HINDUSTAN
INSTITUTE OF TECHNOLOGY & SCIENCE
(DEEMED TO BE UNIVERSITY)

EAL51003 – ARTIFICIAL INTELLIGENCE
B.Tech[AIML] – III Semester

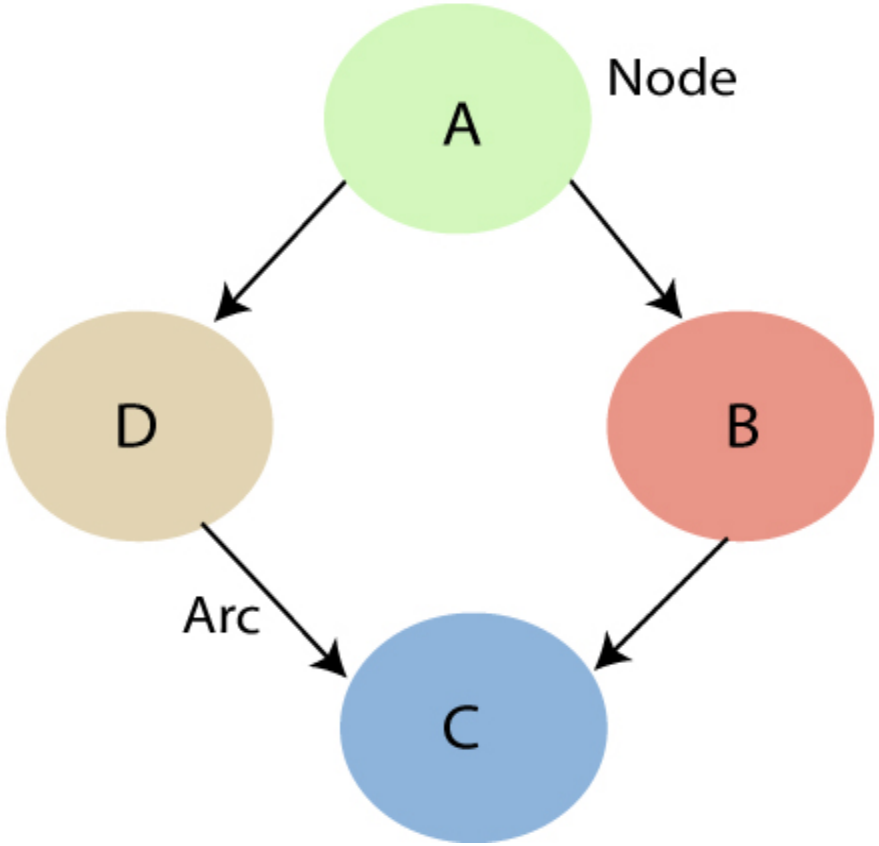
K.Kowsalya
Assistant Professor (SS)
School of Computing Sciences,
Department of Computer Science and Engineering

UNIT-II
BAYESIAN NETWORKS

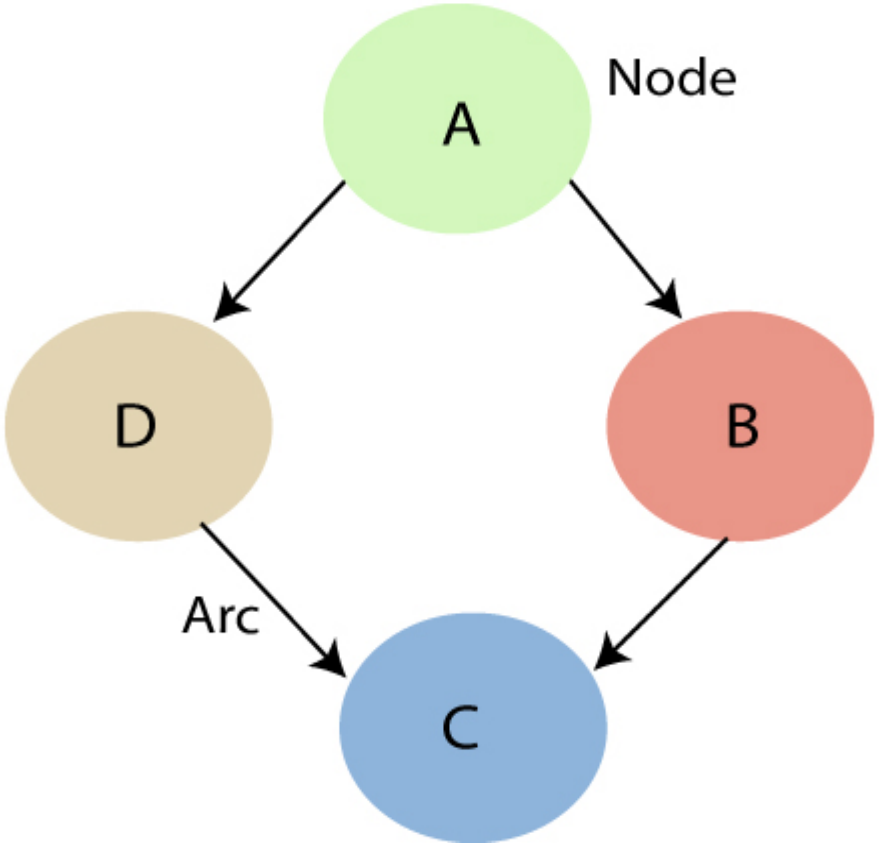
- **"A Bayesian network is a probabilistic graphical model which represents a set of variables and their conditional dependencies using a directed acyclic graph."**
- It is also called a **Bayes network, belief network, decision network**, or **Bayesian model**.
- Bayesian networks are probabilistic, because these networks are built from a **probability distribution**, and also use probability theory for prediction and anomaly detection.

- Real world applications are probabilistic in nature, and to represent the relationship between multiple events, we need a Bayesian network.
- It can also be used in various tasks including **prediction, anomaly detection, diagnostics, automated insight, reasoning, time series prediction**, and **decision making under uncertainty**.
- Bayesian Network can be used for building models from data and experts opinions, and it consists of two parts:
- **Directed Acyclic Graph**
- **Table of conditional probabilities.**

- The generalized form of Bayesian network that represents and solve decision problems under uncertain knowledge is known as an **Influence diagram**.
- **A Bayesian network graph is made up of nodes and Arcs (directed links), where:**



- Each **node** corresponds to the random variables, and a variable can be **continuous** or **discrete**.
- **Arc or directed arrows represent the causal relationship or conditional probabilities between random variables.**
- These directed links or arrows connect the pair of nodes in the graph. These links represent that one node directly influence the other node, and if there is no directed link that means that nodes are independent with each other.



- **In the diagram, A, B, C, and D are random variables represented by the nodes of the network graph.**
- **If we are considering node B, which is connected with node A by a directed arrow, then node A is called the parent of Node B.**
- **Node C is independent of node A.**

- The Bayesian network has mainly two components:
- **Causal Component**
- **Actual numbers**
- Each node in the Bayesian network has condition probability distribution $\mathbf{P(X_i \mid Parent(X_i))}$, which determines the effect of the parent on that node.
- Bayesian network is based on Joint probability distribution and conditional probability. So let's first understand the joint probability distribution:

Joint probability distribution:

If we have variables x1, x2, x3,....., xn, then the probabilities of a different combination of x1, x2, x3.. xn, are known as Joint probability distribution.

$\mathbf{P[x_1, x_2, x_3,....., x_n]}$, it can be written as the following way in terms of the joint probability distribution.

$$\mathbf{= P[x_1| x_2, x_3,....., x_n]P[x_2, x_3,....., x_n]}$$

$$\mathbf{= P[x_1| x_2, x_3,....., x_n]P[x_2|x_3,....., x_n]....P[x_{n-1}|x_n]P[x_n].}$$

In general for each variable Xi, we can write the equation as:

$$P(X_i|X_{i-1},.....,X_1) = P(X_i|Parents(X_i))$$

- **Explanation of Bayesian network:**
- Let's understand the Bayesian network through an example by creating a directed acyclic graph:
- **Example:** Harry installed a new burglar alarm at his home to detect burglary. The alarm reliably responds at detecting a burglary but also responds for minor earthquakes. Harry has two neighbors David and Sophia, who have taken a responsibility to inform Harry at work when they hear the alarm. David always calls Harry when he hears the alarm, but sometimes he got confused with the phone ringing and calls at that time too. On the other hand, Sophia likes to listen to high music, so sometimes she misses to hear the alarm. Here we would like to compute the probability of Burglary Alarm.
- **Problem:**
- **Calculate the probability that alarm has sounded, but there is neither a burglary, nor an earthquake occurred, and David and Sophia both called the Harry.**

- **Solution:**
- The Bayesian network for the above problem is given below. The network structure is showing that burglary and earthquake is the parent node of the alarm and directly affecting the probability of alarm's going off, but David and Sophia's calls depend on alarm probability.
- The network is representing that our assumptions do not directly perceive the burglary and also do not notice the minor earthquake, and they also not confer before calling.
- The conditional distributions for each node are given as conditional probabilities table or CPT.
- Each row in the CPT must be sum to 1 because all the entries in the table represent an exhaustive set of cases for the variable.
- In CPT, a boolean variable with k boolean parents contains 2^K probabilities. Hence, if there are two parents, then CPT will contain 4 probability values

- **List of all events occurring in this network:**
- **Burglary (B)**
- **Earthquake(E)**
- **Alarm(A)**
- **David Calls(D)**
- **Sophia calls(S)**

- We can write the events of problem statement in the form of probability: **P[D, S, A, B, E]**, can rewrite the above probability statement using joint probability distribution:

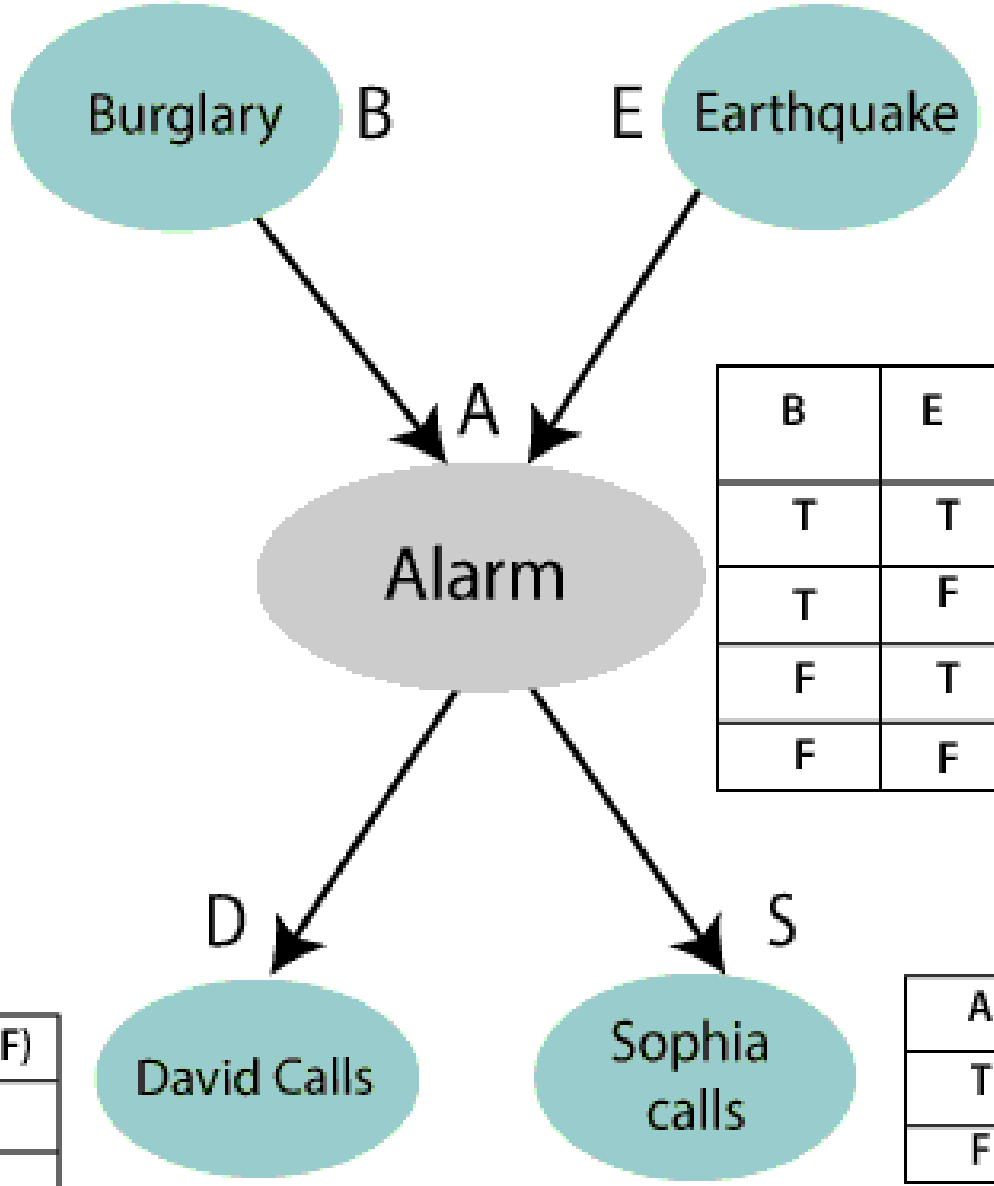
- **P[D, S, A, B, E]= P[D | S, A, B, E]. P[S, A, B, E]**

