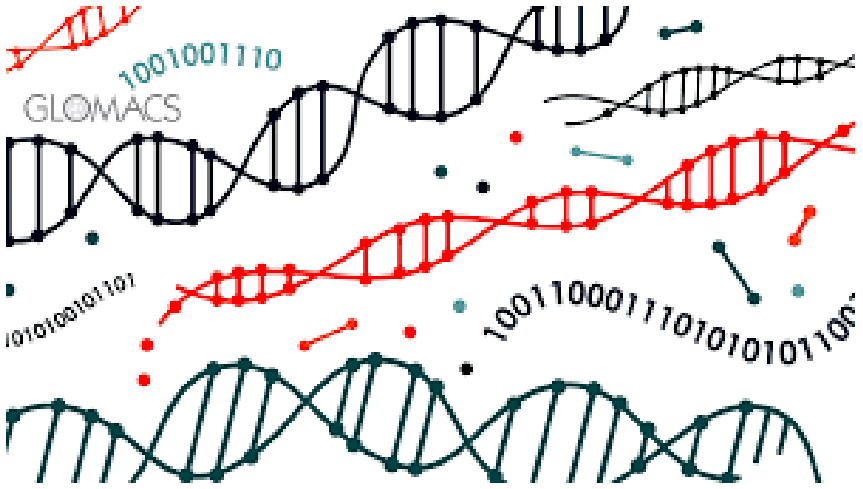
GENETIC ALGORITHMS (PART 1)



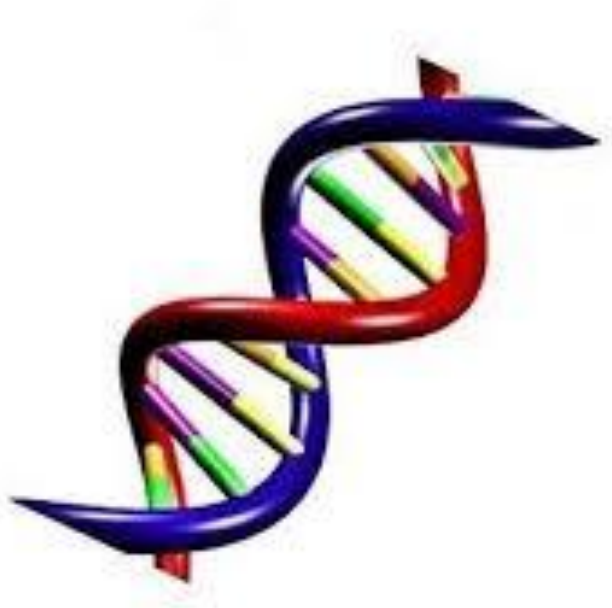
Dr. Krishnendu Guha
Assistant Professor (On Contract)
National Institute of Technology (NIT), Jamshedpur
Email: krishnendu.ca@nitjsr.ac.in

OVERVIEW OF GA

- Genetic Algorithm (GA) is a search-based optimization technique based on the principles of **Genetics and Natural Selection**.

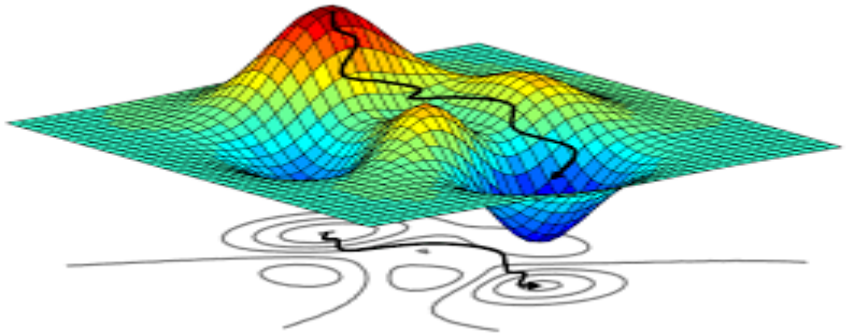
- It is frequently used to find optimal or near-optimal solutions to difficult problems which otherwise would take a lifetime to solve.

