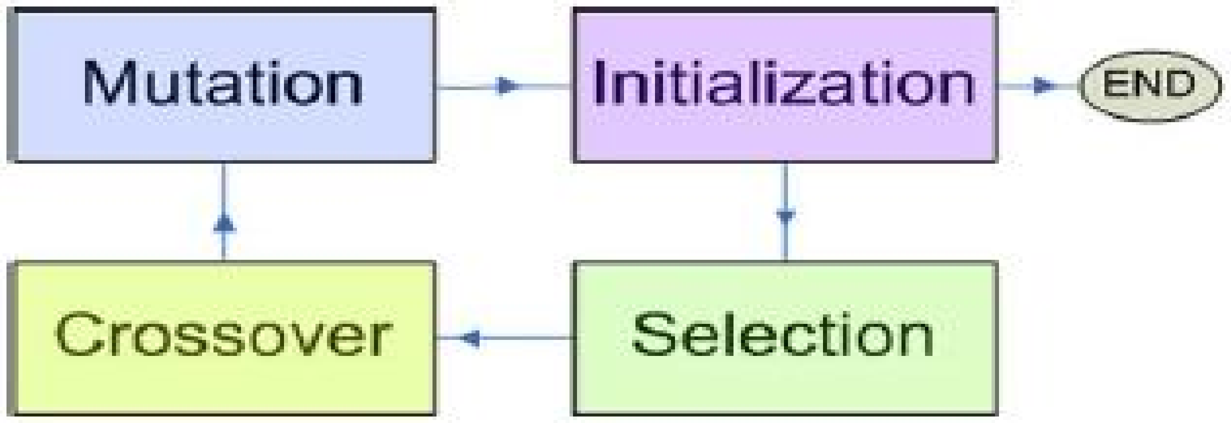


The Principle Structure of a GA



GA Stages:

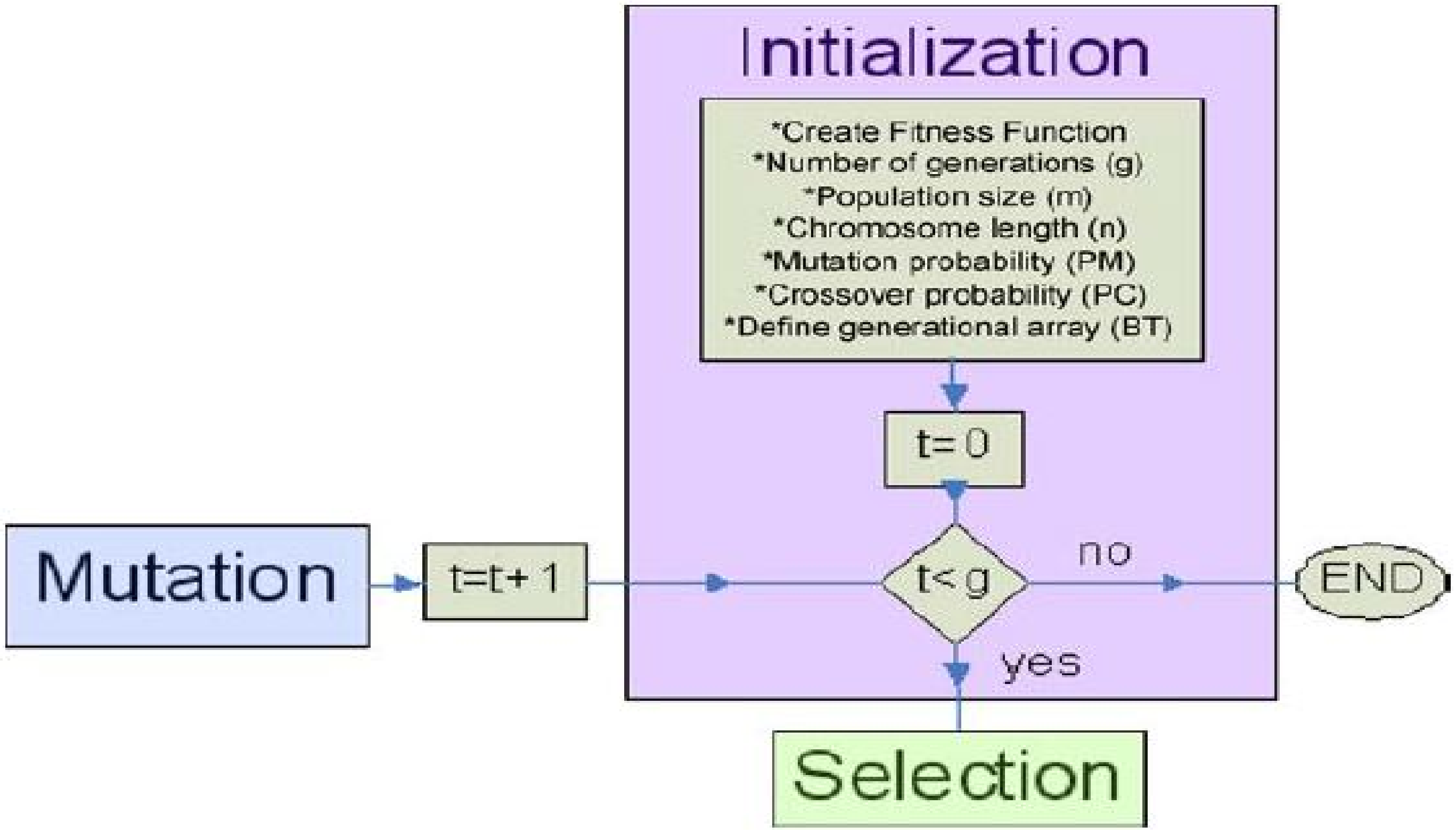
A GA can be divided into four main stages:



Initialization: The initialization of the necessary elements to start the algorithm.
Selection: This operation selects chromosomes in the population for reproduction by means of evaluating them in the fitness function. The fitter the chromosome, the more times it will be selected.
Crossover. Two individuals are selected and then a random point is selected and the parents are cut, then their tails are crossed.
Mutation. A gene, usually represented by a bit is randomly complemented in a chromosome, the possibility of this happening is very low because the population can fall into chaotic disorder.

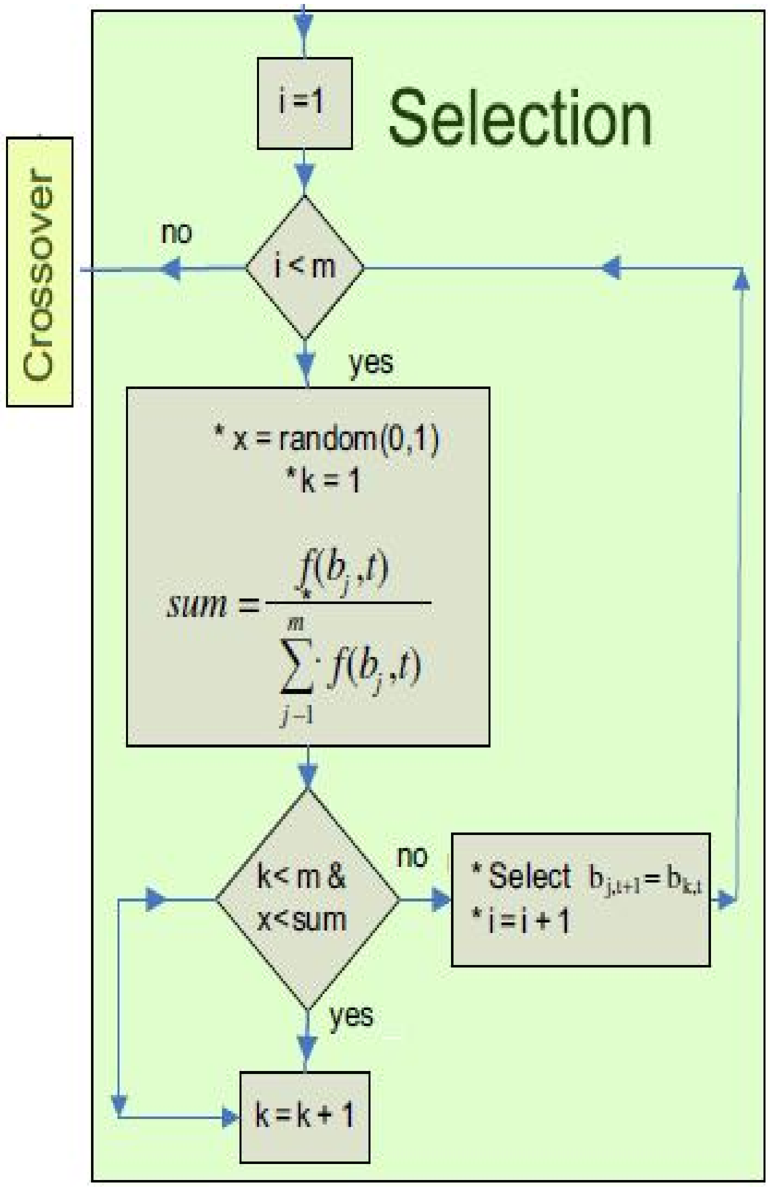
Initialization Stage:

In this stage, the initial individuals are generated, and the constants and functions are also initiated.