=P[D | S, A, B, E]. P[S | A, B, E]. P[A, B, E]

= P [D| A]. P [S| A, B, E]. P[A, B, E]

= P[D | A]. P[S | A]. P[A| B, E]. P[B, E]

= P[D | A]. P[S | A]. P[A| B, E]. P[B |E]. P[E]



T	0.002
F	0.998

T	0.001
F	0.999

B	E	P(A=T)	P(A=F)
T	T	0.94	0.06
T	F	0.95	0.04
F	T	0.69	0.69
F	F	0.999	0.999

A	P (D=T)	P (D=F)
T	0.91	0.09
F	0.05	0.95

A	P (S=T)	P (S=F)
T	0.75	0.25
F	0.02	0.98

- Let's take the observed probability for the Burglary and earthquake component:
- P(B= True) = 0.002, which is the probability of burglary.
- P(B= False)= 0.998, which is the probability of no burglary.
- P(E= True)= 0.001, which is the probability of a minor earthquake
- P(E= False)= 0.999, Which is the probability that an earthquake not occurred.
- We can provide the conditional probabilities as per the below tables:

- **Conditional probability table for Alarm A:**
- The Conditional probability of Alarm A depends on Burglar and earthquake:

The Conditional probability of Alarm A depends on Burglar and earthquake:

B	E	P(A= True)	P(A= False)
True	True	0.94	0.06
True	False	0.95	0.04
False	True	0.31	0.69
False	False	0.001	0.999

Conditional probability table for David Calls:

The Conditional probability of David that he will call depends on the probability of Alarm.

A	P(D= True)	P(D= False)
True	0.91	0.09
False	0.05	0.95

Conditional probability table for Sophia Calls:

The Conditional probability of Sophia that she calls is depending on its Parent Node "Alarm."

A	P(S= True)	P(S= False)
True	0.75	0.25
False	0.02	0.98

From the formula of joint distribution, we can write the problem statement in the form of probability distribution:

P(S, D, A, ¬B, ¬E) = P (S|A) *P (D|A)*P (A|¬B ^ ¬E) *P (¬B) *P (¬E).

= 0.75* 0.91* 0.001* 0.998*0.999

= 0.00068045.

- **Hence, a Bayesian network can answer any query about the domain by using Joint distribution.**
- **The semantics of Bayesian Network:**
- There are two ways to understand the semantics of the Bayesian network, which is given below:

1. To understand the network as the representation of the Joint probability distribution.

- It is helpful to understand how to construct the network.

2. To understand the network as an encoding of a collection of conditional independence statements.

- It is helpful in designing inference procedure.

Merits of Bayesian Networks

- Intuitive, graphical, and efficient
- Accounts for sources of uncertainty
- Allows for information updating
- Models multiple interdependencies
- Models distributed & interacting systems
- Identifies critical components & cut sets
- Includes utility and decision nodes

- **Benefits of Bayesian Networks**
- Bayesian networks **provide a more efficient way to represent and compute joint probabilities by exploiting the conditional independence properties of the variables in the network**.
- In Bayesian networks, you represent the joint probability distribution by decomposing it into a product of conditional probabilities.
- Bayesian Networks are **more extensible than other networks and learning methods**.
- Adding a new piece in the network requires only a few probabilities and a few edges in the graph.
- So, it is an excellent network for adding a new piece of data to an existing probabilistic model.
- The graph of a Bayesian Network is useful.

- **Why learn Bayesian networks?**
- A Bayesian network is **a compact, flexible and interpretable representation of a joint probability distribution**.
- It is also an useful tool in knowledge discovery as directed acyclic graphs allow representing causal relations between variables.
- Bayesian networks are **helpful for solving probabilistic problems**. Such issues depend on too many variables.
- Since Bayesian networks are probabilistic in nature, they help predict events and derive relationships between multiple variables or events.

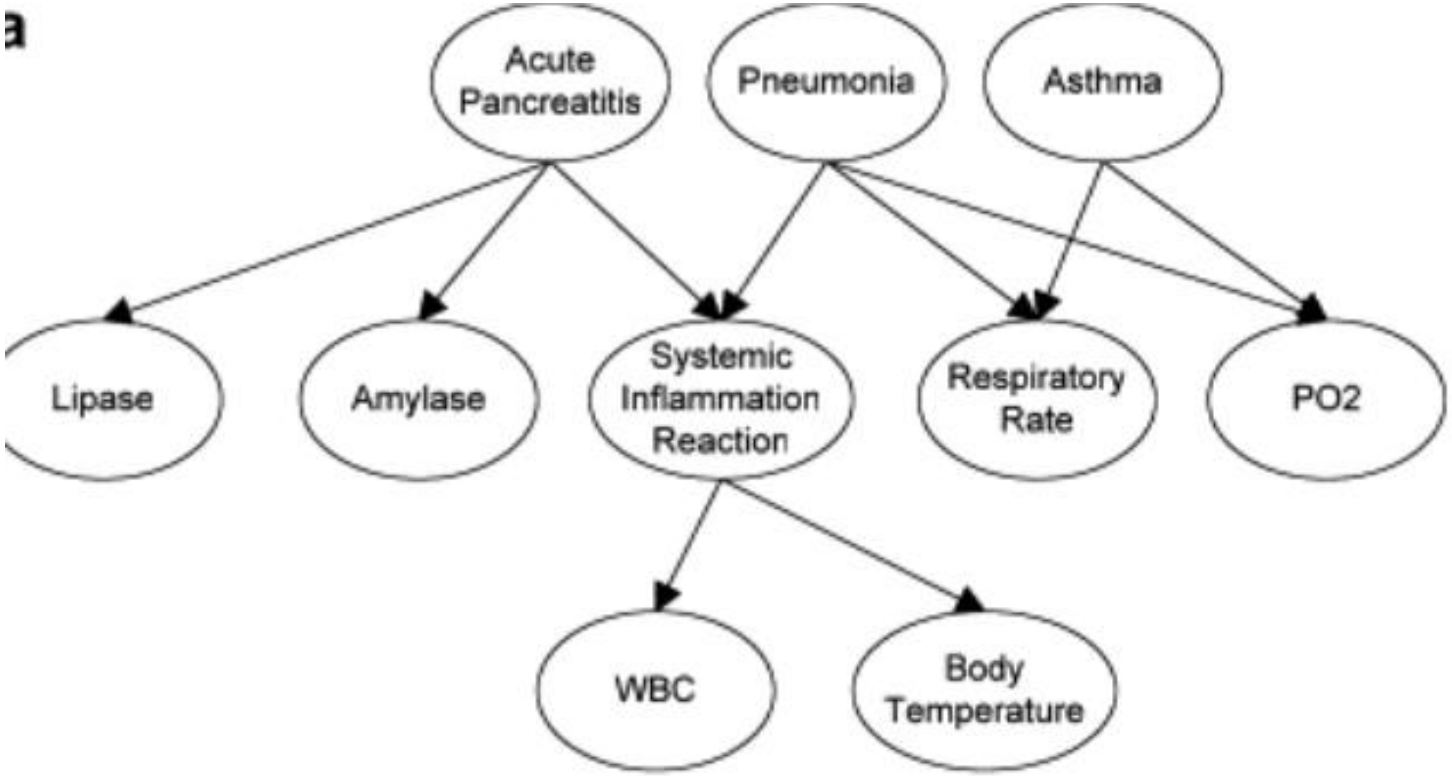
- **Applications of Bayesian Networks in AI**
- **Prediction and classification:** Bayesian belief networks can be used to predict the probability of an event or classify data into different categories based on a set of inputs. This is useful in areas such as fraud detection, medical diagnosis, and image recognition.
- **Decision making:** Bayesian networks can be used to **make decisions based on uncertain or incomplete information**. For example, they can be used to determine the optimal route for a delivery truck based on traffic conditions and delivery schedules.
- **Risk analysis:** Bayesian belief networks can be used to analyze the risks associated with different actions or events. **This is useful in areas such as financial planning, insurance, and safety analysis.**

- **Applications of Bayesian Networks in AI**

- **Anomaly detection:** Bayesian networks can be used to detect anomalies in data, such as outliers or unusual patterns. This is useful in areas such as cybersecurity, where unusual network traffic may indicate a security breach.
- **Natural language processing:** Bayesian belief networks can be used to model the probabilistic relationships between words and phrases in natural language, which is useful in applications such as language translation and sentiment analysis.

• Example-Medical Diagnostics

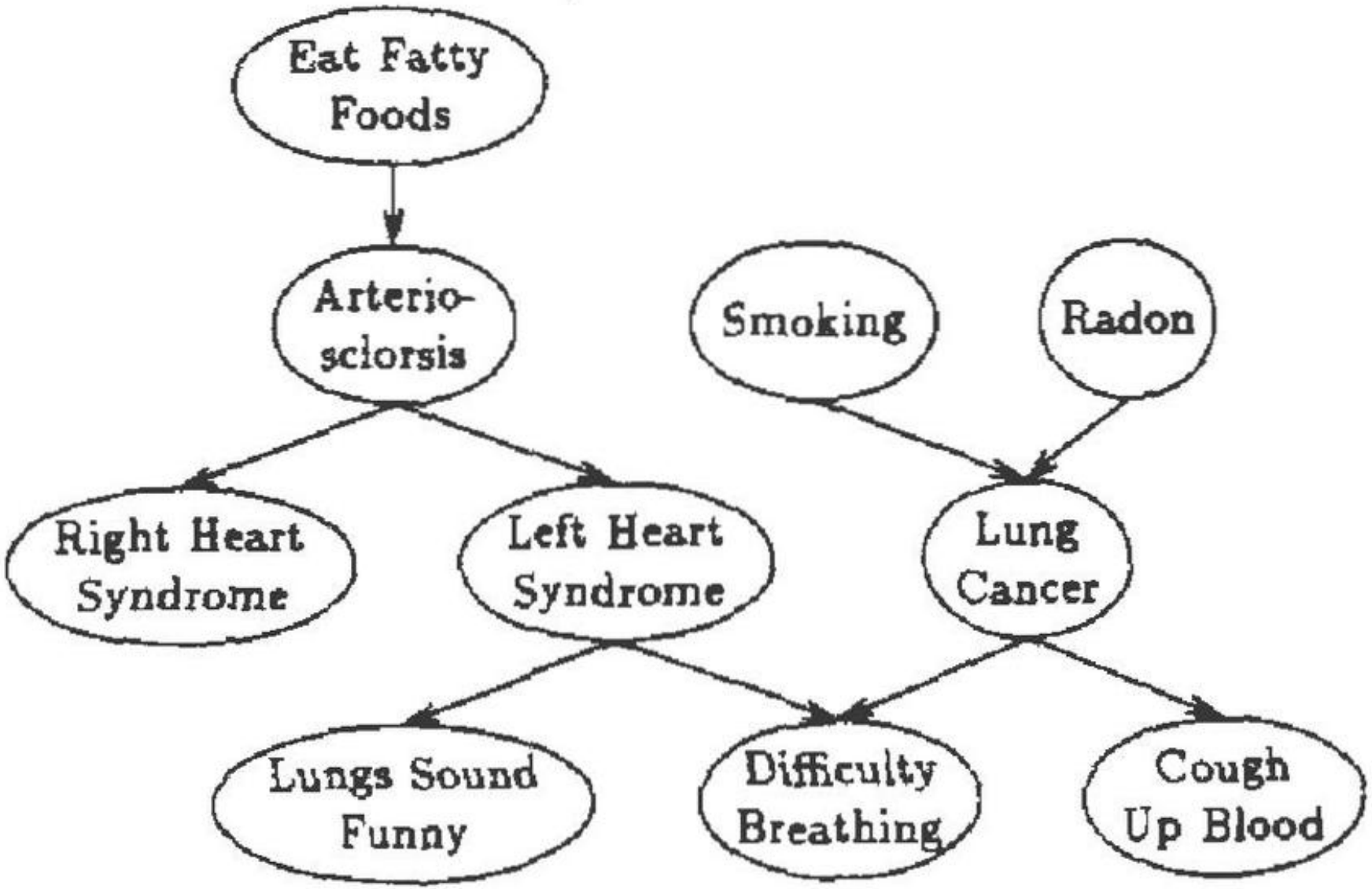
a



b

Pneumonia	Asthma	PO2	Probability
Positive	Positive	<60	0.6
		60~80	0.2
		>80	0.2
Positive	Negative	<60	0.3
		60~80	0.3
		>80	0.4
Negative	Positive	<60	0.4
		60~80	0.3
		>80	0.3
Negative	Negative	<60	0.05
		60~80	0.1
		>80	0.85

Example-Medical Diagnostics



- **Software/Open Source Tools**
- Bayesian Network, also known as Bayes network is a probabilistic directed acyclic graphical model, which can be used for time series prediction, anomaly detection, diagnostics and more.

