Fundamentals of Genetic Algorithms

Assist. Prof. Dr. Mohammed Najm Abdullah

https://itswtech.academia.edu/MohammedAlSalam

What are Genetic Algorithms

- Genetic Algorithms (GAs) are adaptive heuristic search algorithm based on the evolutionary ideas of natural selection and genetics.
- Genetic algorithms (GAs) are a part of Evolutionary computing, a rapidly growing area of artificial intelligence. GAs are inspired by Darwin's theory about evolution - "survival of the fittest".
- GAs represent an intelligent exploitation of a random search used to solve optimization problems.
- GAs, although randomized, exploit historical information to direct the search into the region of better performance within the search space.
- In nature, competition among individuals for scanty resources results in the fittest individuals dominating over the weaker ones.

Why Genetic Algorithms?

- It is better than conventional AI ; It is more robust.
- Unlike older AI systems, the GA's do not break easily even if the inputs changed slightly, or in the presence of reasonable noise.
- While performing search in large state-space, or multi-modal state-space, or n-dimensional surface, a genetic algorithms offer significant benefits over many other typical search optimization techniques like - linear programming, heuristic, depth-first, breath-first.

"Genetic Algorithms are good at taking large, potentially huge search spaces and navigating them, looking for optimal combinations of things, the solutions one might not otherwise find in a lifetime."