- It is frequently used to solve optimization problems, in research, and in machine learning.



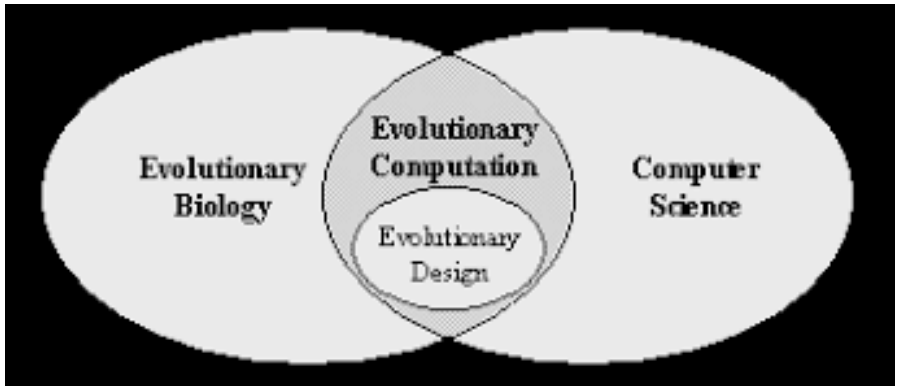
OPTIMIZATION

- Optimization is the process of **making something better**.
- In any process, we have a set of inputs and a set of outputs
- Optimization refers to finding the values of inputs in such a way that we get the “best” output values.
- The definition of “best” varies from problem to problem, but in mathematical terms, it refers to maximizing or minimizing one or more objective functions, by varying the input parameters.
- The set of all possible solutions or values which the inputs can take make up the search space.
- In this search space, lies a point or a set of points which gives the optimal solution.
- The aim of optimization is to find that point or set of points in the search space



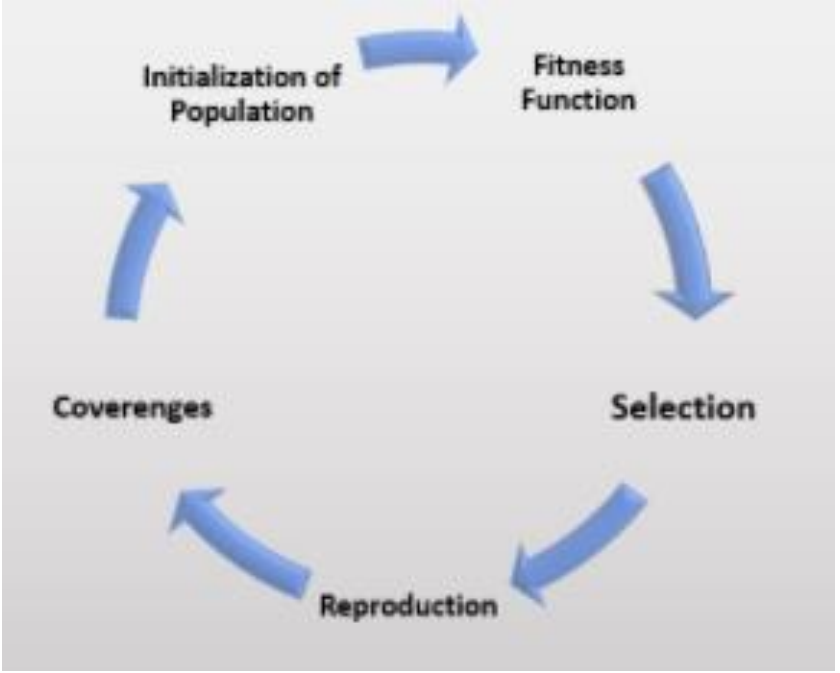
WHAT ARE GENETIC ALGORITHMS?