1| BUGS

- Bayesian inference Using Gibbs Sampling or BUGS is a software package for the Bayesian analysis of **statistical models** by utilising the Markov chain Monte Carlo techniques. The tool carries a system that determines an appropriate Markov chain Monte Carlo scheme, which is based on the Gibbs sampler for analysing the designated model.

2| BNFinder

- BNFinder or Bayes Net Finder is an open-source tool for learning **Bayesian networks written purely in Python**. The BNF script is the main part of BNfinder command-line tools. It is used for learning the Bayesian network from data and can be executed by typing *bnf <options>*. It can be used for both dynamic and static networks.

3| bnlearn

- bnlearn is an R package for learning the graphical structure of Bayesian networks, estimate their parameters and perform some useful inference. It includes various algorithms for learning the **structure of Bayesian networks with discrete as well as continuous variables.**
- bnlearn implements several constraint-based structure learning algorithms, including Fast Incremental Association (Fast-IAMB), Incremental Association Markov Blanket (IAMB), Hybrid Parents & Children (HPC), among others

4| Banjo

- Banjo is a software application and framework written to comply with Java 5 for structure learning of static and dynamic Bayesian networks.
- The tool is mainly designed to provide efficient structure inference when analysing **research-oriented data sets.**
- Banjo focuses on score-based structure inference, which is a plethora of code that already exists for variable inference within a Bayesian network of known structure.

5| Free-BN

- Free-BN or FBN is an open-source Bayesian network structure learning API licensed under the Apache 2.0 license. **This tool is meant for constraint-based structural learning of Bayesian networks.** The features of FBN include structural learning, exact inference and logic sampling.
- The FBN API is dependent on two other minor projects called Free-Display and Free-GA (FGA). While Free-Display API is used to visualise the Bayesian networks, FGA API is used for search-and-scoring methods for Bayesian network structure learning.

6| jBNC

- jBNC is a library of Java classes for training and testing Bayesian network classifiers. **This Java toolkit is mainly used for training, testing, and applying Bayesian Network Classifiers.**
- The classifiers implemented by jBNC have been shown to perform well in a variety of artificial intelligence, machine learning, and data mining applications.

7| JavaBayes

- It is a set of tools used for the creation and manipulation of Bayesian networks. **The system is composed of a graphical editor, a core inference engine and a set of parsers.** The graphical editor allows a user to create and modify Bayesian networks in a friendly interface, while the parsers allow a user to import Bayesian networks in a variety of formats.
- Also, the core inference engine is responsible for manipulating the data structures that represent Bayesian networks and can produce marginal probability for any variable in a Bayesian network.

8| UnBBayes

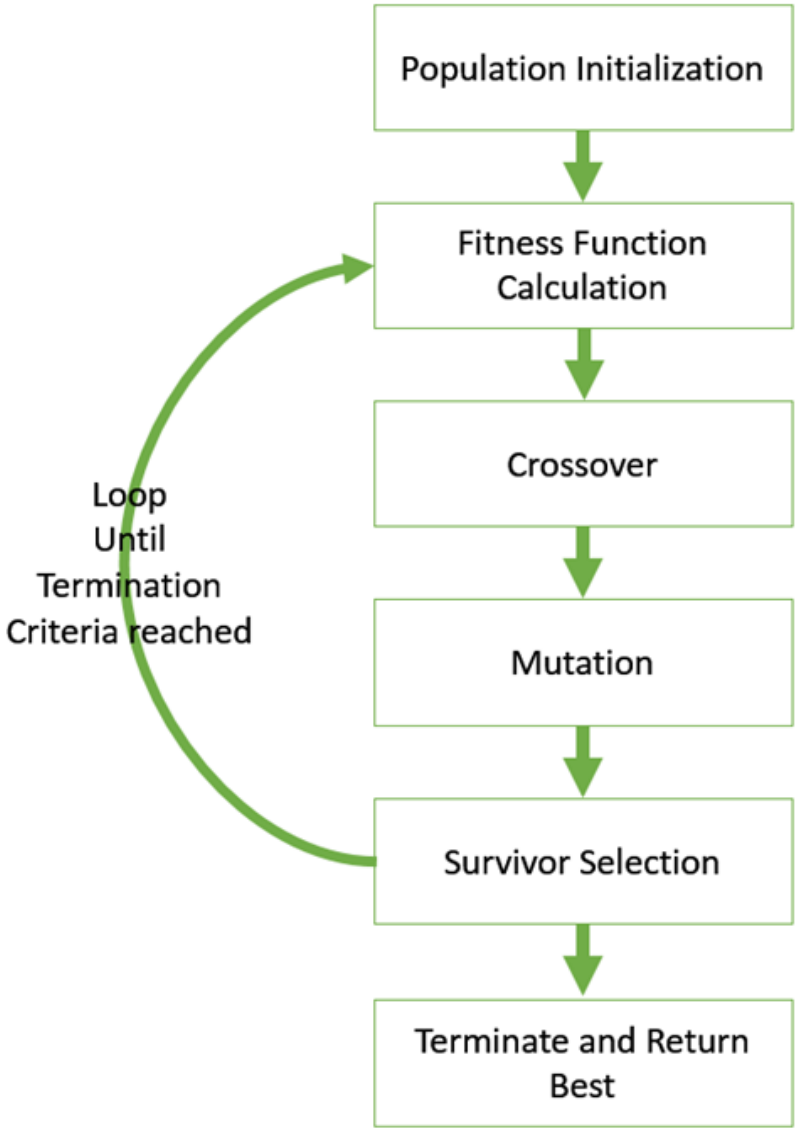
- UnBBayes is a probabilistic network framework written in Java language that has both a GUI and an API with inference, sampling, learning and evaluation.
- The framework supports Bayesian networks, influence diagrams, MSBN, HBN, PRM, structure, parameter and incremental learning, among others.
- Some other features of UnBBayes include decision graphs, approximate inference, Noisy-Max distribution, plug-in support, data mining and more.

HINDUSTAN
INSTITUTE OF TECHNOLOGY & SCIENCE
(DEEMED TO BE UNIVERSITY)

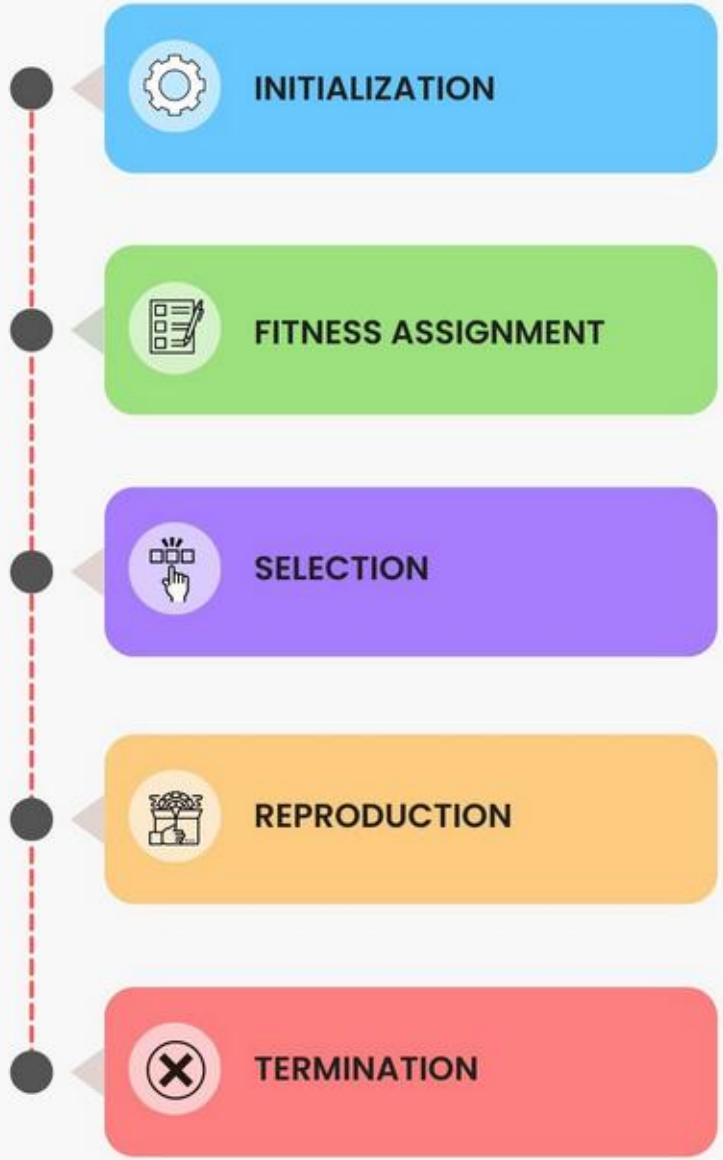
EAL51003 – ARTIFICIAL INTELLIGENCE
B.Tech[AIML] – III Semester

K.Kowsalya
Assistant Professor (SS)
School of Computing Sciences,
Department of Computer Science and Engineering

UNIT-II
GENETIC ALGORITHMS



How Genetic Algorithm Works:



GENETIC ALGORITHM

- 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.

Introduction to Optimization

- Optimization is the process of **making something better**.
- In any process, we have a set of inputs and a set of outputs as shown in the following figure.

- 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 is Genetic Algorithm?

- The genetic algorithm is based on the genetic structure and behavior of the chromosome of the population. The following things are the foundation of genetic algorithms.
- Each chromosome indicates a possible solution. **Thus the population is a collection of chromosomes.**
- A fitness function characterizes each individual in the population. Therefore, greater fitness better is the solution.
- **Out of the available individuals in the population, the best individuals are used to reproduce the next generation offsprings.**
- The offspring produced will have features of both the parents and is a result of mutation. **A mutation is a small change in the gene structure**.

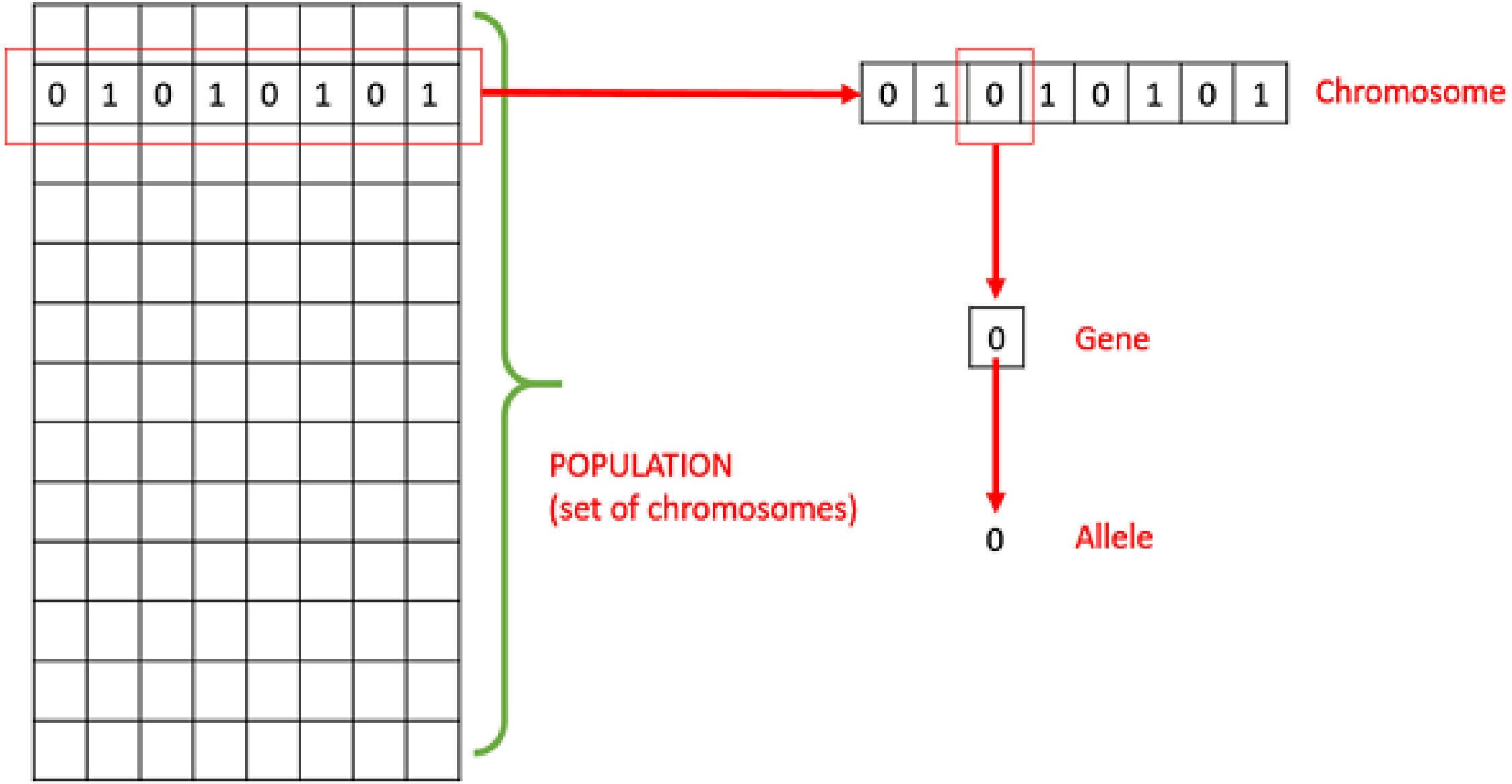
- **Advantages of GAs**
- GAs have various advantages which have made them immensely popular. These include –
- Does not require any derivative information (which may not be available for many real-world problems).
- Is faster and more efficient as compared to the traditional methods.
- Has very good 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 GAs**
- Like any technique, GAs also suffer from a few limitations. These include –
- 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.