Selection Stage:

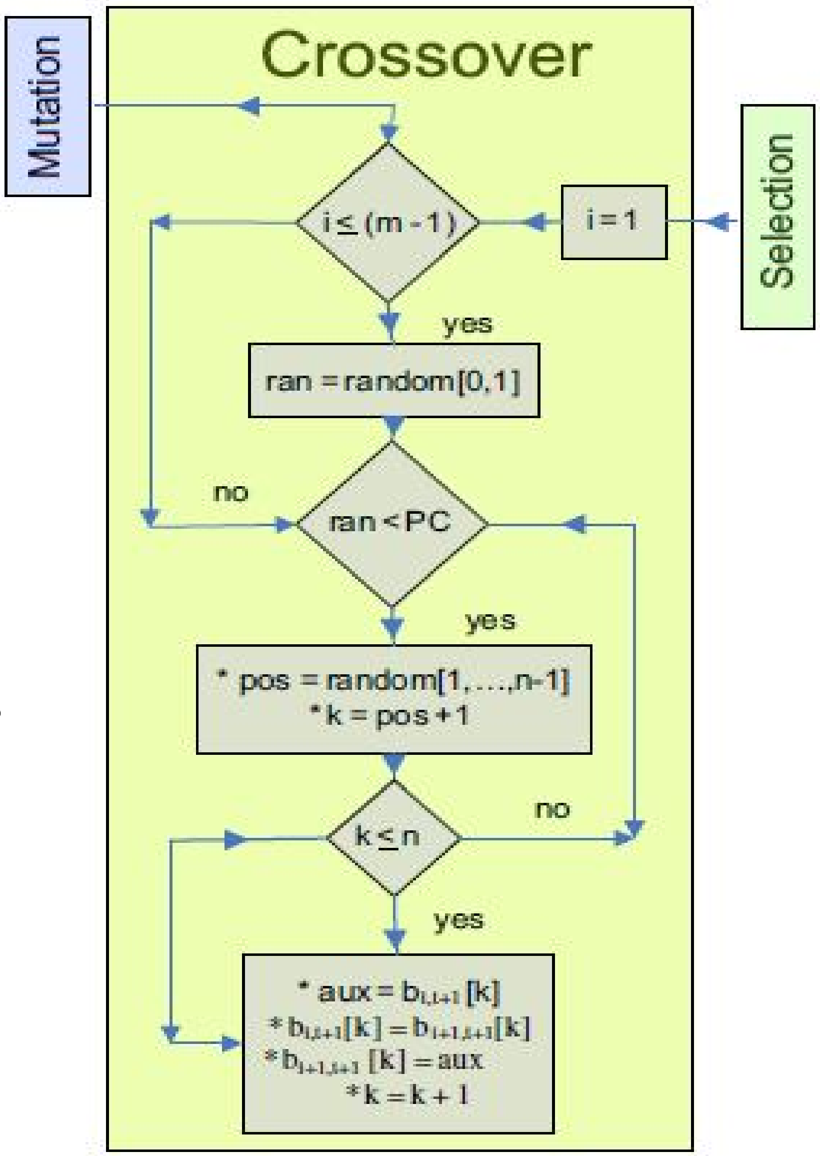
- Selection is a mechanism by which the “parents” are chosen for producing offspring to be passed into next generation.
- Selection tends to pick best population elements as parents. A careful selection of the individuals must be performed because missing a single high-fit individual may sometimes mislead the search process.
- It is necessary to introduce a measure of the performance of individuals. By selection we aim to maximize the performance of individuals.
- A performance value is associated with each individual in the current population, and represents the fitness of the function.
- A **fitness function** is used to measure explicitly the performance of chromosomes.



Selection
Crossover
i=1
i<m
no
yes
* x = random(0,1)
*k = 1
$sum = \frac{f(b_j^*,t)}{\sum_{j=1}^{m} f(b_j,t)}$
k<m & x<sum
no
yes
* Select $b_{j,t+1} = b_{k,t}$
* i=i+1
k=k+1

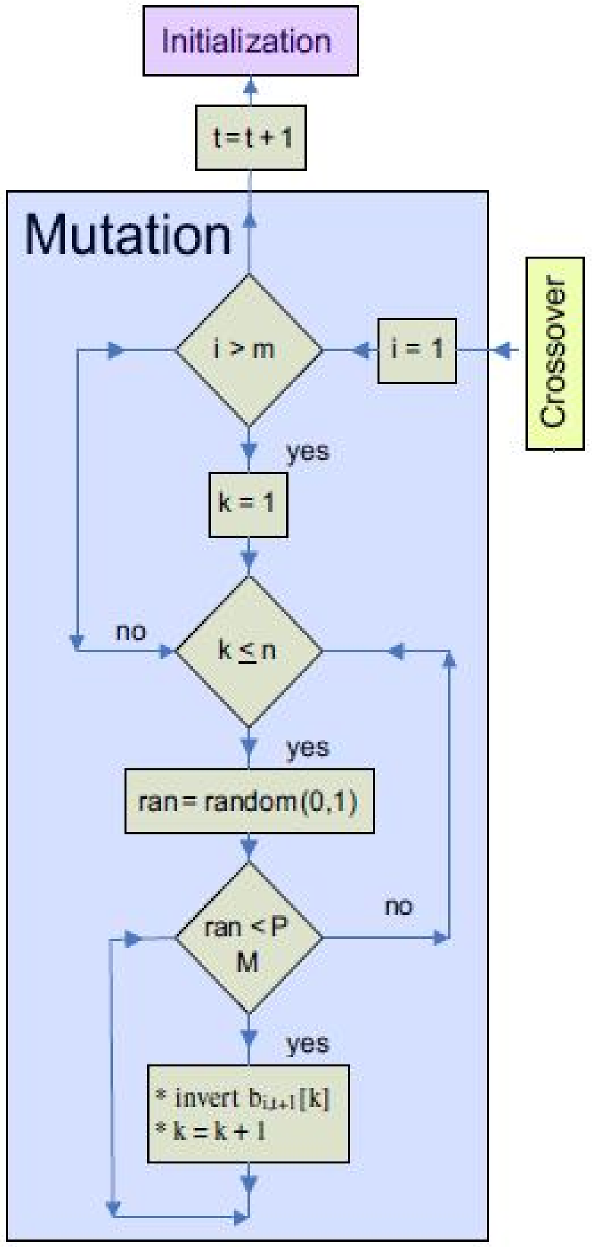
Crossover Stage:

- Crossover operation is used to increase population diversity.
- Crossover operations achieve the recombination of the selected individuals by combining segments belonging to chromosomes corresponding to parents.
- The crossover probability (CP) is compared with a random number between (0,1) to check if is going to be crossover or not. When a crossover is made, the positions in which the parents are going to be cut in a random position are then interchanged.