- Nature has always been a great source of inspiration to all mankind.
- Genetic Algorithms (GAs) are search based algorithms based on the concepts of natural selection and genetics.
- GAs are a subset of a much larger branch of computation known as **Evolutionary Computation**.
- GAs were developed by John Holland and his students and colleagues at the University of Michigan, most notably David E. Goldberg and has since been tried on various optimization problems with a high degree of success.



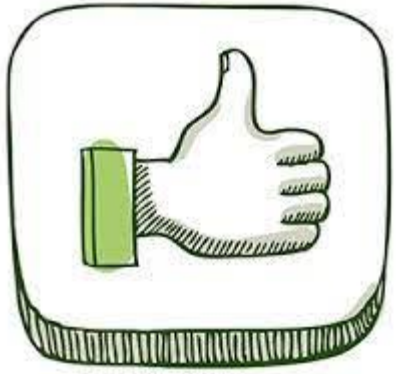
GA BASIC CONCEPT

- In GAs, we have a **pool or a population of possible solutions** to the given problem.
- These solutions then undergo recombination and mutation (like in natural genetics), producing new children, and the process is repeated over various generations.
- Each individual (or candidate solution) is assigned a fitness value (based on its objective function value) and the fitter individuals are given a higher chance to mate and yield more “fitter” individuals.
- This is in line with the Darwinian Theory of “Survival of the Fittest”.
- In this way we keep “evolving” better individuals or solutions over generations, till we reach a stopping criterion.
- Genetic Algorithms are sufficiently randomized in nature, but they perform much better than random local search (in which we just try various random solutions, keeping track of the best so far), as they exploit historical information as well.



ADVANTAGES OF GA

- Does not require any derivative information (which may not be available for many real-world problems).
- Is faster and more efficient as compared to the traditional methods.
- Has very good parallel capabilities.
- Optimizes both continuous and discrete functions and also multi-objective problems.
- Provides a list of “good” solutions and not just a single solution.
- Always gets an answer to the problem, which gets better over the time.
- Useful when the search space is very large and there are a large number of parameters involved.

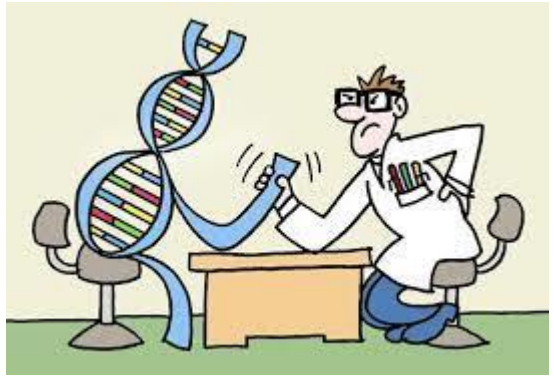


LIMITATIONS OF GA

- GAs are not suited for all problems, especially problems which are simple and for which derivative information is available.
- Fitness value is calculated repeatedly which might be computationally expensive for some problems.
- Being stochastic, there are no guarantees on the optimality or the quality of the solution.
- If not implemented properly, the GA may not converge to the optimal solution.