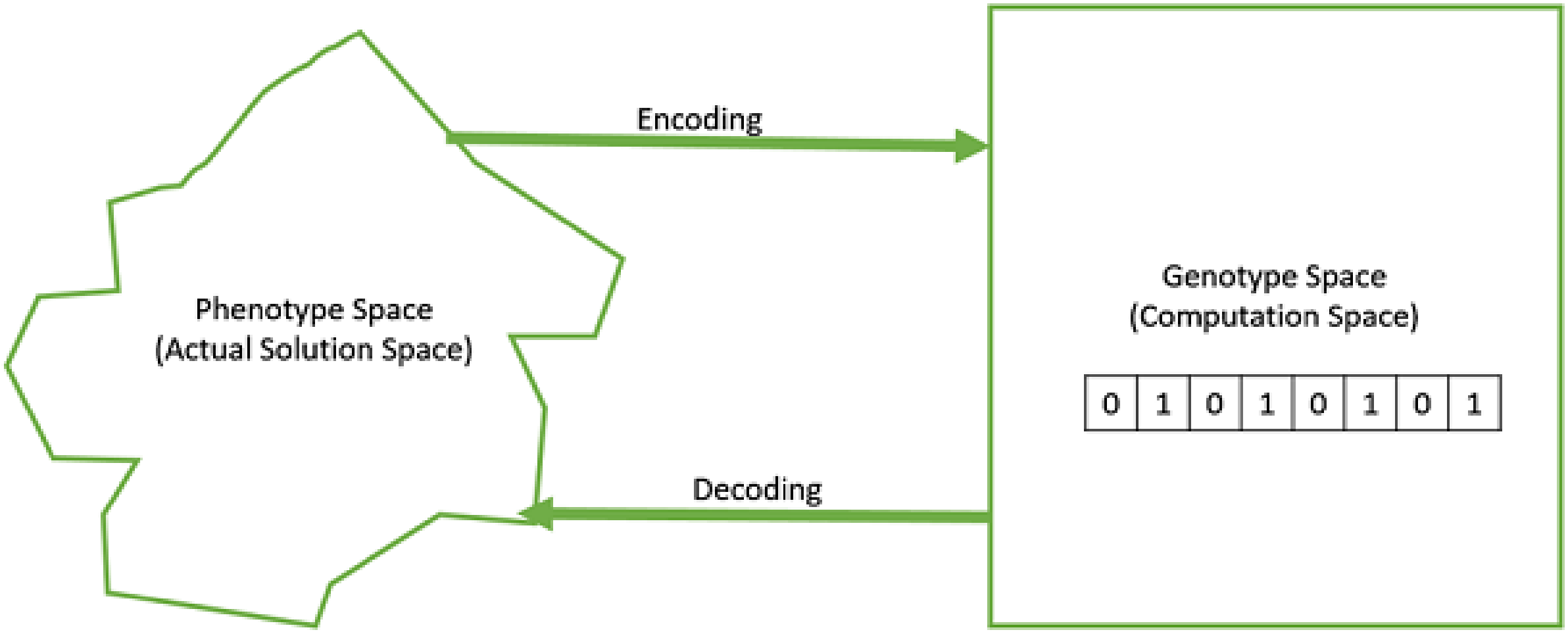
- **GA – Motivation**
- **Genetic Algorithms have the ability to deliver a "good-enough" solution "fast-enough". This makes genetic algorithms attractive for use in solving optimization problems.** The reasons why GAs are needed are as follows –
- **Solving Difficult Problems-NP-Hard**.
- GAs prove to 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.

Genetic Algorithms - Fundamentals

- **Basic Terminology**
- Before beginning a discussion on Genetic Algorithms, it is essential to be familiar with some basic terminology which will be used throughout this tutorial.
- **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.

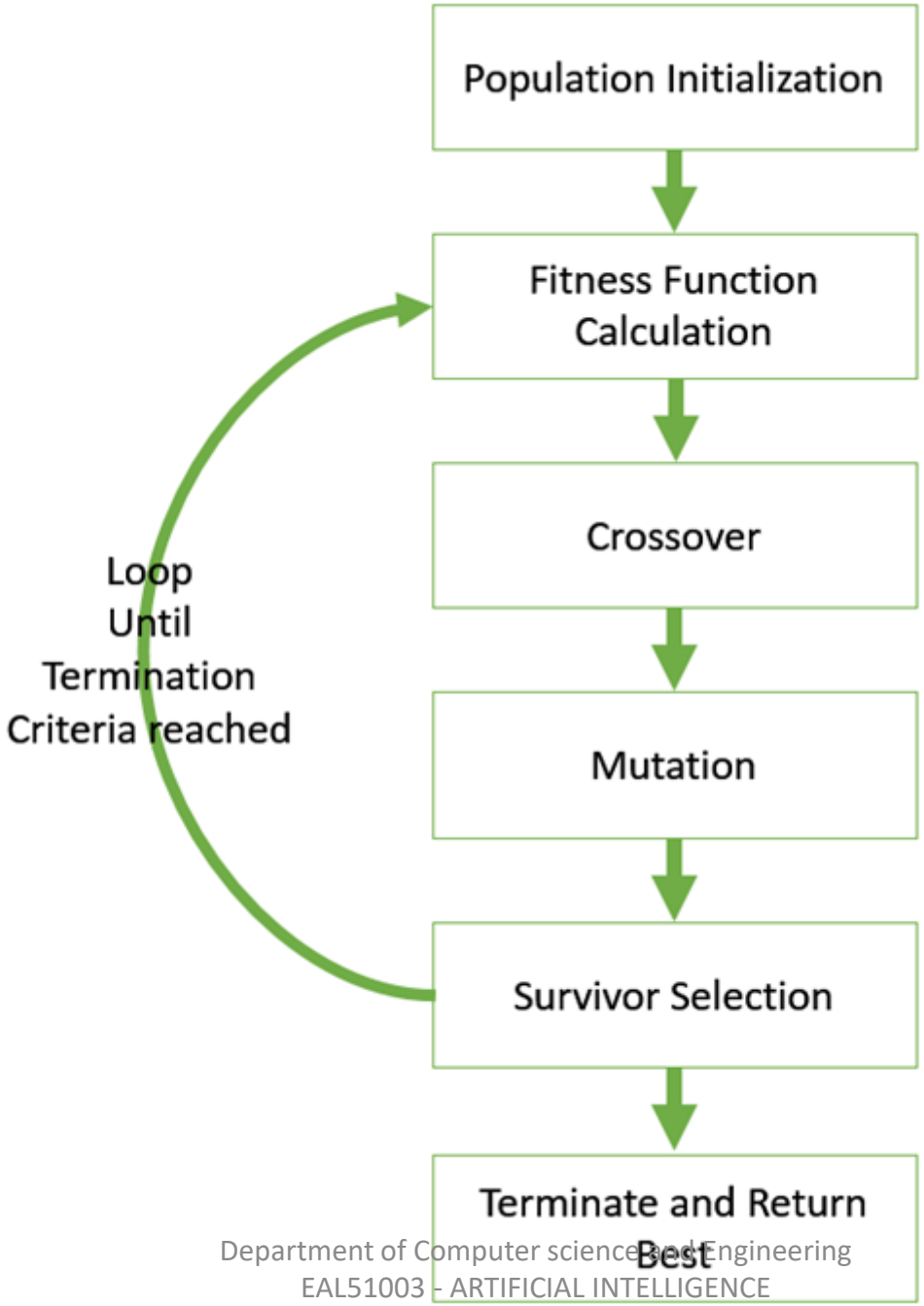

0 1 0 1 0 1 0 1
0 1 0 1 0 1 0 1
Chromosome
0
Gene
POPULATION
(set of chromosomes)
0
Allele

- **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.


Encoding
Phenotype Space
(Actual Solution Space)
Genotype Space
(Computation Space)
0 1 0 1 0 1 0 1
Decoding

- 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**
- The basic structure of a GA is as follows –
- 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.


Population Initialization
Fitness Function Calculation
Crossover
Mutation
Survivor Selection
Terminate and Return Best
Loop Until Termination Criteria reached