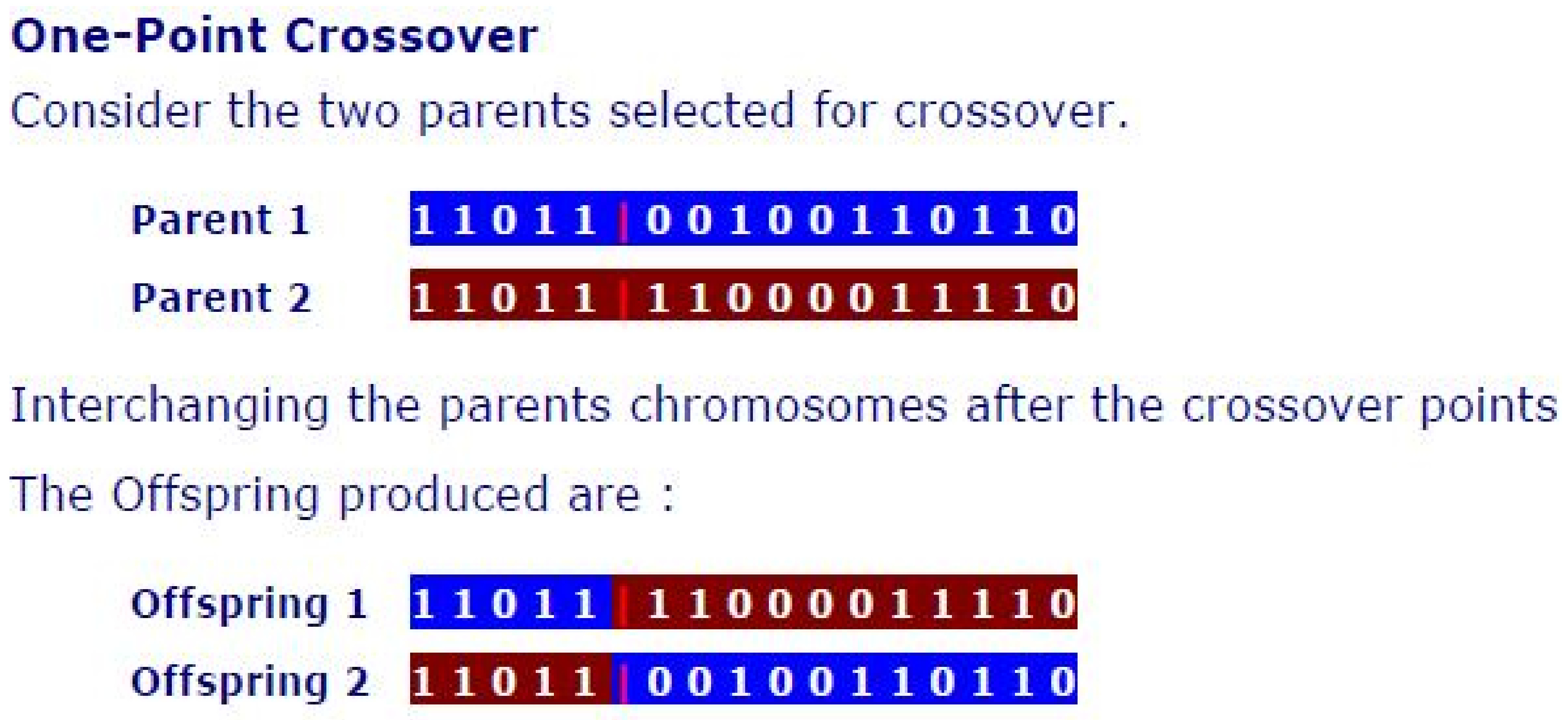
Mutation Stage:

- In classical genetics, mutation is identified by an altered phenotype, and in molecular genetics mutation refers to any alternation of a segment of DNA.
- Mutation makes "slight" random modifications to some or all of the offspring in next generation.
- The effect of this operator is to change a single position (gene) within a chromosome. If it were not for mutation, other individuals could not be generated through other mechanisms, which are then introduced to the population.
- One of the simplest executions of mutation is when the mutation probability (MP) is compared with a random number between (0,1). If it is going to be a mutation, a randomly chosen bit of the string is inverted.



Crossover:

Crossover is performed by taking parts of the bit-string of one of the parents and the other parts from the other parent and combining both in the child. There a three basic kinds of crossover: one-point, two-point and uniform.



One-Point Crossover

Consider the two parents selected for crossover.

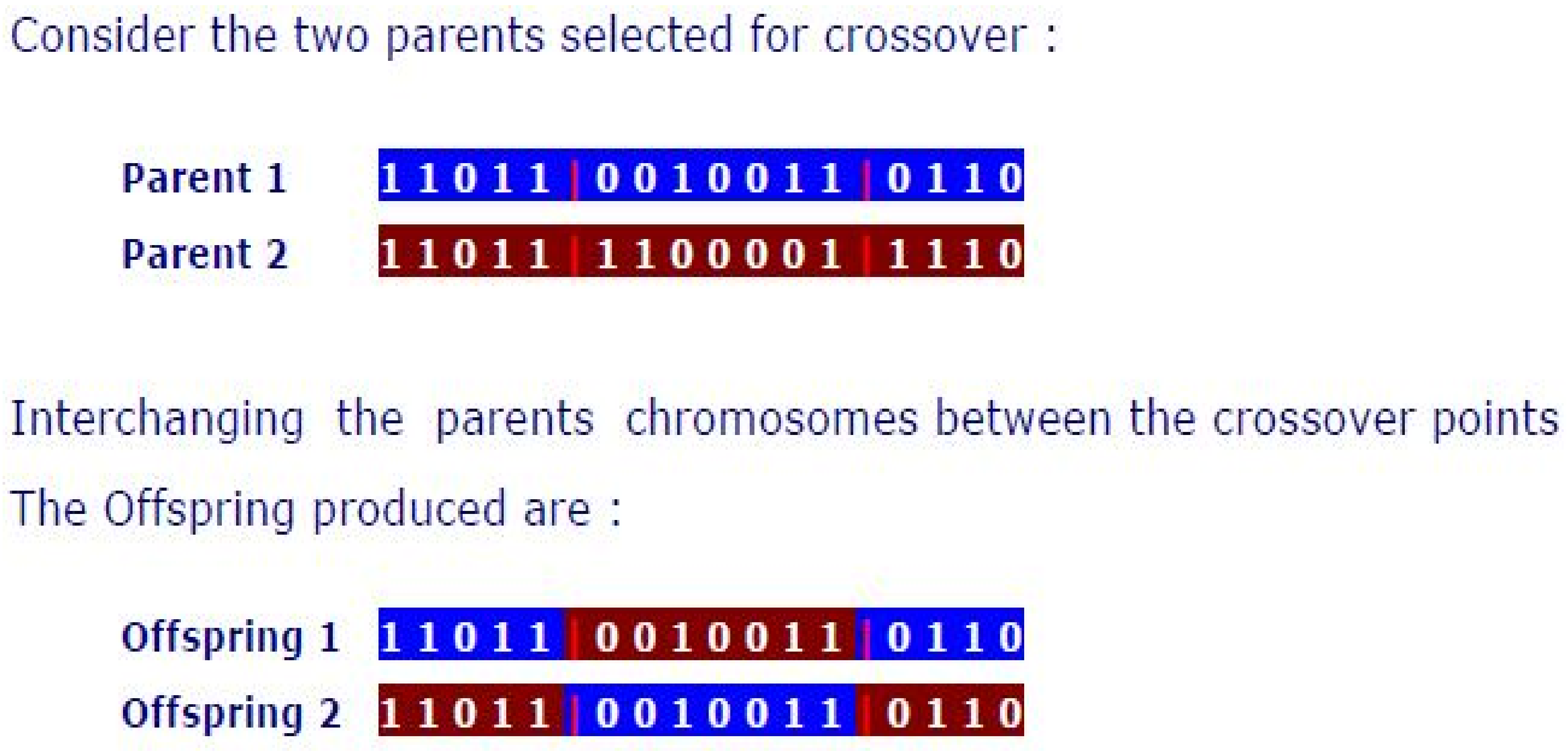
Parent 1	1 1 0 1 1 | 0 0 1 0 0 1 1 0 1 1 0
Parent 2	1 1 0 1 1 | 1 1 0 0 0 0 1 1 1 1 0

Interchanging the parents chromosomes after the crossover points

The Offspring produced are :

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

Two-point crossover differs from the previous version merely in the point that two random cuts are made, so three pieces have to be put together in order to produce an offspring.



Consider the two parents selected for crossover :