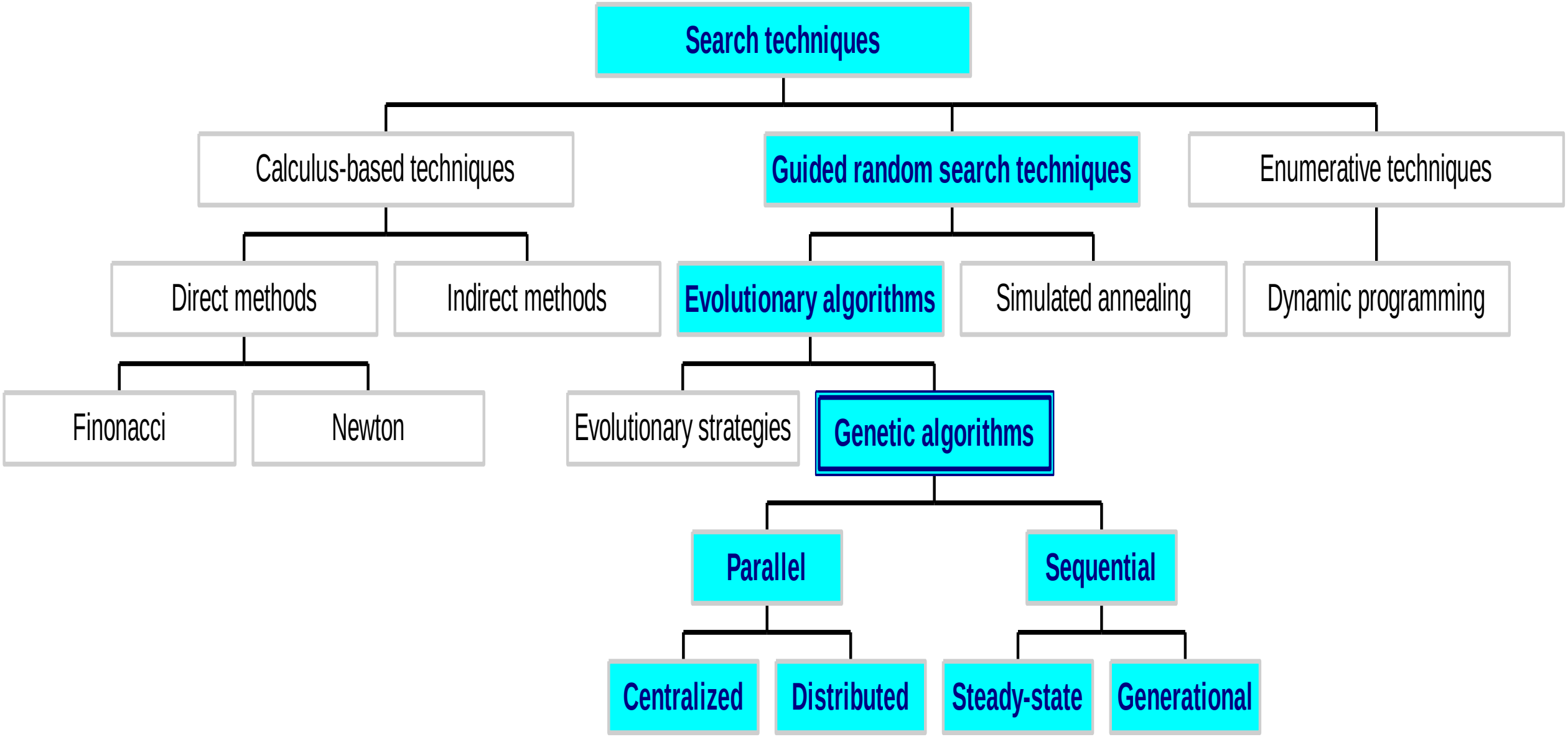
Genetic Algorithms Background

- Directed search algorithms based on the mechanics of biological evolution
- Developed by **John Holland**, University of Michigan (1970's)
  - To understand the adaptive processes of natural systems
  - To design artificial systems software that retains the robustness of natural systems
- Provide efficient, effective techniques for optimization and machine learning applications
- Widely-used today in business, scientific and engineering circles