A generalized pseudo-code for a GA is explained in the following program –

```
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 vs Phenotype

Genotype of an organism is defined as an actual or complete genetic makeup of an organism. It also refers to the pair of alleles inherited in an individual for a particular gene.

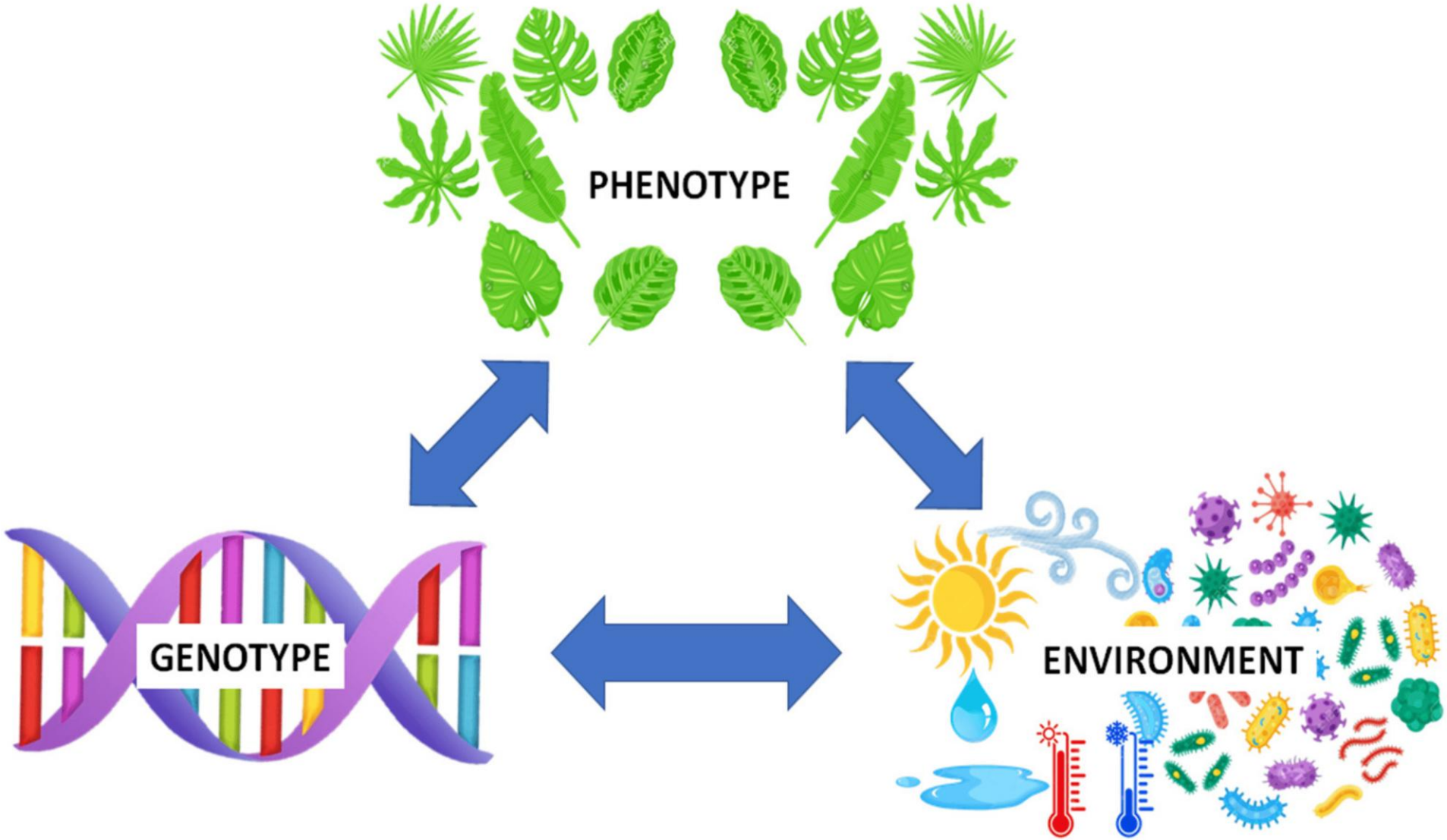
Example

Phenotype refers to an individual's observable traits, such as height, eye color and blood type. A person's phenotype is determined by both their genomic makeup (genotype) and environmental factors.

Example

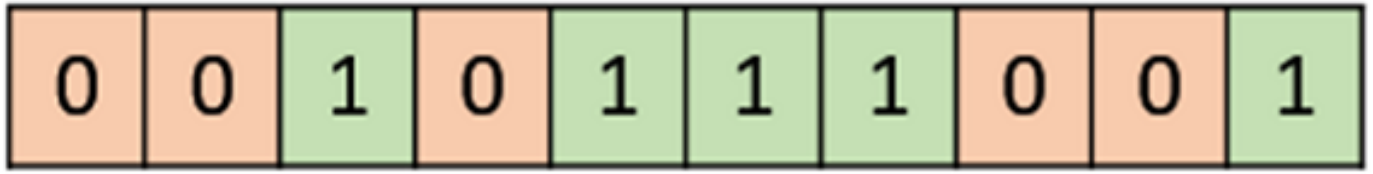
Hair colour

Blood group


PHENOTYPE
GENOTYPE
ENVIRONMENT

- **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.

- **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).



- **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

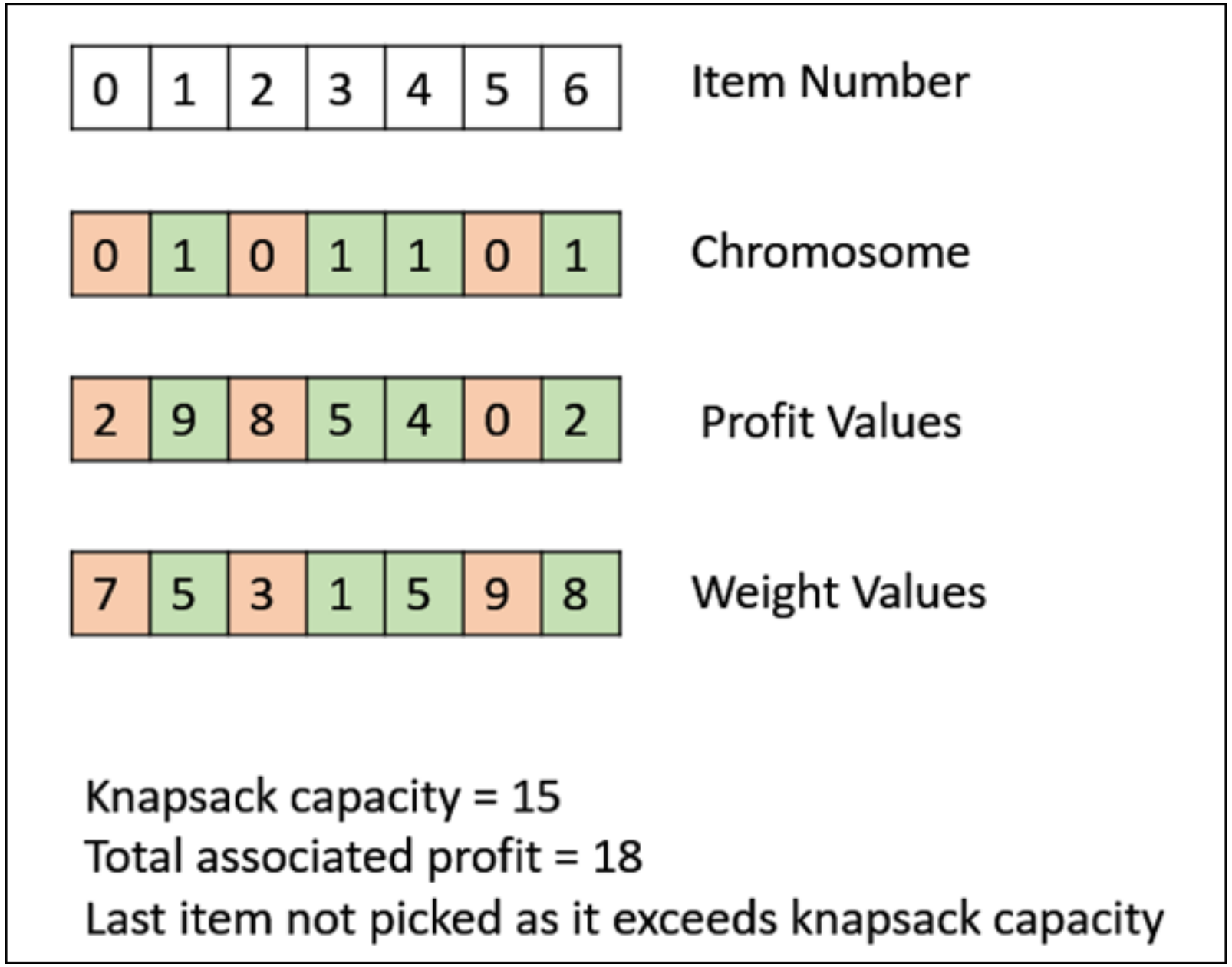
- **Integer Representation**
- For discrete valued genes, we cannot always limit the solution space to binary ‘yes’ or ‘no’. For example, if we want to encode the four distances – North, South, East and West, we can encode them as **{0,1,2,3}**. In such cases, integer representation is desirable.

1	2	3	4	3	2	4	1	2	1

- **Genetic Algorithms - Fitness Function**
- The fitness function simply defined is a function which takes a **candidate solution to the problem as input and produces as output** how "fit" our how "good" the solution is with respect to the problem in consideration.
- Calculation of fitness value is done repeatedly in a GA and therefore it should be sufficiently fast.
- **Fitness Function** (also known as the **Evaluation Function**) evaluates how close a given solution is to the optimum solution of the desired problem. It determines how fit a solution is.

- A slow computation of the fitness value can adversely affect a GA and make it exceptionally slow.
- In most cases the fitness function and the objective function are the same as the objective is to either maximize or minimize the given objective function.
- However, for more complex problems with multiple objectives and constraints, an **Algorithm Designer** might choose to have a different fitness function.

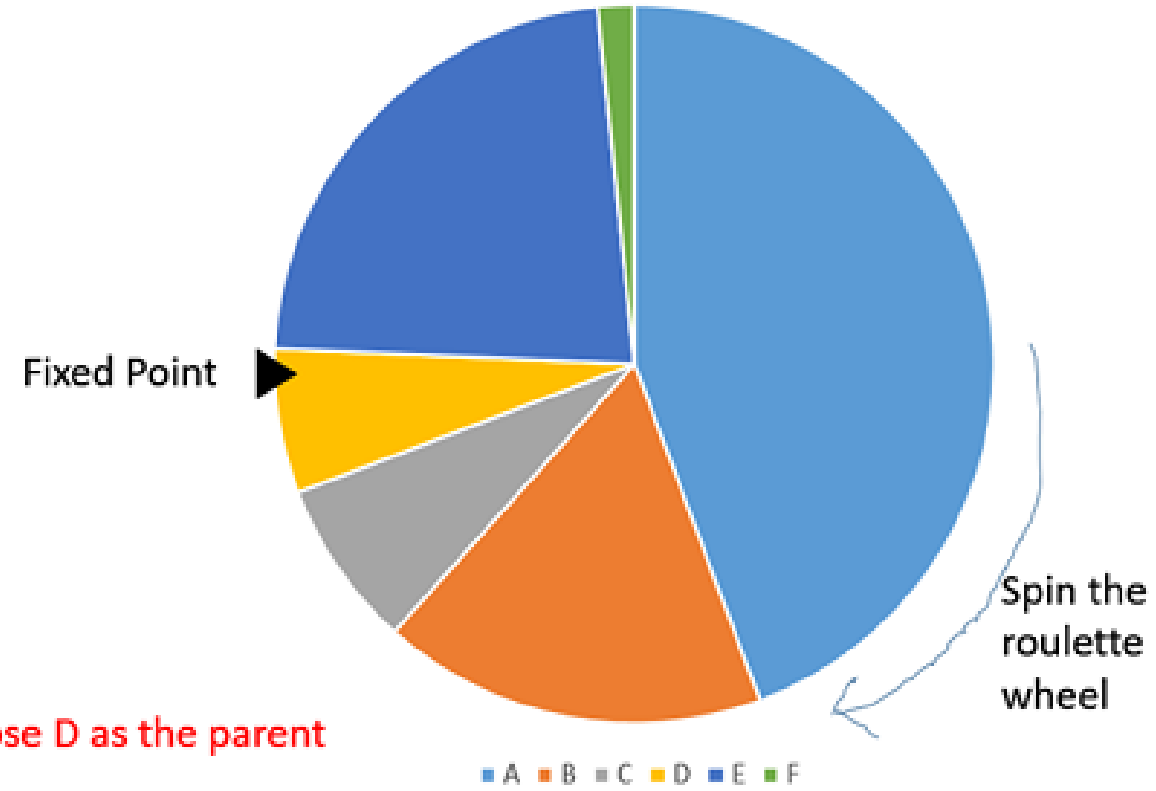
- A fitness function should possess the following characteristics –
- The fitness function should be sufficiently fast to compute.
- It must quantitatively measure how fit a given solution is or how fit individuals can be produced from the given solution.
- In some cases, calculating the fitness function directly might not be possible due to the inherent complexities of the problem at hand. In such cases, we do fitness approximation to suit our needs.
- The following image shows the fitness calculation for a solution of the 0/1 Knapsack. It is a simple fitness function which just sums the profit values of the items being picked (which have a 1), scanning the elements from left to right till the knapsack is full.


0 1 2 3 4 5 6
Item Number
0 1 0 1 1 0 1
Chromosome
2 9 8 5 4 0 2
Profit Values
7 5 3 1 5 9 8
Weight Values
Knapsack capacity = 15
Total associated profit = 18
Last item not picked as it exceeds knapsack capacity

- **Genetic Algorithms - Parent Selection**
- Parent Selection is the process of selecting parents which mate and recombine to create off-springs for the next generation.
- Parent selection is very crucial to the convergence rate of the GA as good parents drive individuals to a better and fitter solutions.
- However, care should be taken to prevent one extremely fit solution from taking over the entire population in a few generations, as this leads to the solutions being close to one another in the solution space thereby leading to a loss of diversity.
- **Maintaining good diversity** in the population is extremely crucial for the success of a GA. This taking up of the entire population by one extremely fit solution is known as **premature convergence** and is an undesirable condition in a GA

- **Fitness Proportionate Selection**
- Fitness Proportionate Selection is one of the most popular ways of parent selection. In this every individual can become a parent with a probability which is proportional to its fitness.
- Therefore, fitter individuals have a higher chance of mating and propagating their features to the next generation.
- Therefore, such a selection strategy applies a selection pressure to the more fit individuals in the population, evolving better individuals over time.
- Consider a circular wheel. The wheel is divided into **n pies**, where n is the number of individuals in the population.
- Each individual gets a portion of the circle which is proportional to its fitness value.
- Two implementations of fitness proportionate selection are possible –

- **Roulette Wheel Selection**
- In a roulette wheel selection, the circular wheel is divided as described before. A fixed point is chosen on the wheel circumference as shown and the wheel is rotated. The region of the wheel which comes in front of the fixed point is chosen as the parent. For the second parent, the same process is repeated.

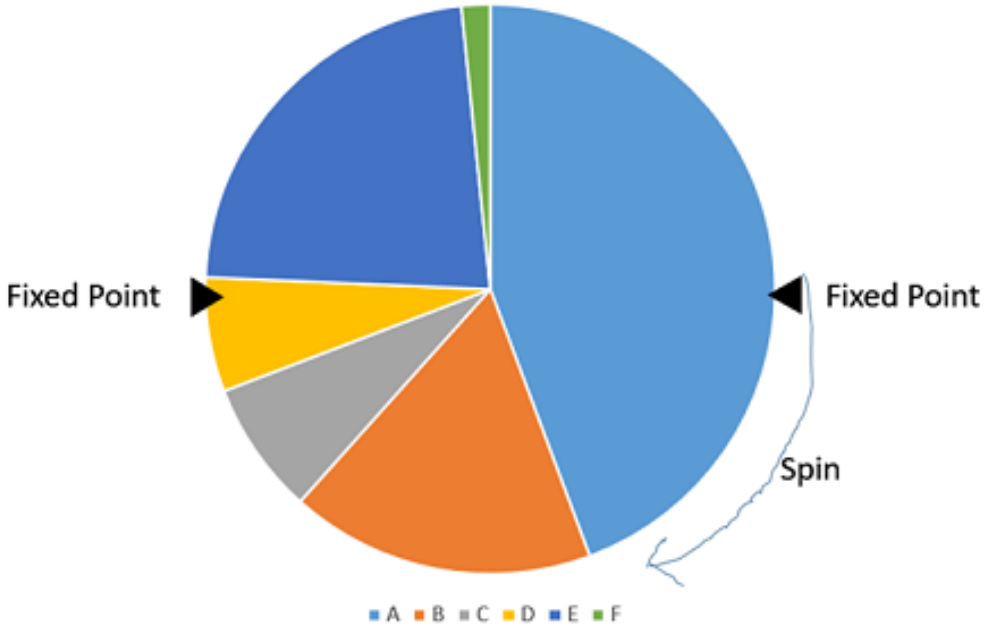


Chromosome	Fitness Value
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3

Choose D as the parent

- It is clear that a fitter individual has a greater pie on the wheel and therefore a greater chance of landing in front of the fixed point when the wheel is rotated. Therefore, the probability of choosing an individual depends directly on its fitness.
- Implementation wise, we use the following steps −
- Calculate S = the sum of a finesses.
- Generate a random number between 0 and S.
- Starting from the top of the population, keep adding the finesses to the partial sum P, till P<S.
- The individual for which P exceeds S is the chosen individual.

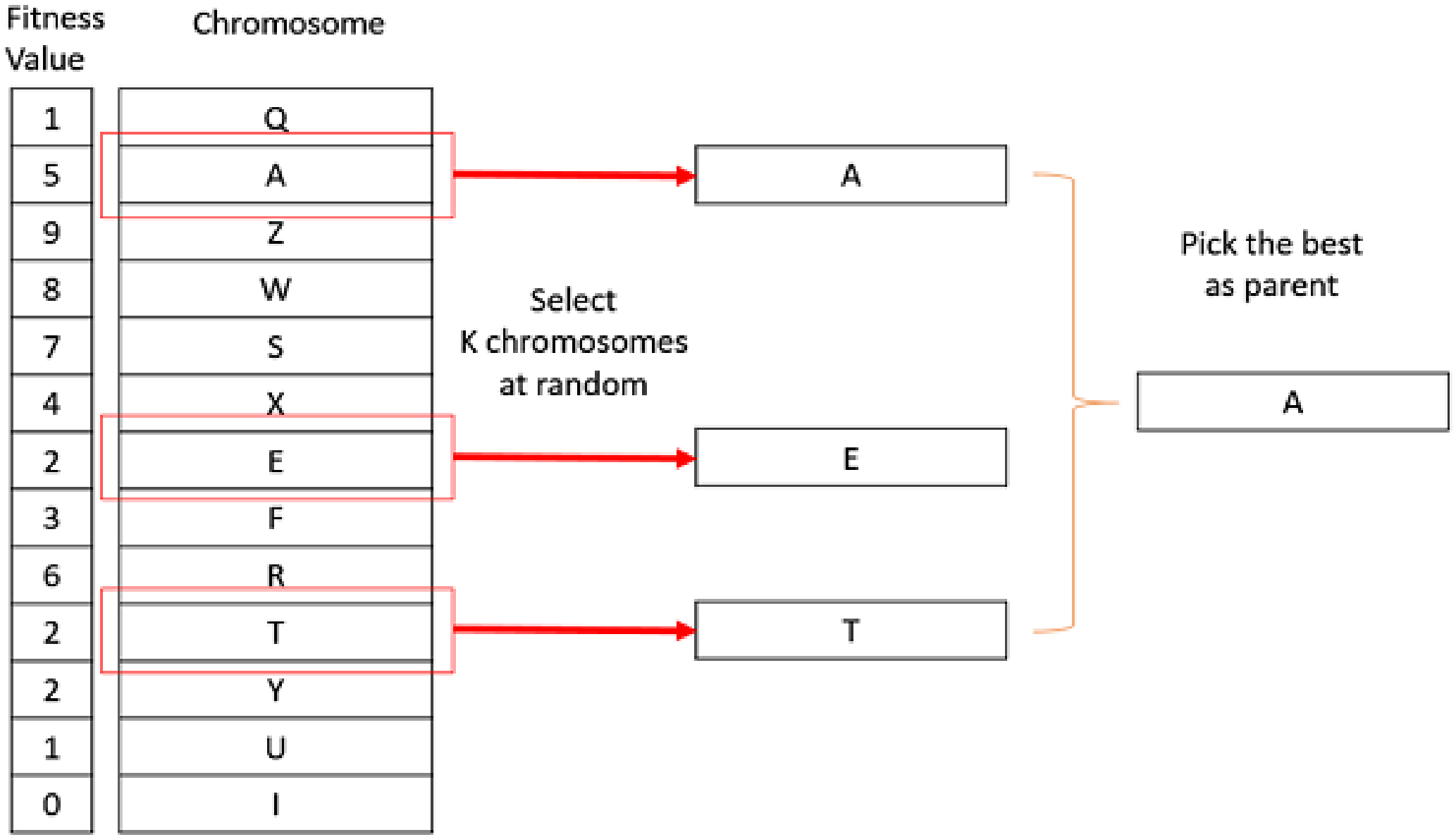
- **Stochastic Universal Sampling (SUS)**