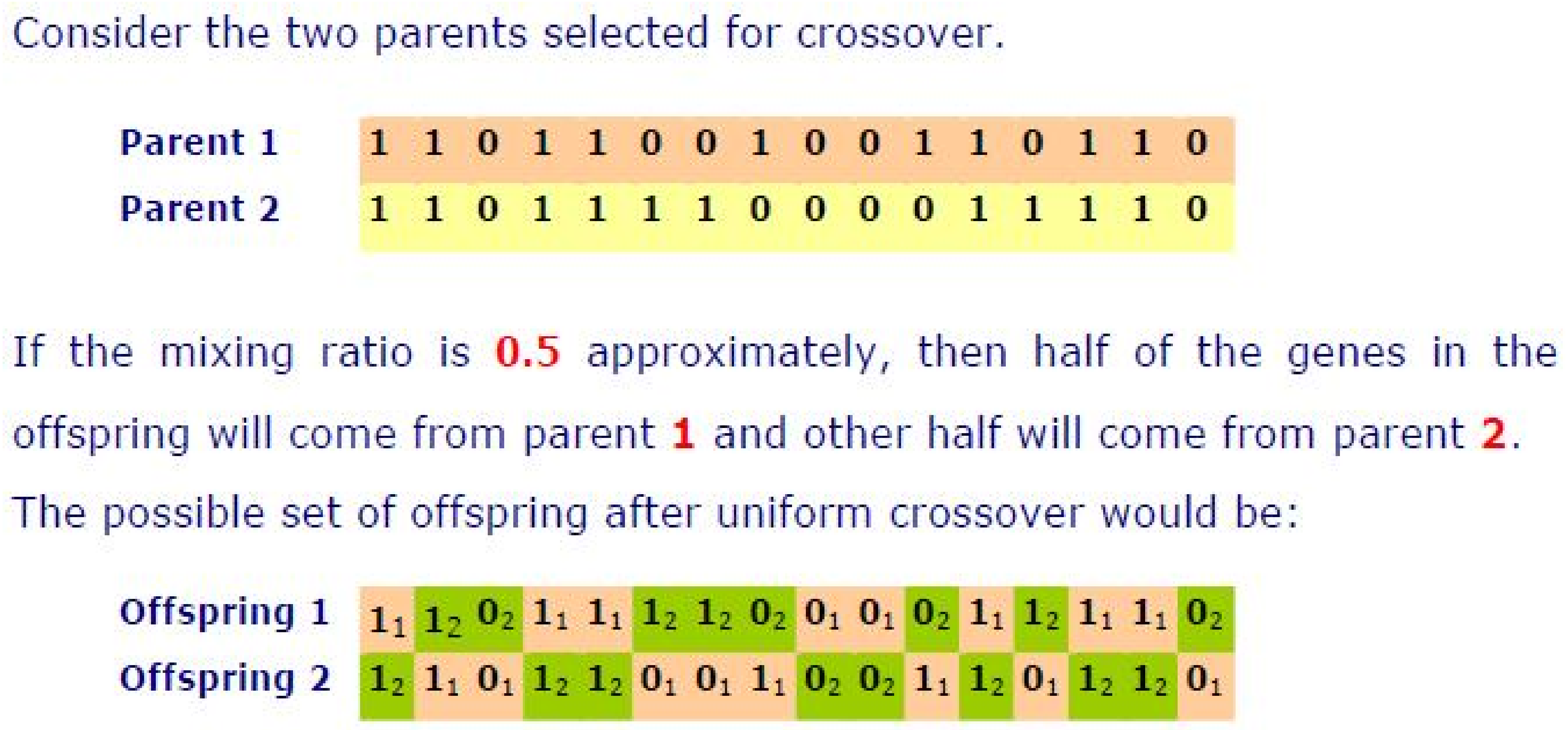
Parent 1	1 1 0 1 1 | 0 0 1 0 0 1 1 | 0 1 1 0
Parent 2	1 1 0 1 1 | 1 1 0 0 0 0 1 | 1 1 1 0

Interchanging the parents chromosomes between the crossover points

The Offspring produced are :

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

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



Consider the two parents selected for crossover.

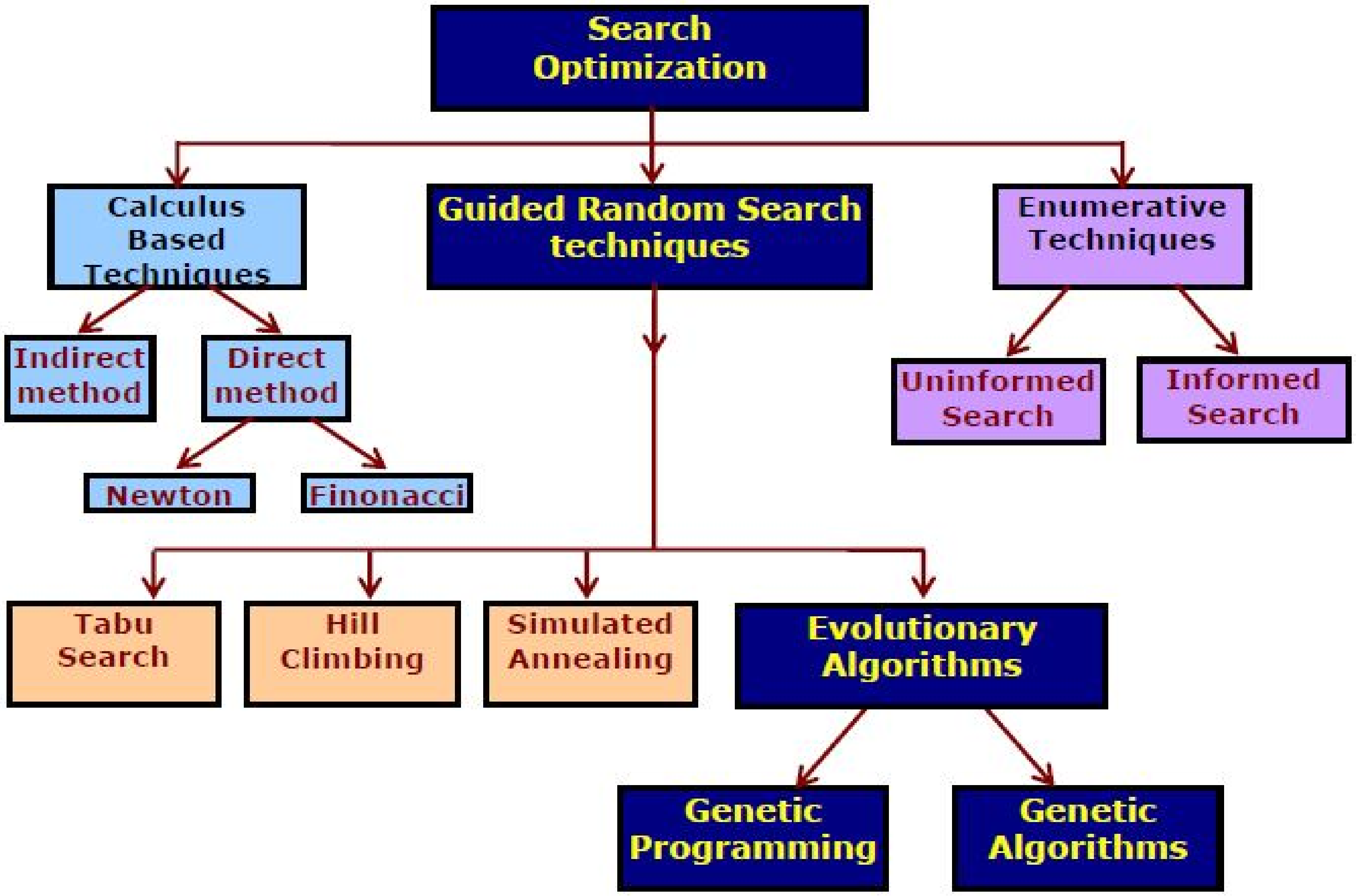
Parent 1	1	1	0	1	1	0	0	1	0	0	1	1	0	1	1	0
Parent 2	1	1	0	1	1	1	1	0	0	0	0	1	1	1	1	0

If the mixing ratio is **0.5** approximately, then half of the genes in the offspring will come from parent **1** and other half will come from parent **2**.

The possible set of offspring after uniform crossover would be:

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

Search Optimization Techniques:



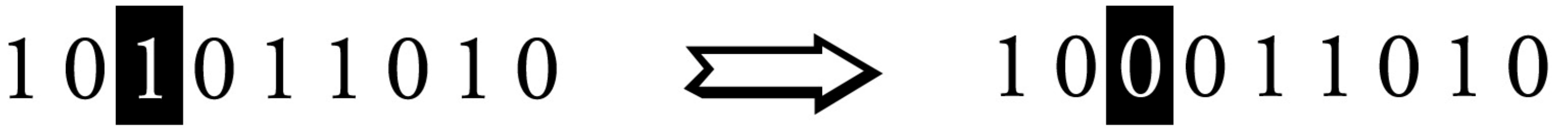
Selection: (pick best population elements as parents)

- ➢ Selection is the mechanism by which the "parents" are chosen for producing offspring to be passed into next generation.
- ➢ Parent selection methods based on probability of selection being increasing function of fitness

- **Roulette-wheel selection:** (common method)
  - Probability an individual is selected is equal to its fitness divided by the total fitness in the population.
    - ➢ Problem: Selection probability highly dependent on units and scaling for fitness function
  - May cause premature convergence to local optima.
- **Rank selection** and **Tournament selection** methods reduce sensitivity to choice of fitness function
  - More robust: Only compare which chromosomes are better, not relative magnitudes of fitness functions

Mutation:

- Mutation makes "slight" random modifications to some or all of the offspring in next generation.
- Mutation generally makes only small changes to solution.
- Bit-based coding and real (floating point) coding require different type of mutation
  1. Bit-based mutation generally involves "flipping" bit(s).
  2. Real-based mutation often involves adding small (Monte Carlo) random vector to chromosomes
- Example: Mutation on one element in chromosome in bit-based coding:



Design Concerns:

1. It is important to fully understand the optimization problem, know what you want to optimize, and what you can change to achieve the optimization. You also must have an idea of what you will accept as an optimal solution.
2. Choice of representation (e.g., the number of digits in a base-10 representation) is important. Too detailed of a representation causes increases in computational complexity while if the representation is too coarse then you may not be able to achieve enough accuracy in your solution.
3. There are a wide range of other genetic operators and choosing the appropriate ones is important since they can affect convergence significantly.
4. It is important to pick a good termination method.
5. It is difficult to guarantee that you will achieve convergence due to the presence of local maxima.
6. It can be difficult to select the best solution from the many candidate solutions that exist.