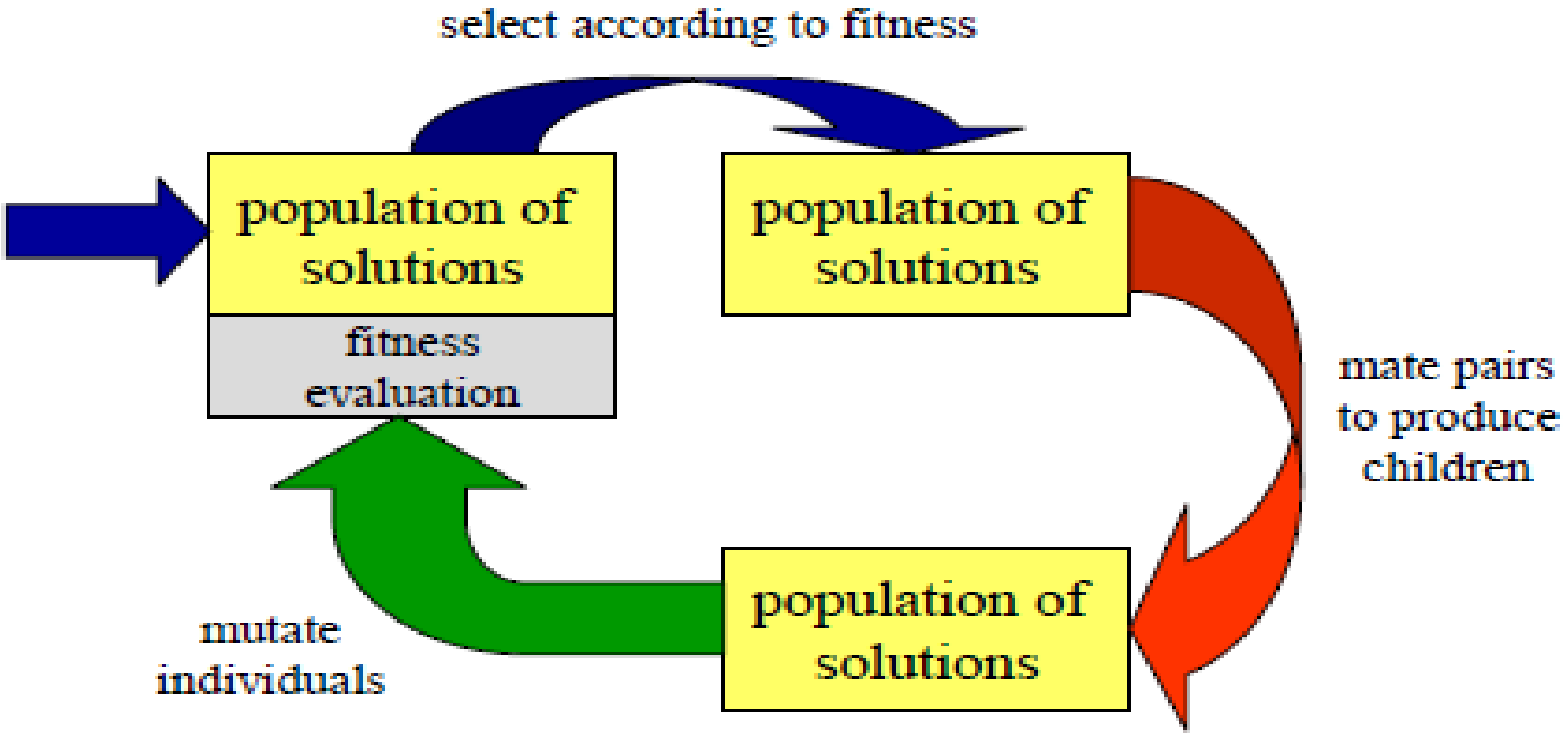
Some GA Application Types

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

Classes of Search Techniques



General idea of Genetic Algorithms



Process broadly same as genetic algorithms.
But significant change in representation.

Components of a GA

A problem to solve, and ...

- **Encoding technique** (***gene, chromosome***)
- **Initialization procedure** ***(creation)***
- **Evaluation function** ***(environment)***
- **Selection of parents** ***(reproduction)***
- **Genetic operators** ***(mutation, recombination)***
- **Parameter settings** ***(practice and art)***

Simple Genetic Algorithm

```
{
    initialize population;
    evaluate population;
    while Termination Criteria Not Satisfied
    {
        select parents for reproduction;
        perform recombination and mutation;
        evaluate population;
    }
}
```