- Stochastic Universal Sampling is quite similar to Roulette wheel selection, however instead of having just one fixed point, we have multiple fixed points as shown in the following image. Therefore, all the parents are chosen in just one spin of the wheel. Also, such a setup encourages the highly fit individuals to be chosen at least once.



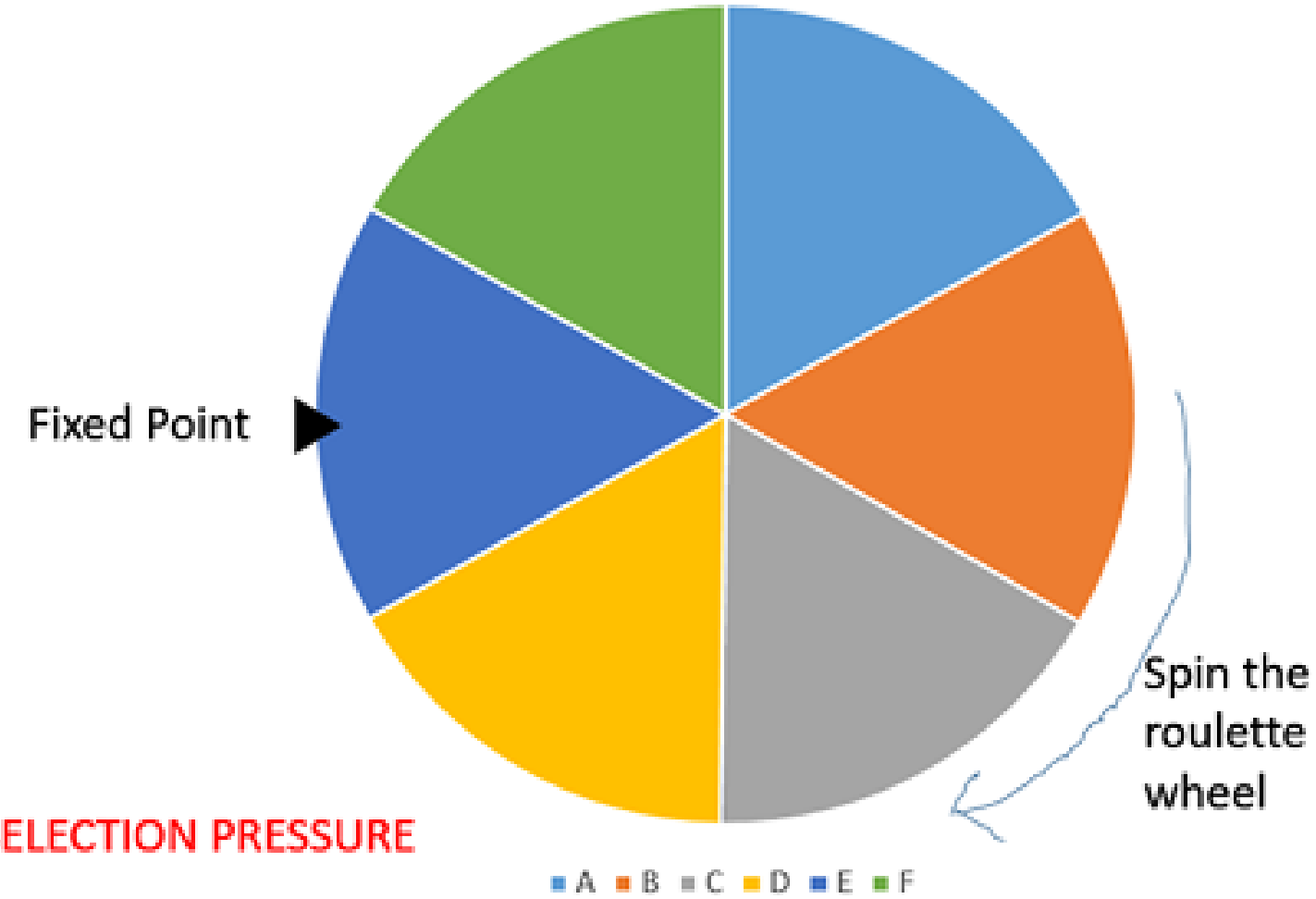
Chromosome	Fitness Value
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3

- It is to be noted that fitness proportionate selection methods don't work for cases where the fitness can take a negative value.

- **Tournament Selection**
- In K-Way tournament selection, we select K individuals from the population at random and select the best out of these to become a parent.
- The same process is repeated for selecting the next parent.
- Tournament Selection is also extremely popular in literature as it can even work with negative fitness values.


Fitness Value
Chromosome
1
Q
5
A
9
Z
8
W
7
S
4
X
2
E
3
F
6
R
2
T
2
Y
1
U
0
I
Select K chromosomes at random
A
E
T
Pick the best as parent
A

- **Rank Selection**
- Rank Selection also works with negative fitness values and is mostly used when the individuals in the population have very close fitness values (this happens usually at the end of the run).
- This leads to each individual having an almost equal share of the pie (like in case of fitness proportionate selection) as shown in the following image and hence each individual no matter how fit relative to each other has an approximately same probability of getting selected as a parent.
- This in turn leads to a loss in the selection pressure towards fitter individuals, making the GA to make poor parent selections in such situations.



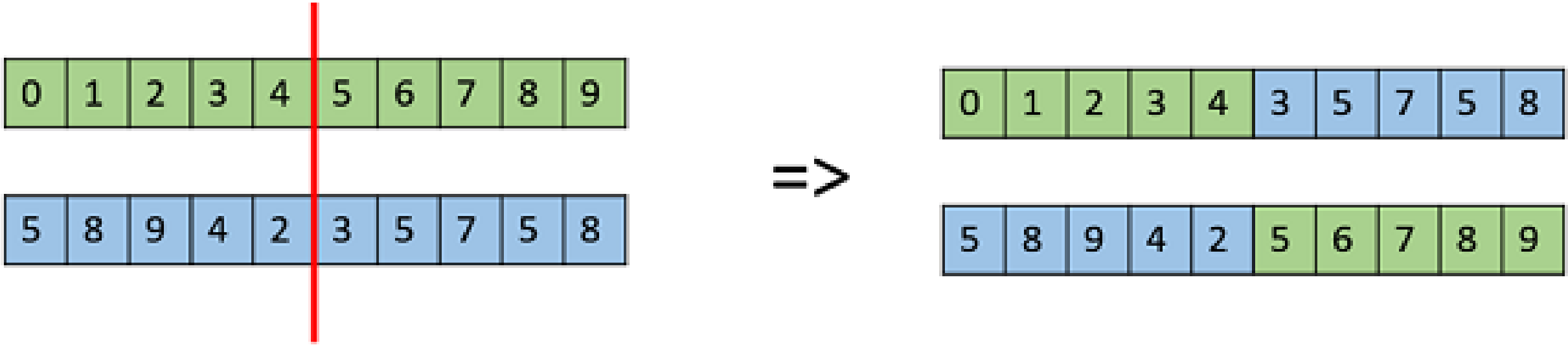
Chromosome	Fitness Value
A	8.1
B	8.0
C	8.05
D	7.95
E	8.02
F	7.99

- In this, we remove the concept of a fitness value while selecting a parent. However, every individual in the population is ranked according to their fitness. The selection of the parents depends on the rank of each individual and not the fitness. The higher ranked individuals are preferred more than the lower ranked ones.

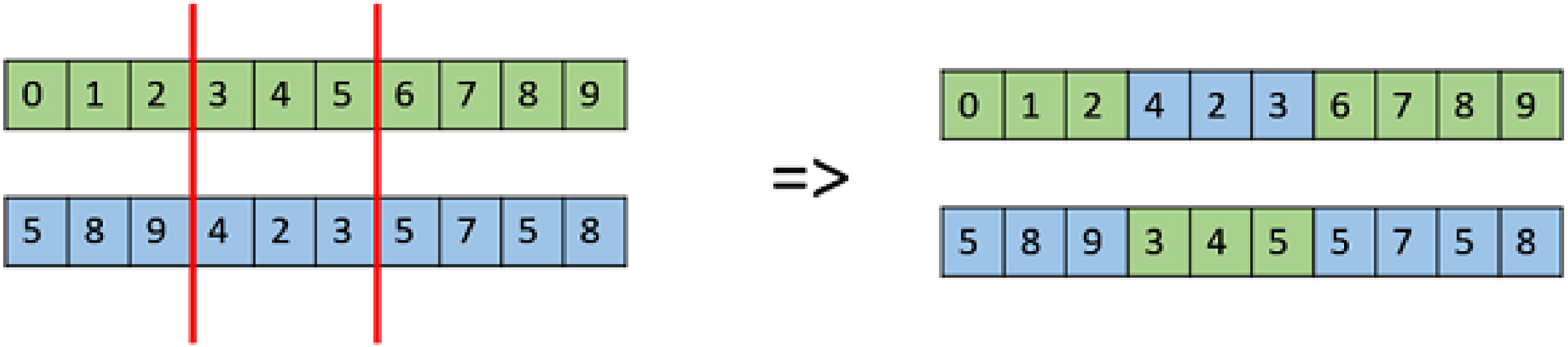
Chromosome	Fitness Value	Rank
A	8.1	1
B	8.0	4
C	8.05	2
D	7.95	6
E	8.02	3
F	7.99	5

- **Random Selection**
- In this strategy we randomly select parents from the existing population. There is no selection pressure towards fitter individuals and therefore this strategy is usually avoided.
- **Genetic Algorithms - Crossover**
- **Introduction to Crossover**
- The crossover operator is analogous to reproduction and biological crossover.
- In this more than one parent is selected and one or more off-springs are produced using the genetic material of the parents.
- Crossover is usually applied in a GA with a high probability – $\boldsymbol{p_c}$.

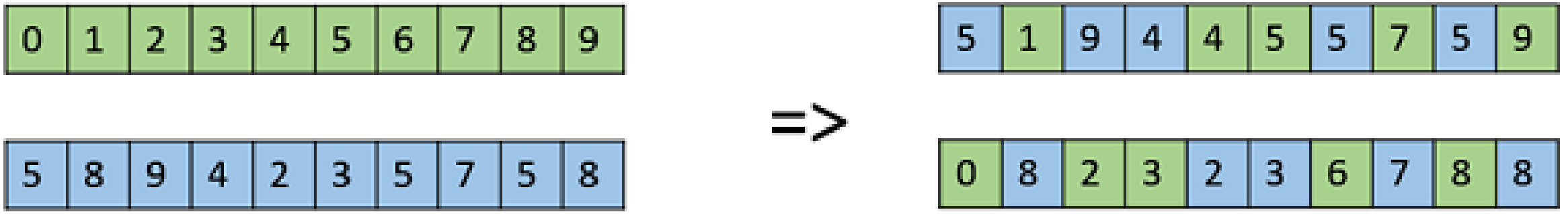
- **Crossover Operators**
- In this section we will discuss some of the most popularly used crossover operators.
- It is to be noted that these crossover operators are very generic and the GA Designer might choose to implement a problem-specific crossover operator as well.
- **One Point Crossover**
- In this one-point crossover, a random crossover point is selected and the tails of its two parents are swapped to get new off-springs.


0 1 2 3 4 5 6 7 8 9
5 8 9 4 2 3 5 7 5 8
=>
0 1 2 3 4 3 5 7 5 8
5 8 9 4 2 5 6 7 8 9

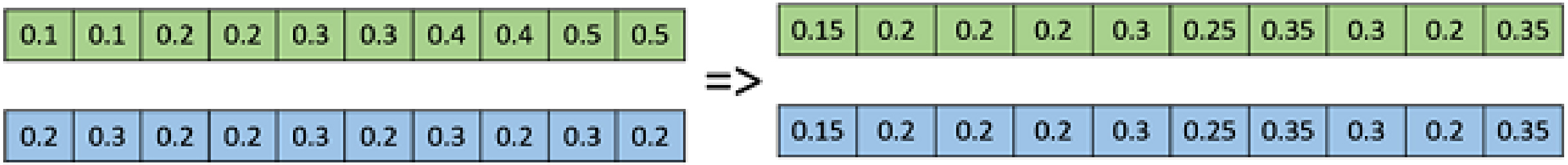
- **Multi Point Crossover**
- Multi point crossover is a generalization of the one-point crossover wherein alternating segments are swapped to get new off-springs.



- **Uniform Crossover**
- In a uniform crossover, we don’t divide the chromosome into segments, rather we treat each gene separately.
- In this, we essentially flip a coin for each chromosome to decide whether or not it’ll be included in the off-spring.
- We can also bias the coin to one parent, to have more genetic material in the child from that parent.



- **Whole Arithmetic Recombination**
- This is commonly used for integer representations and works by taking the weighted average of the two parents by using the following formulae −
- Child1 = α.x + (1-α).y
- Child2 = α.x + (1-α).y
- Obviously, if α = 0.5, then both the children will be identical as shown in the following image.



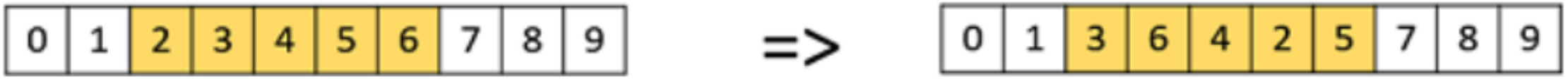
- **Genetic Algorithms - Mutation**
- **Introduction to Mutation**
- In simple terms, mutation may be defined as a small random tweak in the chromosome, to get a new solution.
- It is used to maintain and introduce diversity in the genetic population and is usually applied with a low probability – $\boldsymbol{p_m}$. If the probability is very high, the GA gets reduced to a random search.
- Mutation is the part of the GA which is related to the “exploration” of the search space.
- It has been observed that mutation is essential to the convergence of the GA while crossover is not.

- **Mutation Operators**
- In this section, we describe some of the most commonly used mutation operators. Like the crossover operators, this is not an exhaustive list and the GA designer might find a combination of these approaches or a problem-specific mutation operator more useful.
- **Bit Flip Mutation**
- In this bit flip mutation, we select one or more random bits and flip them. This is used for binary encoded GAs.

- **Random Resetting**
- Random Resetting is an extension of the bit flip for the integer representation. In this, a random value from the set of permissible values is assigned to a randomly chosen gene.
- **Swap Mutation**
- In swap mutation, we select two positions on the chromosome at random, and interchange the values. This is common in permutation based encodings.

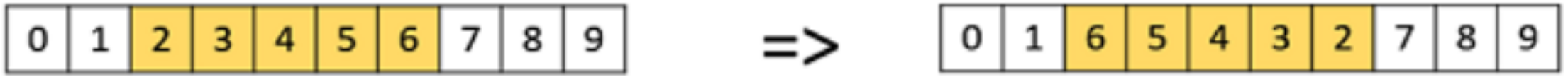
Scramble Mutation

Scramble mutation is also popular with permutation representations. In this, from the entire chromosome, a subset of genes is chosen and their values are scrambled or shuffled randomly.



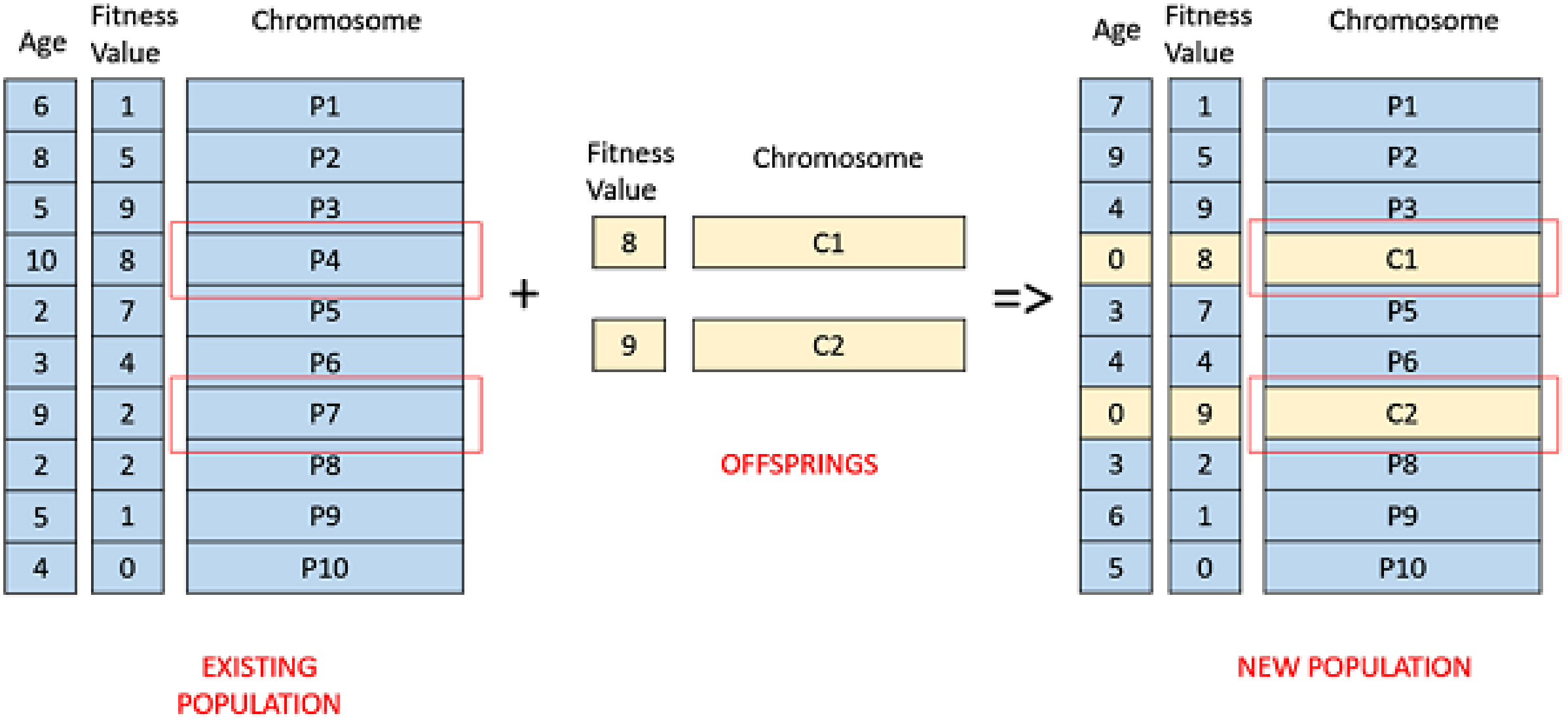
Inversion Mutation

In inversion mutation, we select a subset of genes like in scramble mutation, but instead of shuffling the subset, we merely invert the entire string in the subset.