Essential Steps of Basic GA:

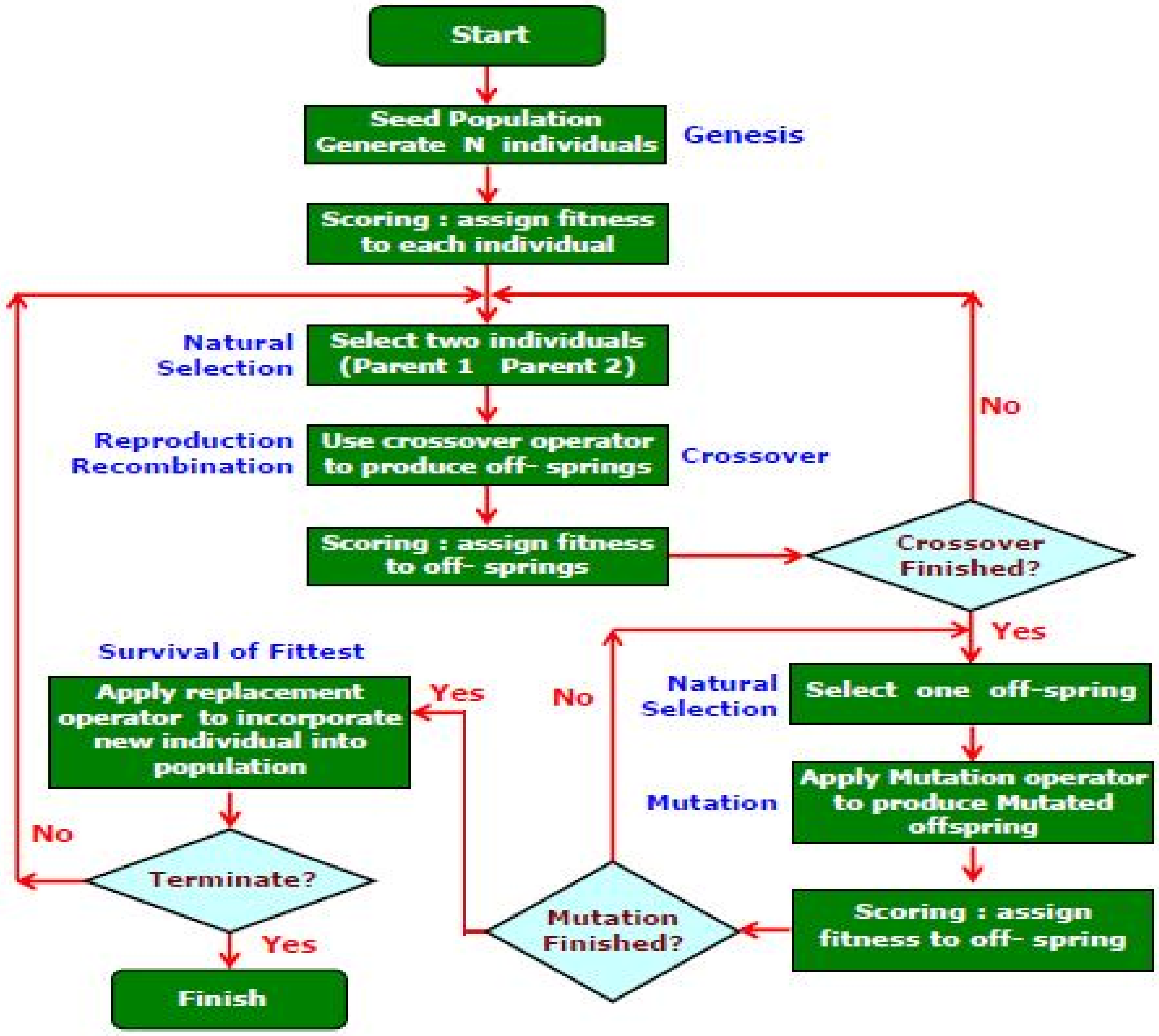
Step 0 (initialization): Randomly generate initial population of N (say) chromosomes and evaluate fitness function.

Step 1 (parent selection): Select with replacement $N - N_e$ parents from full population.

Step 2 (crossover): For each pair of parents identified in step 1, perform crossover on parents at a randomly chosen splice point (or *points* if using multi-point crossover) with probability P_c.

Step 3 (replacement and mutation): Replace the non-elite $N - N_e$ chromosomes with the current population of offspring from step 2. Perform mutation on the bits with a small probability P_m.

Step 4 (fitness and end test): Compute the fitness values for the new population of N chromosomes. Terminate the algorithm if stopping criterion or budget of fitness function evaluations is met; else return to step 1.


Start
Seed Population Generate N individuals
Genesis
Scoring : assign fitness to each individual
Natural Selection
Select two individuals (Parent 1 Parent 2)
Reproduction Recombination
Use crossover operator to produce off- springs
Crossover
Scoring : assign fitness to off- springs
Crossover Finished?
No
Yes
Natural Selection
Select one off-spring
Mutation
Apply Mutation operator to produce Mutated offspring
Scoring : assign fitness to off- spring
Mutation Finished?
No
Yes
Survival of Fittest
Apply replacement operator to incorporate new individual into population
Terminate?
No
Yes
Finish

Main Difficulties of Genetic Algorithms:

- Adjustment of the GA control parameters
    - ✓ population size
    - ✓ crossover probability
    - ✓ mutation probability
- Specification of the termination condition
- Representation of the problem solutions

Chromosome:

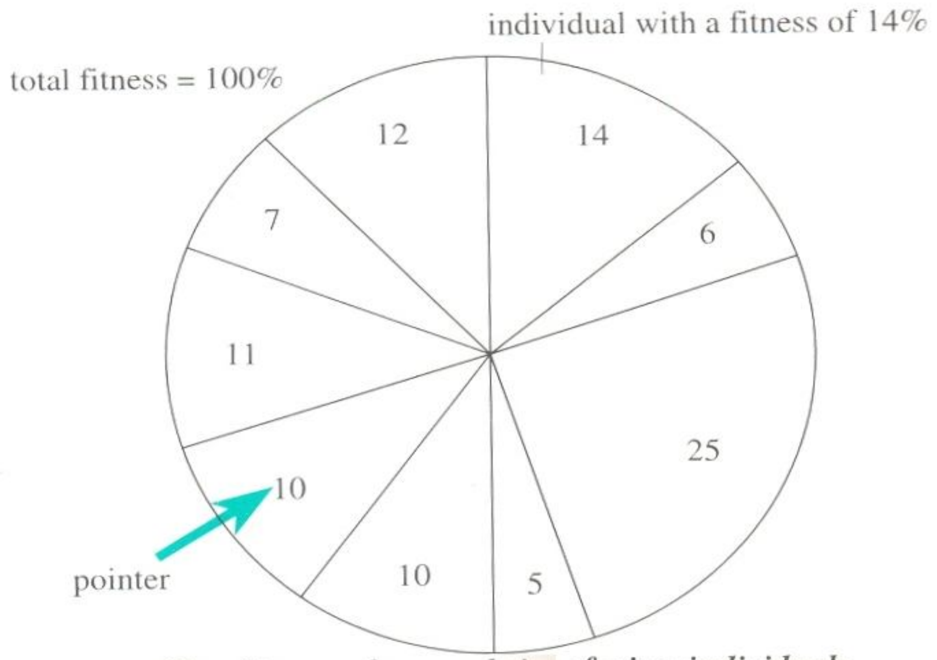
In GAs, a **chromosome** is a set of parameters which define a proposed solution to the problem that the GA is trying to solve.
The chromosome is often represented as a simple string, although a wide variety of other data structures are also used.

Chromosome Design:
The design of the chromosome and its parameters is by necessity specific to the problem to be solved.

Example:
Suppose the problem is to find the integer value of x between 0 and 255 that provides the maximal result for *f(x) = SQR(x).*
Our possible solutions are the integers from 0 to 255, which can all be represented as 8-bit binary strings.
Use an 8-bit binary string as our chromosome. If a given chromosome in the population represents the value 155, its chromosome would be 10011011.