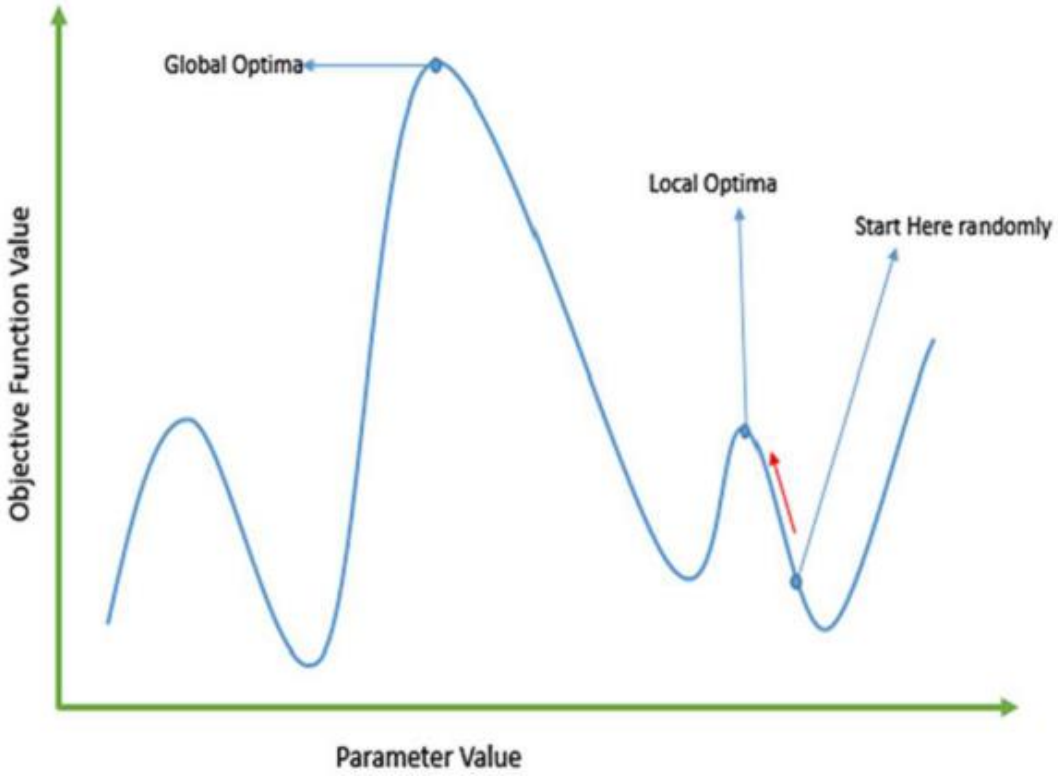
GA – MOTIVATION

- Genetic Algorithms have the ability to deliver a “good-enough” solution “fast-enough”.
- This makes genetic algorithms attractive for use in solving optimization problems. The reasons why GAs are needed are as follows −
- Solving Difficult Problems
- In computer science, there is a large set of problems, which are **NP-Hard**. What this essentially means is that, even the most powerful computing systems take a very long time (even years!) to solve that problem. In such a scenario, GAs prove to be an efficient tool to provide **usable near-optimal solutions** in a short amount of time.



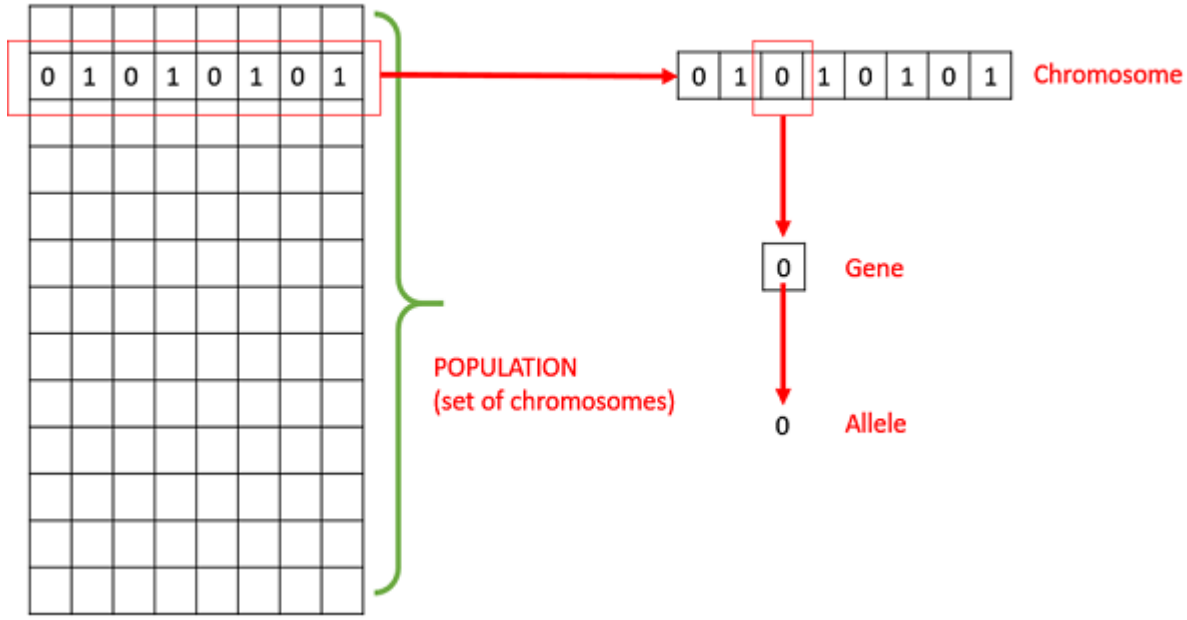
- Getting a Good Solution Fast
- Some difficult problems like the Travelling Salesperson Problem (TSP), have real-world applications like path finding and VLSI Design.
- Now imagine that you are using your GPS Navigation system, and it takes a few minutes (or even a few hours) to compute the “optimal” path from the source to destination.
- Delay in such real world applications is not acceptable and therefore a “good-enough” solution, which is delivered “fast” is what is required.