The GA Cycle of Reproduction

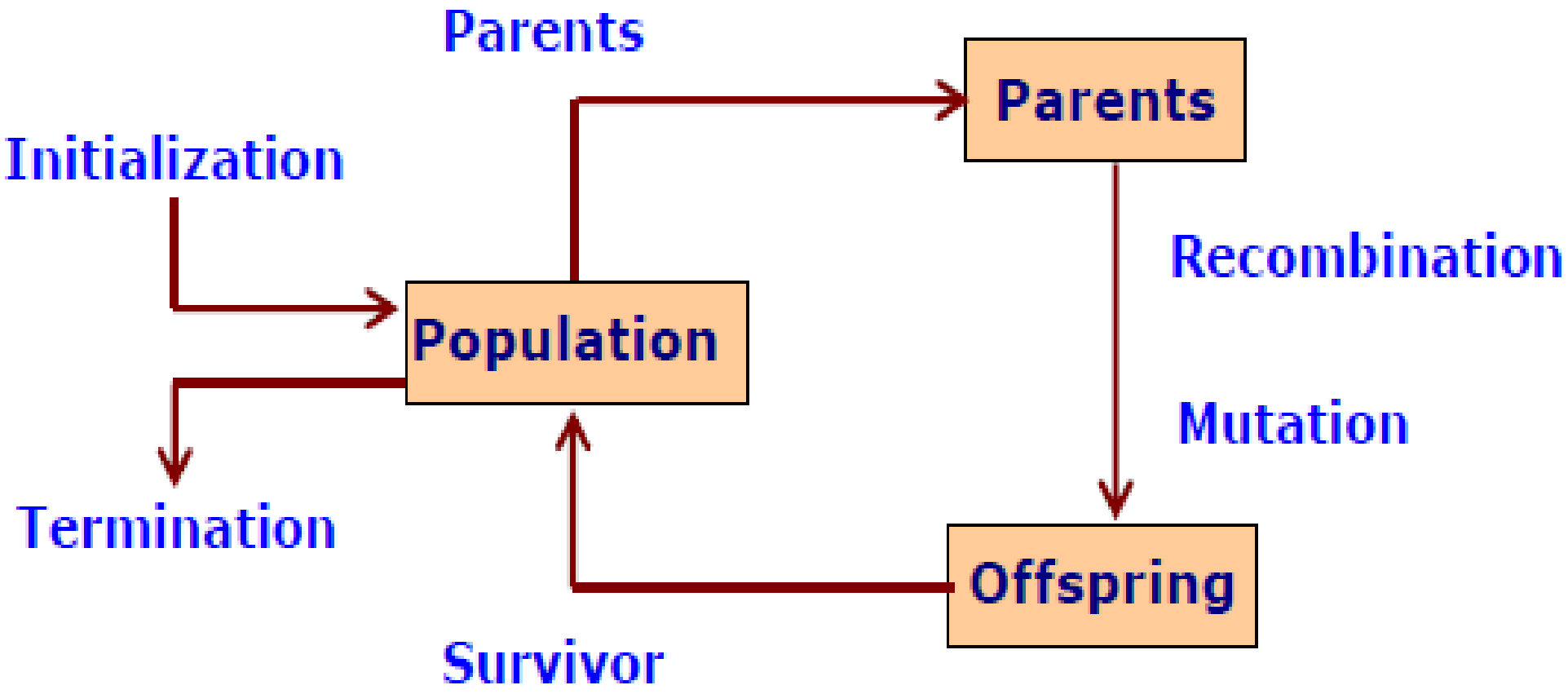


Fig. General Scheme of Evolutionary process

Pseudo-Code

BEGIN

INITIALISE population with random candidate solution.

EVALUATE each candidate;

REPEAT UNTIL (termination condition) is satisfied DO

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

END.

Working Principles

Before getting into GAs, it is necessary to explain few terms.

- Chromosome : a set of genes; a chromosome contains the solution in form of genes.
- Gene : a part of chromosome; a gene contains a part of solution. It determines the solution. e.g. 16743 is a chromosome and 1, 6, 7, 4 and 3 are its genes.
- Individual : same as chromosome.
- Population: number of individuals present with same length of chromosome.
- Fitness : the value assigned to an individual based on how far or close a individual is from the solution; greater the fitness value better the solution it contains.
- Fitness function : a function that assigns fitness value to the individual. It is problem specific.
- Breeding : taking two fit individuals and then intermingling there chromosome to create new two individuals.
- Mutation : changing a random gene in an individual.
- Selection : selecting individuals for creating the next generation.

Outline of the Basic Genetic Algorithm

1. [Start] Generate random population of **n** chromosomes (i.e. suitable solutions for the problem).
2. [Fitness] Evaluate the fitness **f(x)** of each chromosome **x** in the population.
3. [New population] Create a new population by repeating following steps until the new population is complete.
   (a) [Selection] Select two parent chromosomes from a population according to their fitness (better the fitness, bigger the chance to be selected)
   (b) [Crossover] With a crossover probability, cross over the parents to form new offspring (children). If no crossover was performed, offspring is the exact copy of parents.
   (c) [Mutation] With a mutation probability, mutate new offspring at each locus (position in chromosome).
   (d) [Accepting] Place new offspring in the new population
4. [Replace] Use new generated population for a further run of the algorithm
5. [Test] If the end condition is satisfied, stop, and return the best solution in current population
6. [Loop] Go to step 2