Roulette-wheel selection:

- Chromosomes are the essential features of a GA, since they contain the genetic information. These are strings of data that define a particular solution.
- For example: a chromosome representing six genes might be specified as a 6-bit number. A population of these chromosomes is created, and then we need some way of measuring the fitness of the chromosomes to select the good solutions to be parents.



The fitness of a population of nine individuals.

- The mechanism for producing a new population from the current one is called Breeding.
- **Parents** are chosen in proportion to their fitness using a mechanism called **Roulette-wheel selection**. Each chromosome has a fitness, which can be regarded as a portion of the total fitness of the population.

In fitness-proportionate selection :

- the chance of an individual's being selected is proportional to its fitness, greater or less than its competitors' fitness.
- conceptually, this can be thought as a game of Roulette.

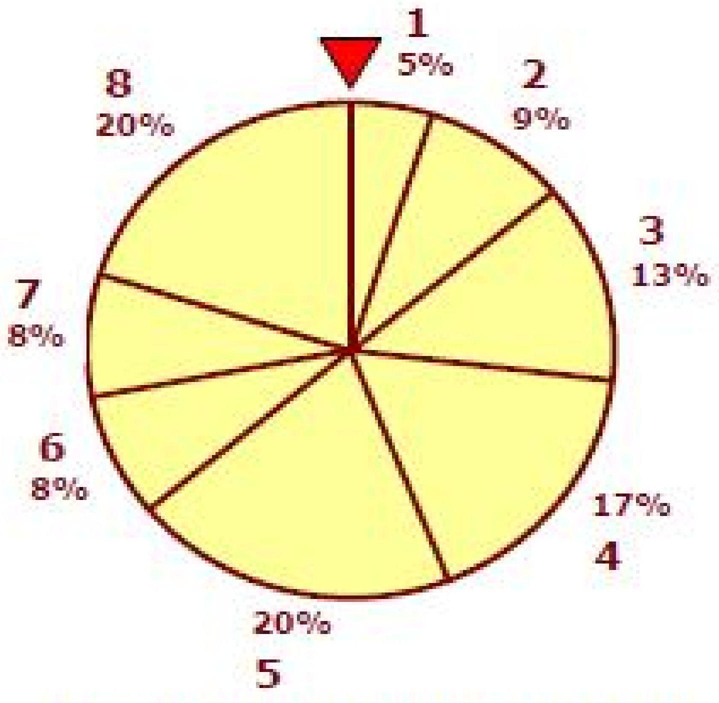


Fig. Roulette-wheel Shows 8 individual with fitness

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

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

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

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

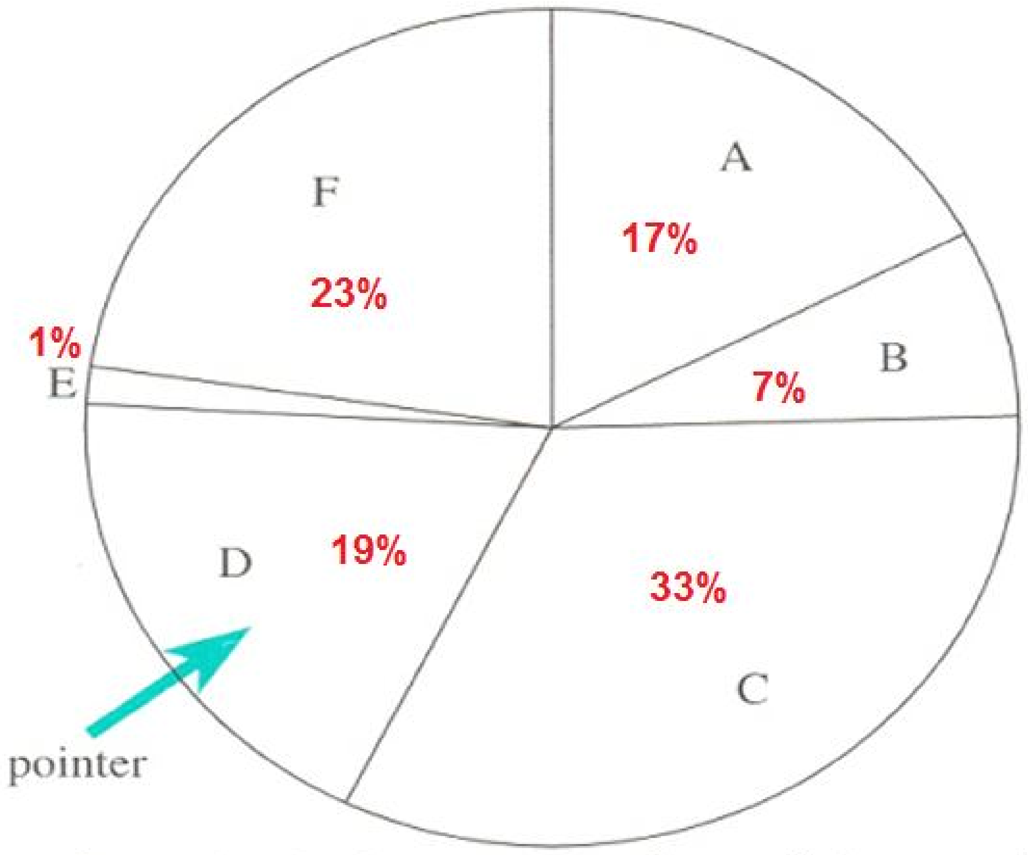
Because the circumference of the wheel is marked according to a string's fitness, the Roulette-wheel mechanism is expected to make $\frac{F}{\bar{F}}$ copies of the i th string.

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

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

Let's look at a simple example. Suppose there are 6 individuals in a population. Initially they are randomly generated and their fitness calculated, as in Table.

Individual	Fitness	Running total
A	12	12
B	5	12 + 5 = 17
C	23	17 + 23 = 40
D	13	40 + 13 = 53
E	1	53 + 1 = 54
F	16	54 + 16 = 70



Roulette wheel of the six members of the population.

The total fitness is 70, so the fitness of A as a proportion is 12/70, about 17%. The roulette wheel is a good way of imagining what is going on in selection. What actually happens is that a random number is generated between 0 and the total fitness, in this example 70. The number generated is compared with the running total, shown in Table.

The random number is then selected. For example, if the random number generated is 45, D would be chosen because D is the first individual found when scanning down the table which has a running total, 53, which is greater than the random number.

In this example there is a population of 6, so a random number is generated six times. Let's assume that the six numbers generated are 45, 23, 31, 57, 4 and 55. The corresponding individuals would be D, C, C, F, A and F. These are the parents of the new population.

Offspring are produced by selecting pairs of parent chromosomes and ***crossing over*** some of the genetic material. In the example just given, the parents would be paired D and C, C and F, and finally A and F. Each pair of parents then produces two offspring. It is permissible for two parents to be the same, such as A and A, even though the offspring are also A and A.

parent 1	**1**	**1**	**0**	**0**	**1**	**1**	offspring 1	**1**	**1**	**0**	0	1	0
parent 2	1	0	1	0	1	0	offspring 2	1	0	1	**0**	**1**	**1**

crossover point

Breeding using single-point crossover.

This form of crossover is called ***single-point crossover***. The actual point at which crossover takes place is randomly chosen. Other forms of crossover exist such as two-point crossovers, but we will only use single-point in this book.
Again, let's take our six chromosomes, and let's assume that they have the following binary structure:

A	1	1	0	0	1	1
B	1	0	1	0	1	1
C	0	1	0	1	0	1
D	1	1	1	0	1	0
E	0	0	0	0	0	1
F	0	1	0	0	1	0

Crossover takes place by generating a random number that corresponds to the position along the chromosome.

If the random numbers are 3, 1 and 2, then crossover takes place after bit 3 in the first pair of parents, after bit 1 in the second pair and after bit 2 in the third pair. Crossover would then look like this (one parent in each pair is shown in bold to show where the genetic material comes from):

	parents						offspring					
	5	4	3	2	1	0	5	4	3	2	1	0
D	1	1	1	0	1	0	1	1	**0**	**1**	**0**	**1**
C	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	1	0	1	0
C	0	1	0	1	0	1	0	1	0	1	**1**	**0**
F	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	0	1
A	1	1	0	0	1	1	1	1	0	**0**	**1**	**0**
F	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	0	1	1
	before crossover						after crossover					

Breeding using single-point crossover.

In addition to crossover, ***mutation*** is allowed. This happens when some of the genetic material changes randomly, as shown

chromosome 1 1 1 0 0 1 **0** 1 0 1 0 1 0

$\downarrow$

mutation

$\downarrow$

chromosome 1 1 1 0 0 1 **1** 1 0 1 0 1 0

Mutation is usually defined by the ***mutation rate***, which is normally set to quite a low value, 0.001 say. This corresponds to one change in a thousand bits of data. The bits that are actually mutated are randomly selected. After a bit has been selected for mutation it is inverted, so a 0 becomes a 1 and vice versa.