- Failure of Gradient Based Methods
- Traditional calculus based methods work by starting at a random point and by moving in the direction of the gradient, till we reach the top of the hill.
- This technique is efficient and works very well for single-peaked objective functions like the cost function in linear regression.
- But, in most real-world situations, we have a very complex problem called as landscapes, which are made of many peaks and many valleys, which causes such methods to fail, as they suffer from an inherent tendency of getting stuck at the local optima as shown in the following figure.



BASIC TERMINOLOGY

- **Population** − It is a subset of all the possible (encoded) solutions to the given problem.
  - The population for a GA is analogous to the population for human beings except that instead of human beings, we have Candidate Solutions representing human beings.
- **Chromosomes** − A chromosome is one such solution to the given problem.
- **Gene** − A gene is one element position of a chromosome.
- **Allele** − It is the value a gene takes for a particular chromosome.

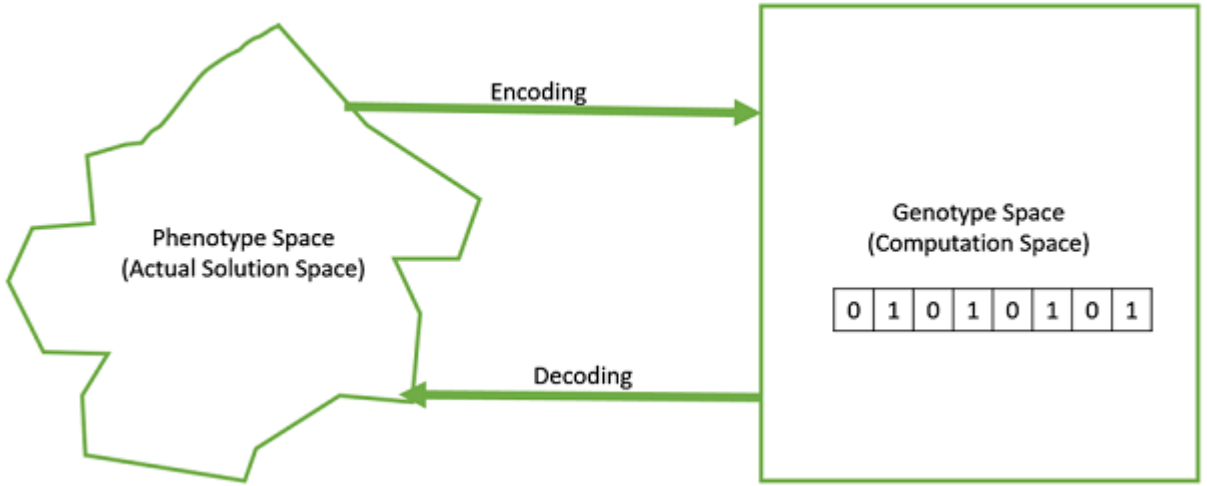


- **Genotype** − Genotype is the population in the computation space. In the computation space, the solutions are represented in a way which can be easily understood and manipulated using a computing system.