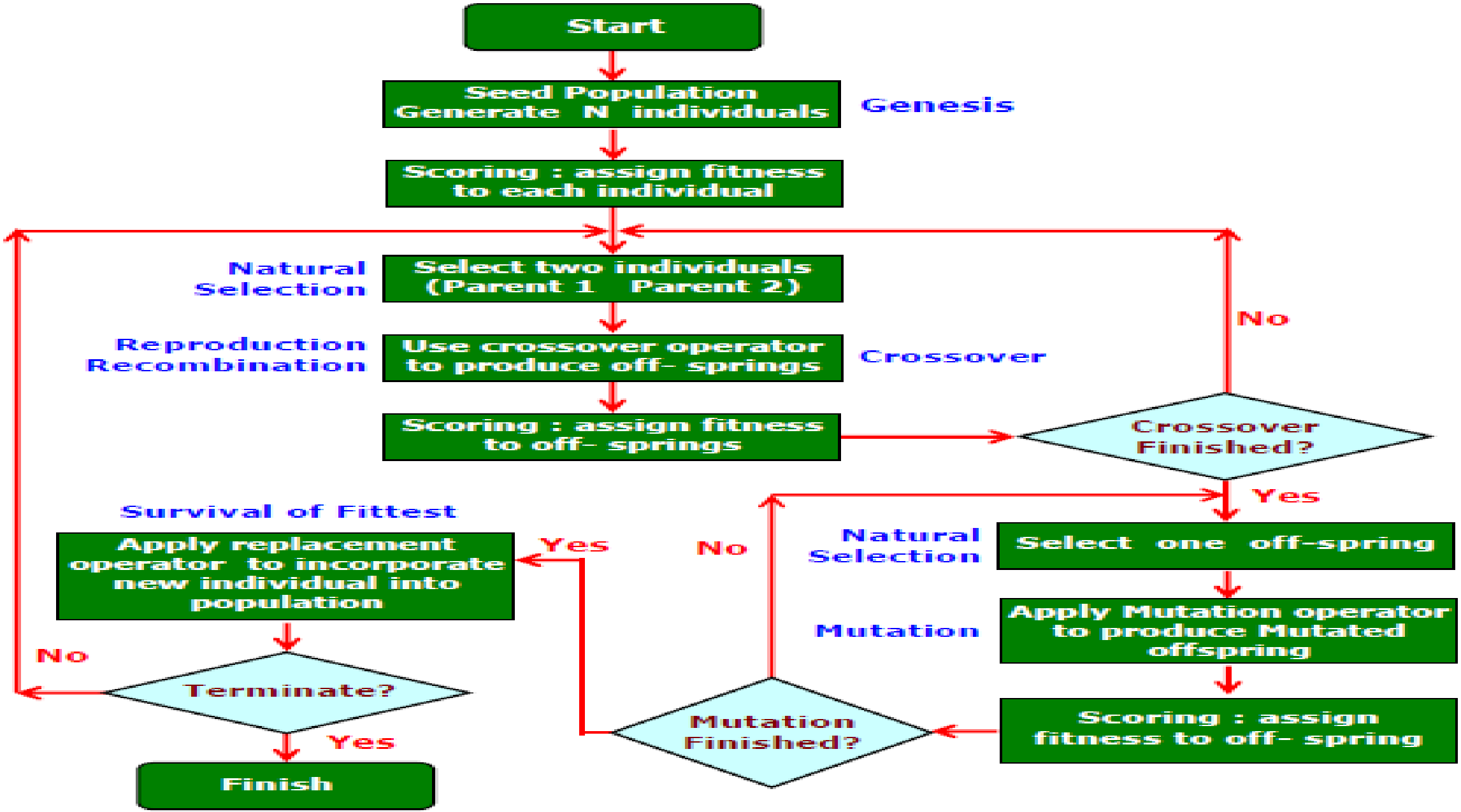


Fig. Genetic Algorithm – program flow chart

Encoding

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

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

  Example :

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

  A chromosome is an array of genes. In binary form

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

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

  (11000010, 00001110, 001111010, 10100011)

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

  Chromosome 1 : 1101100100110110

  Chromosome 2 : 1101111000011110

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

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

Binary Encoding

Binary encoding is the most common to represent information contained. In genetic algorithms, it was first used because of its relative simplicity.

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

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

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

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

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

Example 1:

One variable function, say 0 to 15 numbers, numeric values, represented by 4 bit binary string.

Numeric value	4-bit string	Numeric value	4-bit string	Numeric value	4-bit string
0	0 0 0 0	6	0 1 1 0	12	1 1 0 0
1	0 0 0 1	7	0 1 1 1	13	1 1 0 1
2	0 0 1 0	8	1 0 0 0	14	1 1 1 0
3	0 0 1 1	9	1 0 0 1	15	1 1 1 1
4	0 1 0 0	10	1 0 1 0		
5	0 1 0 1	11	1 0 1 1		

Value Encoding

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

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

   real numbers, characters or objects.

   Examples :

   Chromosome A 1.2324 5.3243 0.4556 2.3293 2.4545

   Chromosome B ABDJEIFJDHDIERJFDLDFLFEGT

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

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

Permutation Encoding

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

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

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

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

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

Examples :

1. The Traveling Salesman problem:

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

2. The Eight Queens problem :

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

Tree Encoding

Tree encoding is used mainly for evolving programs or expressions.

For genetic programming :

- In tree encoding, every chromosome is a tree of some objects, such as functions or commands in programming language.
- Tree encoding is useful for evolving programs or any other structures that can be encoded in trees.
- The crossover and mutation can be done relatively easy way .