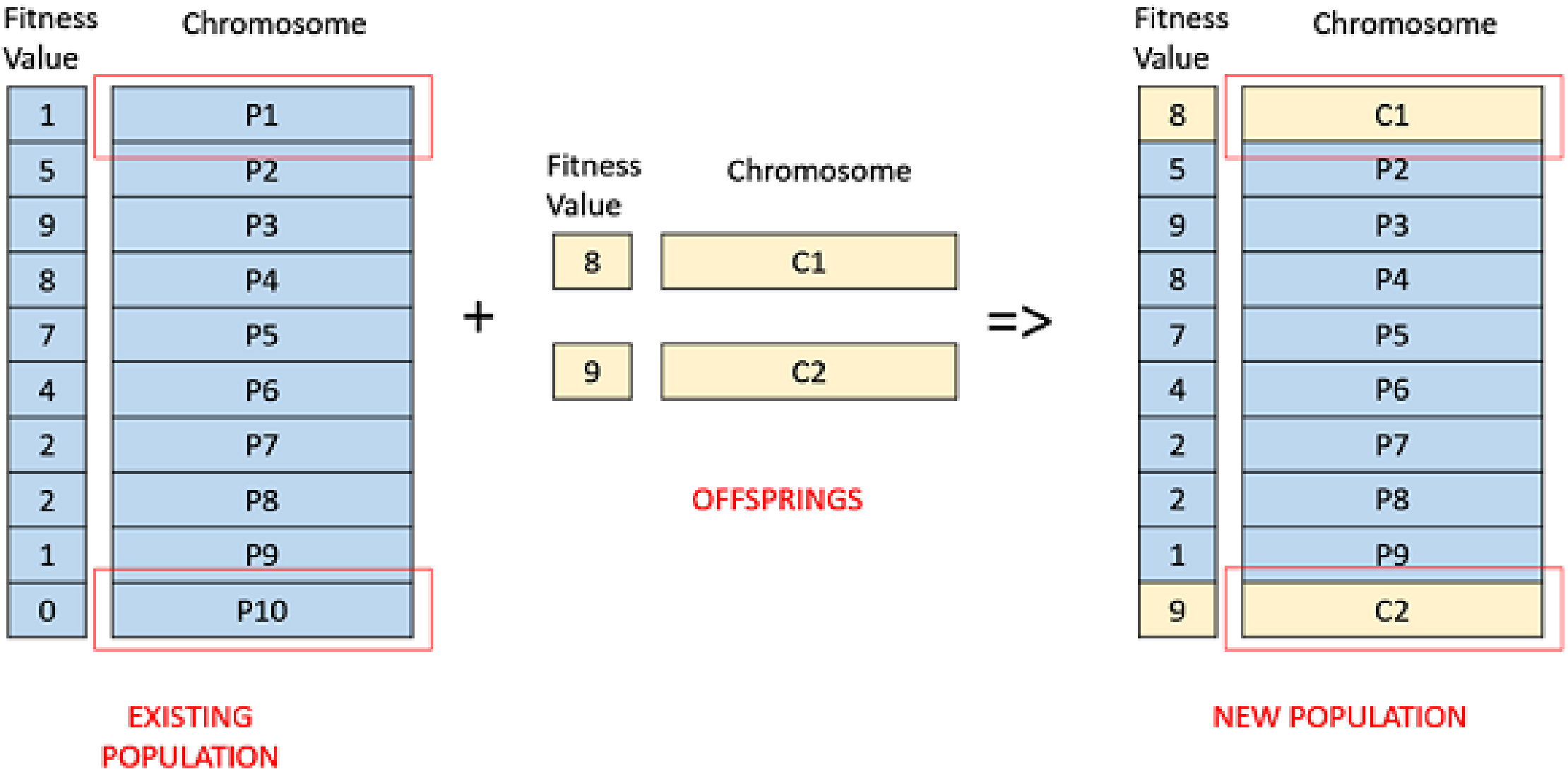


- **Genetic Algorithms - Survivor Selection**
- The Survivor Selection Policy determines which individuals are to be kicked out and which are to be kept in the next generation.
- It is crucial as it should ensure that the fitter individuals are not kicked out of the population, while at the same time diversity should be maintained in the population.
- The easiest policy is to kick random members out of the population, but such an approach frequently has convergence issues, therefore the following strategies are widely used.

- **Age Based Selection**
- In Age-Based Selection, we don’t have a notion of a fitness. It is based on the premise that each individual is allowed in the population for a finite generation where it is allowed to reproduce, after that, it is kicked out of the population no matter how good its fitness is.
- For instance, in the following example, the age is the number of generations for which the individual has been in the population.
- The oldest members of the population i.e. P4 and P7 are kicked out of the population and the ages of the rest of the members are incremented by one.


Age
Fitness Value
Chromosome
6 1 P1
8 5 P2
5 9 P3
10 8 P4
2 7 P5
3 4 P6
9 2 P7
2 2 P8
5 1 P9
4 0 P10
+
Fitness Value
Chromosome
8 C1
9 C2
OFFSPRINGS
=>
Age
Fitness Value
Chromosome
7 1 P1
9 5 P2
4 9 P3
0 8 C1
3 7 P5
4 4 P6
0 9 C2
3 2 P8
6 1 P9
5 0 P10
EXISTING POPULATION
NEW POPULATION

- **Fitness Based Selection**
- In this fitness based selection, the children tend to replace the least fit individuals in the population.
- The selection of the least fit individuals may be done using a variation of any of the selection policies described before – tournament selection, fitness proportionate selection, etc.
- For example, in the following image, the children replace the least fit individuals P1 and P10 of the population.
- It is to be noted that since P1 and P9 have the same fitness value, the decision to remove which individual from the population is arbitrary.


Fitness Value
Chromosome
1
P1
5
P2
9
P3
8
P4
7
P5
4
P6
2
P7
2
P8
1
P9
0
P10
+
Fitness Value
Chromosome
8
C1
9
C2
OFFSPRINGS
=>
Fitness Value
Chromosome
8
C1
5
P2
9
P3
8
P4
7
P5
4
P6
2
P7
2
P8
1
P9
9
C2
EXISTING POPULATION
NEW POPULATION

- **Genetic Algorithms - Termination Condition**
- The termination condition of a Genetic Algorithm is important in determining when a GA run will end.
- It has been observed that initially, the GA progresses very fast with better solutions coming in every few iterations, but this tends to saturate in the later stages where the improvements are very small.
- We usually want a termination condition such that our solution is close to the optimal, at the end of the run.
- Usually, we keep one of the following termination conditions –
- When there has been no improvement in the population for X iterations.
- When we reach an absolute number of generations.
- When the objective function value has reached a certain pre-defined value.

- For example, in a genetic algorithm we keep a counter which keeps track of the generations for which there has been no improvement in the population. Initially, we set this counter to zero.
- Each time we don't generate off-springs which are better than the individuals in the population, we increment the counter.
- However, if the fitness any of the off-springs is better, then we reset the counter to zero.
- The algorithm terminates when the counter reaches a predetermined value.
- Like other parameters of a GA, the termination condition is also highly problem specific and the GA designer should try out various options to see what suits his particular problem the best.

- **Genetic Algorithms - Application Areas**
- Genetic Algorithms are primarily used in optimization problems of various kinds, but they are frequently used in other application areas as well.
- Some of the areas in which Genetic Algorithms are frequently used. These are −

- **Optimization** – Genetic Algorithms are most commonly used in optimization problems wherein we have to maximize or minimize a given objective function value under a given set of constraints. The approach to solve Optimization problems has been highlighted throughout the tutorial.
- **Economics** – GAs are also used to characterize various economic models like the cobweb model, game theory equilibrium resolution, asset pricing, etc.
- **Neural Networks** – GAs are also used to train neural networks, particularly recurrent neural networks.
- **Parallelization** – GAs also have very good parallel capabilities, and prove to be very effective means in solving certain problems, and also provide a good area for research.
- **Image Processing** – GAs are used for various digital image processing (DIP) tasks as well like dense pixel matching.
- **Vehicle routing problems** – With multiple soft time windows, multiple depots and a heterogeneous fleet.
- **Scheduling applications** – GAs are used to solve various scheduling problems as well, particularly the time tabling problem.
- **Machine Learning** – as already discussed, genetics based machine learning (GBML) is a niche area in machine learning.

- **Robot Trajectory Generation** – GAs have been used to plan the path which a robot arm takes by moving from one point to another.
- **Parametric Design of Aircraft** – GAs have been used to design aircrafts by varying the parameters and evolving better solutions.
- **DNA Analysis** – GAs have been used to determine the structure of DNA using spectrometric data about the sample.
- **Multimodal Optimization** – GAs are obviously very good approaches for multimodal optimization in which we have to find multiple optimum solutions.
- **Traveling salesman problem and its applications** – GAs have been used to solve the TSP, which is a well-known combinatorial problem using novel crossover and packing strategies.

Results

The number of iterations to run depends on the nature of the problem. In this tutorial, we run 500 iterations.