- **Phenotype** − Phenotype is the population in the actual real world solution space in which solutions are represented in a way they are represented in real world situations.
- **Decoding and Encoding** − For simple problems, the **phenotype and genotype** spaces are the same. However, in most of the cases, the phenotype and genotype spaces are different. Decoding is a process of transforming a solution from the genotype to the phenotype space, while encoding is a process of transforming from the phenotype to genotype space. Decoding should be fast as it is carried out repeatedly in a GA during the fitness value calculation.
- For example, consider the 0/1 Knapsack Problem. The Phenotype space consists of solutions which just contain the item numbers of the items to be picked.

- However, in the genotype space it can be represented as a binary string of length n (where n is the number of items). A **0 at position x** represents that **x^{th}** item is picked while a 1 represents the reverse. This is a case where genotype and phenotype spaces are different.

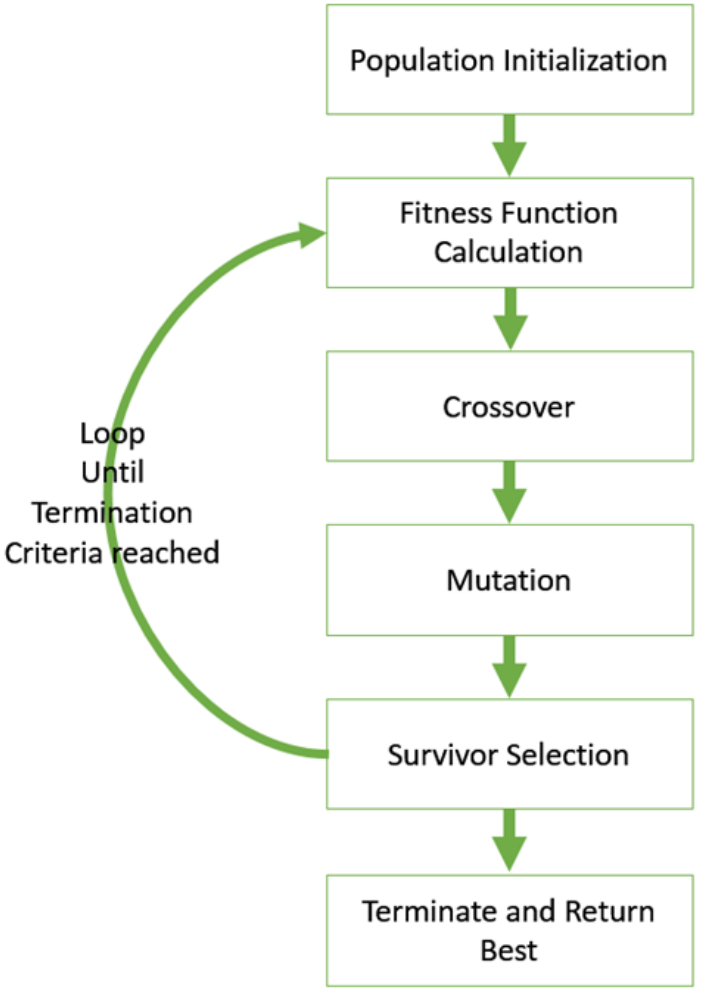
- **Fitness Function** − A fitness function simply defined is a function which takes the solution as input and produces the suitability of the solution as the output. In some cases, the fitness function and the objective function may be the same, while in others it might be different based on the problem.