Example :

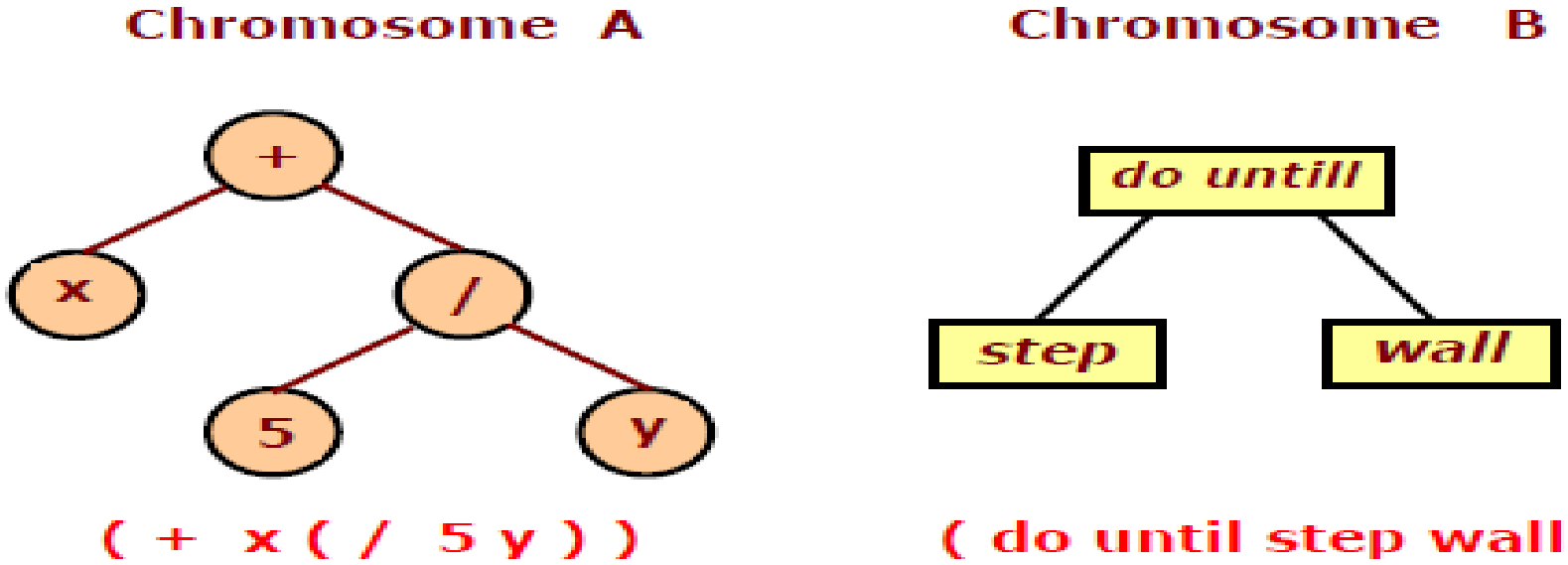


Fig. Example of Chromosomes with tree encoding

Operators of Genetic Algorithm

Genetic operators used in genetic algorithms maintain genetic diversity. Genetic diversity or variation is a necessity for the process of evolution.

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

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

In addition to these operators, there are some parameters of GA.

One important parameter is **Population size**.

- Population size says how many chromosomes are in population (in one generation).
- If there are only few chromosomes, then GA would have a few possibilities to perform crossover and only a small part of search space is explored.
- If there are many chromosomes, then GA slows down.
- Research shows that after some limit, it is not useful to increase population size, because it does not help in solving the problem faster. The population size depends on the type of encoding and the problem.

Reproduction, or Selection

Reproduction is usually the first operator applied on population. From the population, the chromosomes are selected to be parents to crossover and produce offspring.

The problem is how to select these chromosomes ?

According to Darwin's evolution theory "survival of the fittest" – the best ones should survive and create new offspring.

- The Reproduction operators are also called Selection operators.
- Selection means extract a subset of genes from an existing population, according to any definition of quality. Every gene has a meaning, so one can derive from the gene a kind of quality measurement called **fitness function**. Following this quality (fitness value), selection can be performed.