In our simple example, the new population, after mutation, might look something like

A′	1	1	0	1	0	1
B′	0	1	**0**	0	1	0
C′	0	1	0	1	1	0
D′	0	1	0	0	0	1
E′	1	1	0	0	1	**1**
F′	0	1	0	0	1	1

Example of population after mutation.

The actual fitness function used was

$$\text{fitness} = \frac{60}{x} - 6$$

where x is one of the values given in Table. This fitness function ensures that when the value of x is 10, the fitness is 0, and that the largest values of fitness occur at the minima.

After running the algorithm for 2000 trials, the statistics for the number of times the local and global minima were found for each generation were

Generation	Found global minimum	Found local minima	Failed to find any minimum
1	10%	20%	70%
2	15%	30%	55%
3	17%	31%	52%
4	21%	31%	48%
5	21%	35%	44%
6	21%	35%	44%

Example: A Simple Optimization Example

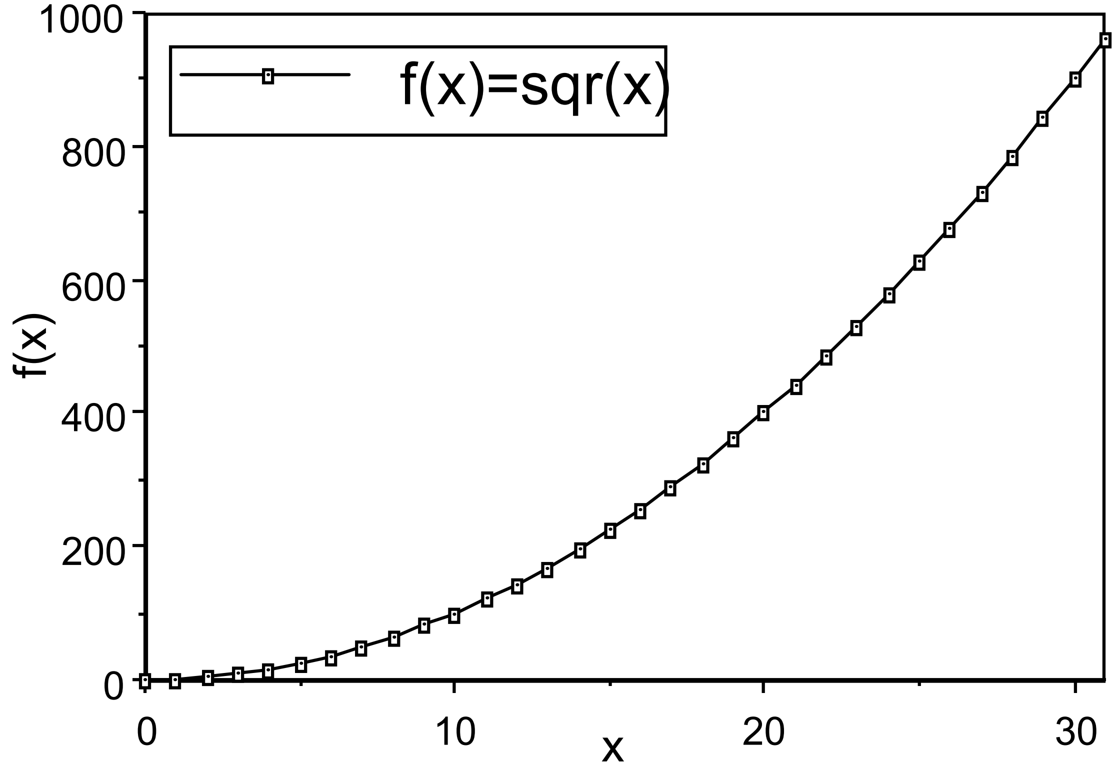
Optimization Problem:

Maximum of

$f(x)=x^2$, with $x \in [0,31]$, where x is an integer.

Problem Representation:

- encoding of the variable x as a binary vector;
- $[0, 31] \longrightarrow [00000, 11111]$



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

1. The first step is encoding of chromosomes; use binary representation for integers; 5-bits are used to represent integers up to 31.
2. Assume that the population size is 4.
3. Generate initial population at random. They are chromosomes or genotypes; e.g., 01101, 11000, 01000, 10011.
4. Calculate fitness value for each individual.

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

   $01101 \rightarrow 13$; $11000 \rightarrow 24$; $01000 \rightarrow 8$; $10011 \rightarrow 19$;

   (b) Evaluate the fitness according to $f(x) = x^2$,

   $13 \rightarrow 169$; $24 \rightarrow 576$; $8 \rightarrow 64$; $19 \rightarrow 361$.

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

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

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

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

n * p_i is expected count

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

The string no 2 has maximum chance of selection.

A Genetic Algorithm by Hand

string No.	initial popu-lation	x value	fitness $f(x)=x^2$	% of total fitness	selection probability
1	01101	13	169	14,4	0,144
2	11000	24	576	49,2	0,492
3	01000	8	64	5,5	0,055
4	10011	19	361	30,9	0,309

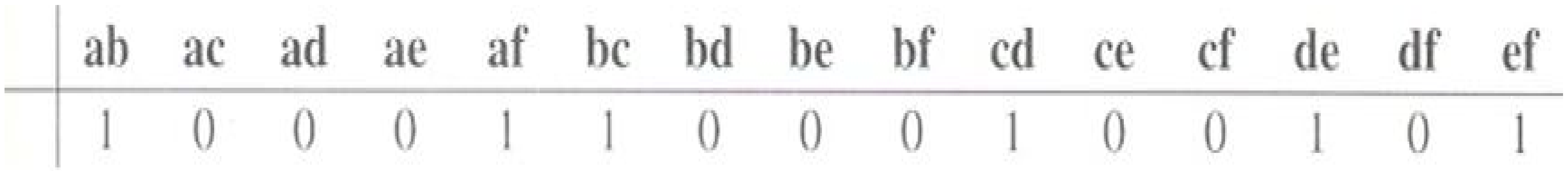
after selection	mate	crossover point	mutation	new population	fitness $f(x)=x^2$
0110 | 1	2	4	-	01100	144
1100 | 0	1	4	3	11101	841
11 | 000	4	2	-	11011	729
10 | 011	2	2	-	10000	256

Example: Traveling Salesperson Problem

- Given (n) cities and distances between them, find the tour of minimum total distance. A tour involves starting from any city, visiting every city exactly once and returning to the starting city.
- Setting up the chromosomes as to use the list of cities in the order that they are to be visited.
- There will be many possible routes that can be taken.
- The fitness measure is a function of the total distance travelled,

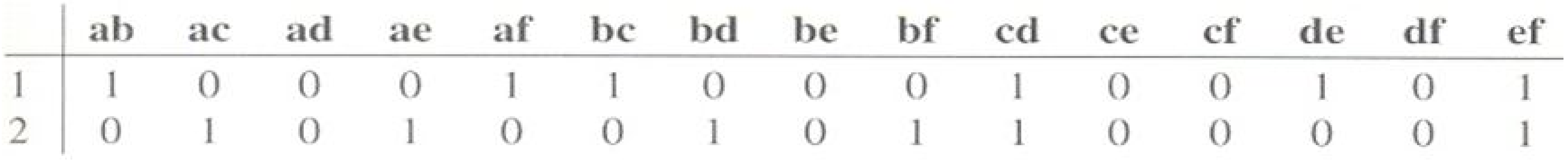
Fitness = 1/ total_distance

- Consider a problem of six cities and a path which can be listed as **abcdef**, this would look like;



	ab	ac	ad	ae	af	bc	bd	be	bf	cd	ce	cf	de	df	ef
	1	0	0	0	1	1	0	0	0	1	0	0	1	0	1

- Recombination is a type of crossover used in such a problem. Now take the following two paths; **abcdefa** and **aefbdca**;



	ab	ac	ad	ae	af	bc	bd	be	bf	cd	ce	cf	de	df	ef
1	1	0	0	0	1	1	0	0	0	1	0	0	1	0	1
2	0	1	0	1	0	0	1	0	1	1	0	0	0	0	1

The method of recombination is as follows:

Step 1: Produce a table with all the cities in one column, and in the second column list all the cities connected to them in either of the parent chromosomes. Using the two chromosomes above, this table is:

City	Connected to:
a	b, f, c, e
b	a, c, d, f
c	b, d, a
d	c, e, b
e	d, f, a
f	a, e, b

Step 2: Randomly select a city as the current city. Let's choose **c** as the current city.