- **Genetic Operators** − These alter the genetic composition of the offspring. These include crossover, mutation. selection. etc.



BASIC STRUCTURE OF GA

- We start with an initial population
  - (which may be generated at random or seeded by other heuristics),
- select parents from this population for mating.
- Apply crossover and mutation operators on the parents to generate new off-springs.
- And finally these off-springs replace the existing individuals in the population and the process repeats.
- In this way genetic algorithms actually try to mimic the human evolution to some extent.



PSEUDO-CODE FOR GA

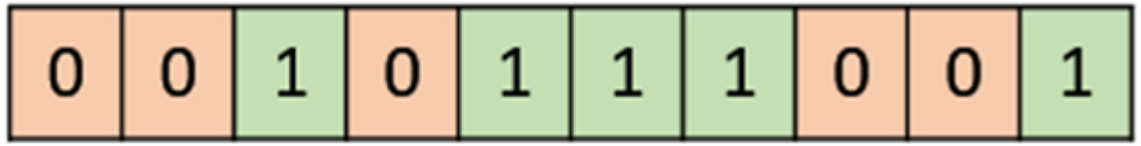
```
GA()
   initialize population
   find fitness of population

   while (termination criteria is reached) do
      parent selection
      crossover with probability pc
      mutation with probability pm
      decode and fitness calculation
      survivor selection
      find best
   return best
```

GENOTYPE REPRESENTATION

- One of the most important decisions to make while implementing a genetic algorithm is deciding the representation that we will use to represent our solutions.
- It has been observed that improper representation can lead to poor performance of the GA.
- Therefore, choosing a proper representation, having a proper definition of the mappings between the phenotype and genotype spaces is essential for the success of a GA.
- In this section, we present some of the most commonly used representations for genetic algorithms.
- However, representation is highly problem specific and the reader might find that another representation or a mix of the representations mentioned here might suit his/her problem better.

- **Binary Representation**
- This is one of the simplest and most widely used representation in GAs. In this type of representation the genotype consists of bit strings.
- For some problems when the solution space consists of Boolean decision variables – yes or no, the binary representation is natural. Take for example the 0/1 Knapsack Problem. If there are n items, we can represent a solution by a binary string of n elements, where the x^{th} element tells whether the item x is picked (1) or not (0).
- For other problems, specifically those dealing with numbers, we can represent the numbers with their binary representation. The problem with this kind of encoding is that different bits have different significance and therefore mutation and crossover operators can have undesired consequences. This can be resolved to some extent by using **Gray Coding,** as a change in one bit does not have a massive effect on the solution.



- **Real Valued Representation**
- For problems where we want to define the genes using continuous rather than discrete variables, the real valued representation is the most natural. The precision of these real valued or floating point numbers is however limited to the computer.

0.5	0.2	0.6	0.8	0.7	0.4	0.3	0.2	0.1	0.9

- **Integer Representation**
- For discrete valued genes, we cannot always limit the solution space to binary ‘yes’ or ‘no’. For example, if we want to encode the four distances – North, South, East and West, we can encode them as **{0,1,2,3}**. In such cases, integer representation is desirable.

1	2	3	4	3	2	4	1	2	1

- **Permutation Representation**
- In many problems, the solution is represented by an order of elements. In such cases permutation representation is the most suited.
- A classic example of this representation is the travelling salesman problem (TSP). In this the salesman has to take a tour of all the cities, visiting each city exactly once and come back to the starting city. The total distance of the tour has to be minimized. The solution to this TSP is naturally an ordering or permutation of all the cities and therefore using a permutation representation makes sense for this problem.

1	5	9	8	7	4	2	3	6	0


ngiyabonga
شكراً جزيلاً
danke
謝謝
teşekkür ederim
спасибо
dank je
thank you
gracias
tapadh leat
mochchakkeram
dziękuję
go raibh maith agat
obrigado
sukriya
kop khun krap
arigatō
dakujem
grazie
merci
мерси
감사합니다