- Fitness function quantifies the optimality of a solution (chromosome) so that a particular solution may be ranked against all the other solutions. The function depicts the closeness of a given 'solution' to the desired result.

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

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

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

Example of Selection

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

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.

Roulette wheel selection (Fitness-Proportionate Selection)

Roulette-wheel selection, also known as Fitness Proportionate Selection, is a genetic operator, used for selecting potentially useful solutions for recombination.

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.

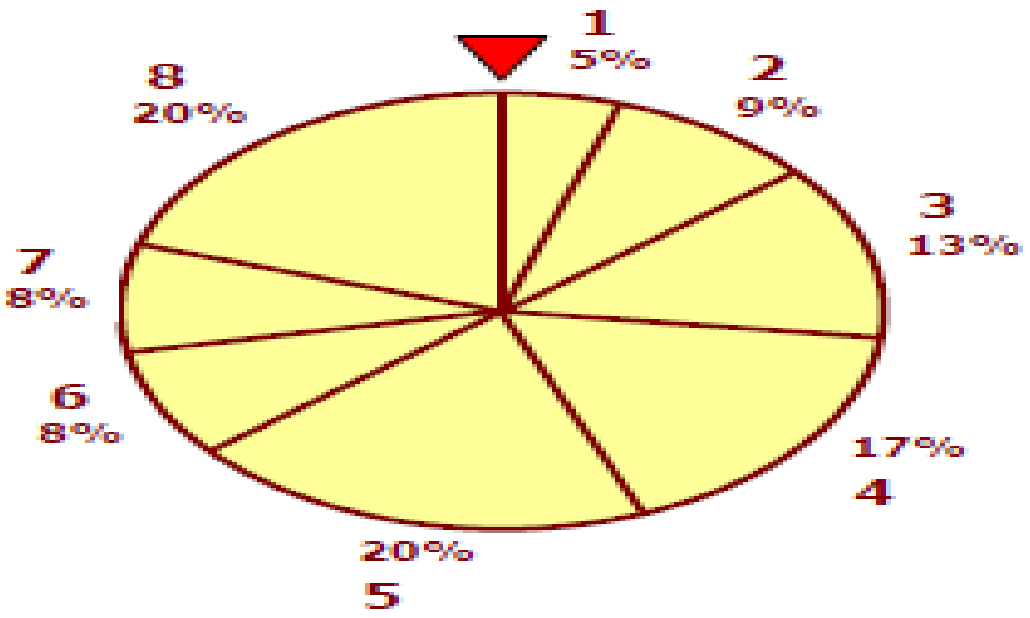


Fig. Roulette-wheel Shows 8 individual with fitness

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

- the 5^{th} individual has a higher fitness than others, so the wheel would choose the 5^{th} 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 = $\bar{F} = \sum F_j / n$; **Expected count = (n =8) x pi**

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

Crossover

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

The Crossover operators are of many types.

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

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

One-Point Crossover

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

Consider the two parents selected for crossover.

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

Interchanging the parents chromosomes after the crossover points -

The Offspring produced are :

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

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

Two-Point Crossover

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

Consider the two parents selected for crossover :

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

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

Offspring 1	**1 1 0 1 1 | 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

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

Consider the two parents selected for crossover.

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

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

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

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

Arithmetic

Arithmetic crossover operator linearly combines two parent chromosome vectors to produce two new offspring according to the equations:

$$\text{Offspring1} = a * \text{Parent1} + (1 - a) * \text{Parent2}$$

$$\text{Offspring2} = (1 - a) * \text{Parent1} + a * \text{Parent2}$$

where **a** is a random weighting factor chosen before each crossover operation.

Consider two parents (each of 4 float genes) selected for crossover:

Parent 1	**(0.3)**	**(1.4)**	**(0.2)**	**(7.4)**
Parent 2	**(0.5)**	**(4.5)**	**(0.1)**	**(5.6)**

Applying the above two equations and assuming the weighting factor **a = 0.7**, applying above equations, we get two resulting offspring. The possible set of offspring after arithmetic crossover would be:

Offspring 1	**(0.36)**	**(2.33)**	**(0.17)**	**(6.87)**
Offspring 2	**(0.402)**	**(2.981)**	**(0.149)**	**(5.842)**

Heuristic

Heuristic crossover operator uses the fitness values of the two parent chromosomes to determine the direction of the search.