Step 3: Delete the current city from the right-hand side of the table. The table becomes:

CURRENT CITY: c

City	Connected to:
a	b, f, e
b	a, d, f
c	b, d, a
d	e, b
e	d, f, a
f	a, e, b

Step 4: Look at all the cities connected to the current city, and select the one with the fewest connections. If there are two or more cities with the same smallest number of connections, choose one randomly. In this example, **d** has two connections, while **b** and **a** have three. Therefore select **d** as the current city.

Step 5: Repeat Steps 3 and 4 until there are no cities left.

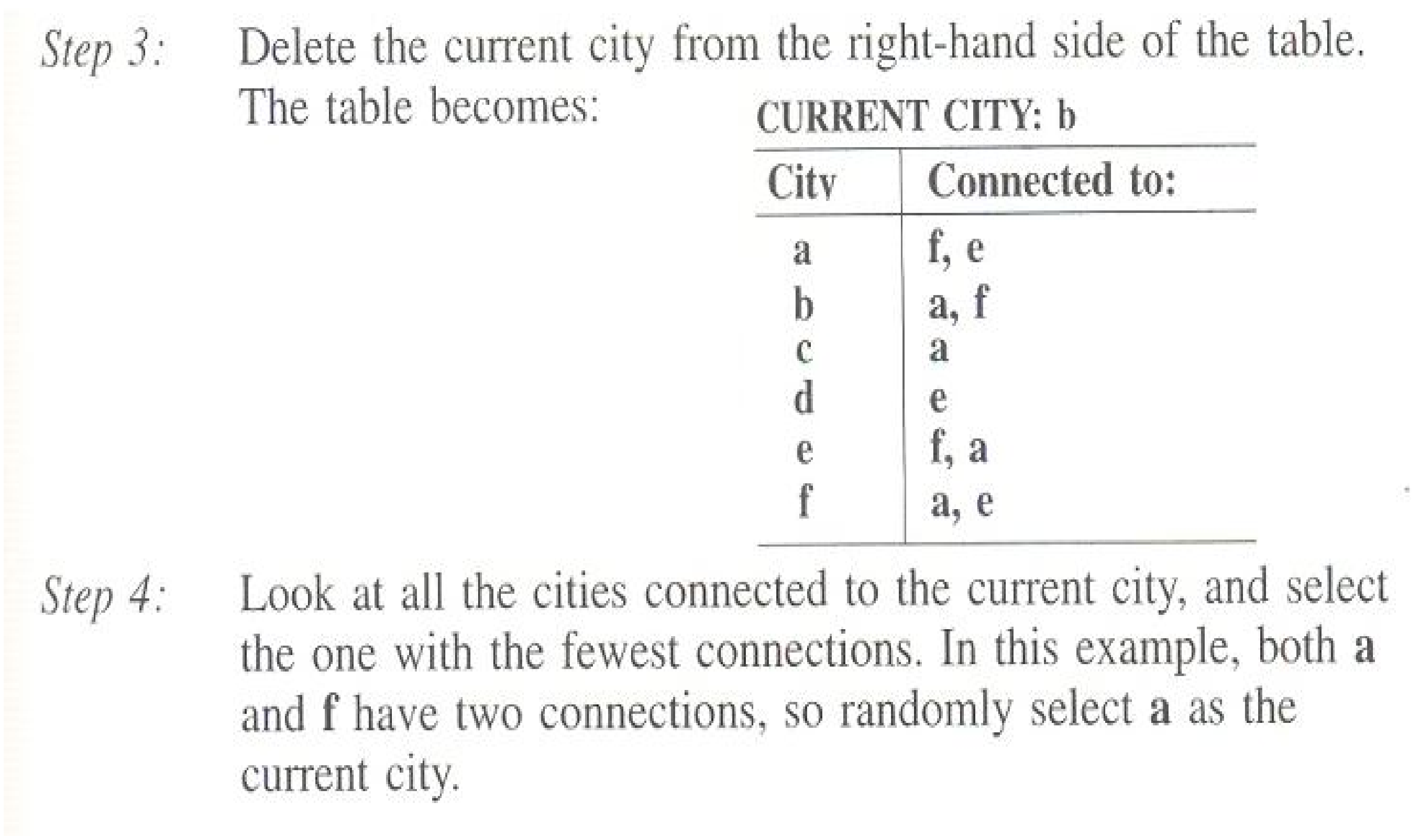
This goes as follows for this example.

Step 3: Delete the current city, **d**, from the right-hand side of the table. The table becomes:

CURRENT CITY: d

City	Connected to:
a	b, f, e
b	a, f
c	b, a
d	e, b
e	f, a
f	a, e, b

Step 4: Look at all the cities connected to the current city, and select the one with the fewest connections. In this example, both **e** and **b** have two connections, so randomly select **b** as the current city.



Step 3: Delete the current city from the right-hand side of the table. The table becomes:

CURRENT CITY: b

City	Connected to:
a	f, e
b	a, f
c	a
d	e
e	f, a
f	a, e

Step 4: Look at all the cities connected to the current city, and select the one with the fewest connections. In this example, both **a** and **f** have two connections, so randomly select **a** as the current city.

Step 3: Delete the current city from the right-hand side of the table. The table becomes:

CURRENT CITY: a

City	Connected to:
a	f, e
b	f
c	
d	e
e	f
f	e

Step 4: Look at all the cities connected to the current city, and select the one with the fewest connections. In this example, both **e** and **f** have one connection, so randomly select **e** as the current city.

Only **f** is left. The resulting path is the list of cities in the order that they were selected by this process of recombination: **cdbaefc**

If we look at this as a chromosome and compare it with the parent chromosomes, you can see that recombination has taken data from each of the parent chromosomes. In the table below, the parent chromosomes are labelled 1 and 2, and the offspring is labelled 3. Bold digits in the parent chromosome indicate the source of the genetic material in the offspring. When the source could be either parent, both are indicated in bold.

	ab	**ac**	**ad**	**ae**	**af**	**bc**	**bd**	**be**	**bf**	**cd**	**ce**	**cf**	**de**	**df**	**ef**
1	**1**	**0**	**0**	0	1	1	0	**0**	**0**	**1**	**0**	0	1	**0**	**1**
2	0	1	**0**	**1**	**0**	**0**	**1**	**0**	1	**1**	**0**	0	**0**	**0**	**1**
3	1	0	0	1	0	0	1	0	0	1	0	**1**	0	0	1

Recombination can also cause mutation. In this example, the offspring has a 1 in the chromosome at **cf** (shown in bold) which was in neither parent. Thus recombination is a powerful method which performs crossover and mutation at the same time.

Benefits of Genetic Algorithms:

The main advantage of GAs is that they offer an evolution-based stochastic search that can be useful in finding good solutions to practical complex optimization problems, especially when gradient information is not conveniently available. Benefits of using GAS;

- Concept is easy to understand
- Modular, separate from application
- Supports multi-objective optimization
- Good for "noisy" environments
- Always an answer; answer gets better with time
- Inherently parallel; easily distributed.
- Many ways to speed up and improve a GA-based application as knowledge about problem domain is gained,
- Easy to exploit previous or alternate solutions.
- Flexible building blocks for hybrid applications
- Substantial history and range of use

Some GA Application Types:

Domain	Application Types
Control	gas pipeline, pole balancing, missile evasion, pursuit
Design	semiconductor layout, aircraft design, keyboard configuration, communication networks
Scheduling	manufacturing, facility scheduling, resource allocation
Robotics	trajectory planning
Machine Learning	designing neural networks, improving classification algorithms, classifier systems
Signal Processing	filter design
Game Playing	poker, checkers, prisoner's dilemma
Combinatorial Optimization	set covering, travelling salesman, routing, bin packing, graph colouring and partitioning

References:

1. S. Rajasekaran & G.A. Vijayalakshmi Pai, "Neural Networks, Fuzzy Logic, and Genetic Algorithms: Synthesis & Applications", ISBN: 81-203-2186-3, Prentice Hall, 2006.
2. A. Zilouchian and M. Jamshidi, "Intelligent Control Systems Using Soft Computing Methodologies", CRC Press, 2001.
3. Z. Michalewicz, "Genetic Algorithms and Data Structures: Evolution Programs", Springer-Verlag, 1999.
4. R. C. Chakraborty, Fundamentals of Genetic Algorithms: AI Course, Lecture 39-40, www.myreaders.info/html/artificial_intelligence.html
5. J. Johnson & P. Picton, “Designing Intelligent Machines, Vol. 2, Concepts in Artificial Intelligence”, Butterworth Heinemann, England, 1995.
6. M. Mitchell, "An Introduction to GA", MIT Press, 1996.
7. A. Engelbrecht, “Computational Intelligence", John Wiley and Sons, 2002.
8. Plus some research papers on the topics.