The offspring are created according to the equations:

$$Offspring1 = BestParent + r * (BestParent - WorstParent)$$

$$Offspring2 = BestParent$$

where **r** is a random number between **0** and **1**.

It is possible that **offspring1** will not be feasible. It can happen if **r** is chosen such that one or more of its genes fall outside of the allowable upper or lower bounds. For this reason, heuristic crossover has a user defined parameter **n** for the number of times to try and find an **r** that results in a feasible chromosome. If a feasible chromosome is not produced after **n** tries, the worst parent is returned as offspring1.

Mutation

After a crossover is performed, mutation takes place.
Mutation is a genetic operator used to maintain genetic diversity from one generation of a population of chromosomes to the next.

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

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

Mutation is an important part of the genetic search, helps to prevent the population from stagnating at any local optima. Mutation is intended to prevent the search falling into a local optimum of the state space.

The Mutation operators are of many type.

- one simple way is, **Flip Bit**.
- the others are **Boundary, Non-Uniform, Uniform, and Gaussian**.

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

Flip Bit

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

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

This mutation operator can only be used for binary genes.

Consider the two original off-springs selected for mutation.

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

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

The Mutated Off-spring produced are :

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

Example 1 :

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

1. Devise a means to represent a solution to the problem :
   Assume, we represent **x** with five-digit unsigned binary integers.
2. Devise a heuristic for evaluating the fitness of any particular solution :
   The function **f(x)** is simple, so it is easy to use the **f(x)** value itself to rate the fitness of a solution; else we might have considered a more simpler heuristic that would more or less serve the same purpose.
3. Coding - Binary and the String length :
   GAs often process binary representations of solutions. This works well, because crossover and mutation can be clearly defined for binary solutions. A Binary string of length **5** can represents 32 numbers (0 to 31).
4. Randomly generate a set of solutions :
   Here, considered a population of four solutions. However, larger populations are used in real applications to explore a larger part of the search. Assume, four randomly generated solutions as : **01101, 11000, 01000, 10011.** These are chromosomes or genotypes.

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

The calculated fitness values for each individual are -

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

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

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

13 → 169; 24 → 576; 8 → 64; 19 → 361.

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

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

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

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

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

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

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

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

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

7. Randomly pair the members of the new generation

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

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

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

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

0 1 1 0 1 ⇒ 0 1 1 0 |1 ⇒ 0 1 1 0 0

1 1 0 0 0 ⇒ 1 1 0 0 |0 ⇒ 1 1 0 0 1

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

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

1 1 0 0 0 ⇒ 1 1 |0 0 0 ⇒ 1 1 0 1 1

1 0 0 1 1 ⇒ 1 0 |0 1 1 ⇒ 1 0 0 0 0

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

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

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

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

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

Observe that :

1. Initial populations : At start step 5 were

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

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

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

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

GA: Disadvantages

1. No guarantee for optimal solution within a finite time
2. Weak theoretical basis
3. Interdependency of genes
4. Parameter tuning is an issue
5. Often computationally expensive, i.e. *slow*

GA: Advantages

1. A robust search technique
2. No (little) knowledge (assumption) the problem space
3. Fairly simple to develop: low development costs
4. Easy to incorporate with other methods
5. Solutions are interpretable
6. Can be run interactively, i.e. accommodate user preference
7. Provide many alternative solutions
8. Acceptable performance at acceptable costs on a wide range of problems
9. Intrinsic parallelism (robustness, fault tolerance)

Example

Find the value of (x,y,z) in which the following function is maximum using genetic algorithm approach (only determine G_1 and G_2)

$$F(x,y) = (x-5)^2 + (y-3)^2 + (z-1)^2$$ **, Assume:**

- **The selected population consists of the following chromosomes**

  $P_0 = \{(5, 3, 1), (3, 10,1), (11,5,7), (6,3,11), (1,7,13)\}$;

  $0 \le x \le 15$ and $0 \le y \le 15$

- **The fitness ratio=** $|F(x_i,y_i)| / \sum| F(x_i,y_i)|$
- **Two–point crossover at Bit 3 and Bit 9 is used; and mutation is happened in the Bit 6.**