```
Iteration 490: Best Cost = 0.13436917704918208
Iteration 491: Best Cost = 0.13436917704918208
Iteration 492: Best Cost = 0.13436917704918208
Iteration 493: Best Cost = 0.13436917704918208
Iteration 494: Best Cost = 0.13436917704918208
Iteration 495: Best Cost = 0.13436917704918208
Iteration 496: Best Cost = 0.13436917704918208
Iteration 497: Best Cost = 0.13436917704918208
Iteration 498: Best Cost = 0.13436917704918208
Iteration 499: Best Cost = 0.13436917704918208
Iteration 500: Best Cost = 0.13436917704918208
<function matplotlib.pyplot.show>
```

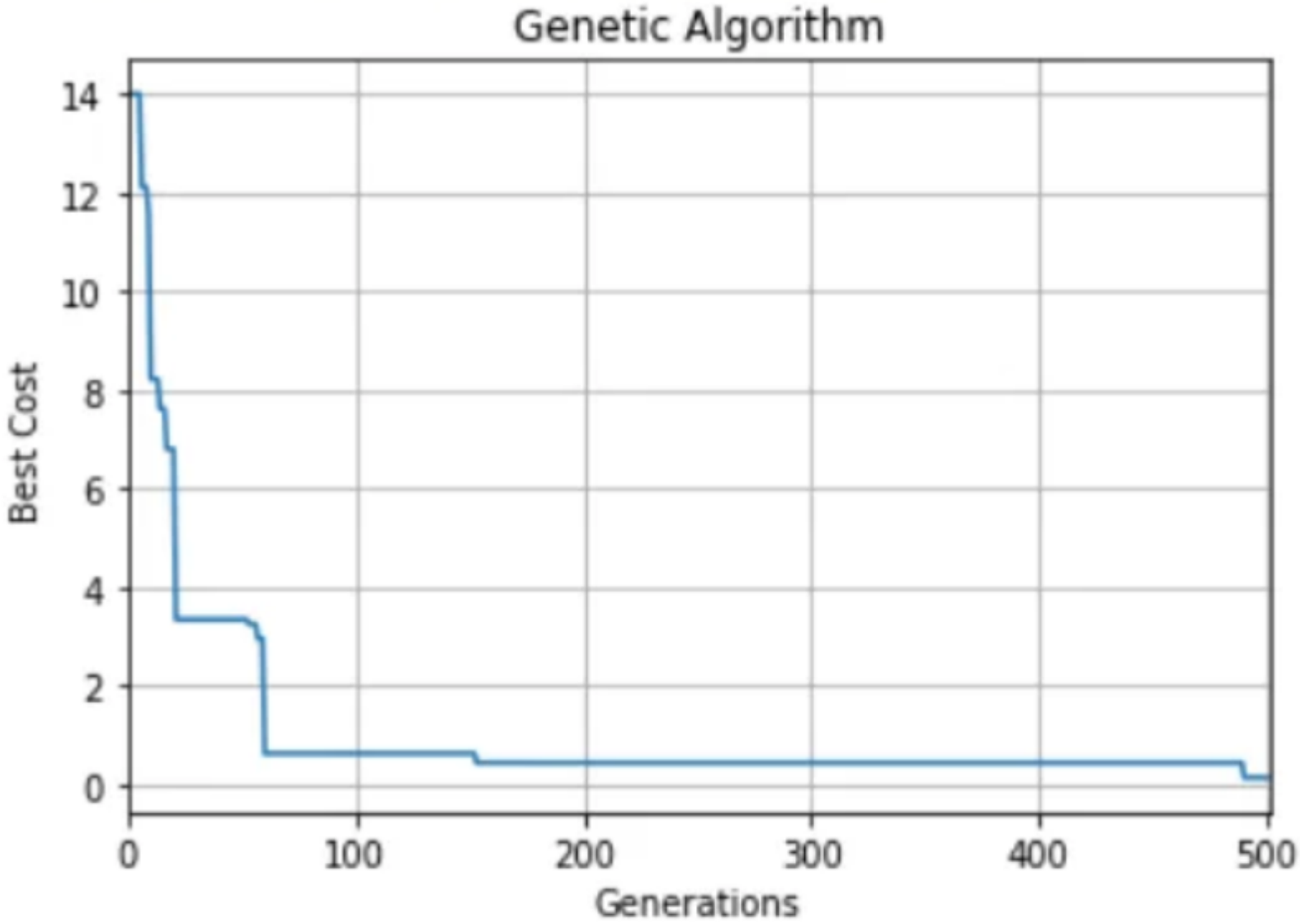


Figure 8: Generating results of our genetic algorithm (GA) implementation.

We can see how the cost reduces at every iteration, and at approximately 490 iterations, the cost reduces to 0.134 and remains the same throughout the rest of the 10 iterations. Consequently, giving us our optimal solution.

HINDUSTAN
INSTITUTE OF TECHNOLOGY & SCIENCE
(DEEMED TO BE UNIVERSITY)

EAL51003 – ARTIFICIAL INTELLIGENCE
B.Tech[AIML] – III Semester

K.Kowsalya
Assistant Professor (SS)
School of Computing Sciences,
Department of Computer Science and Engineering

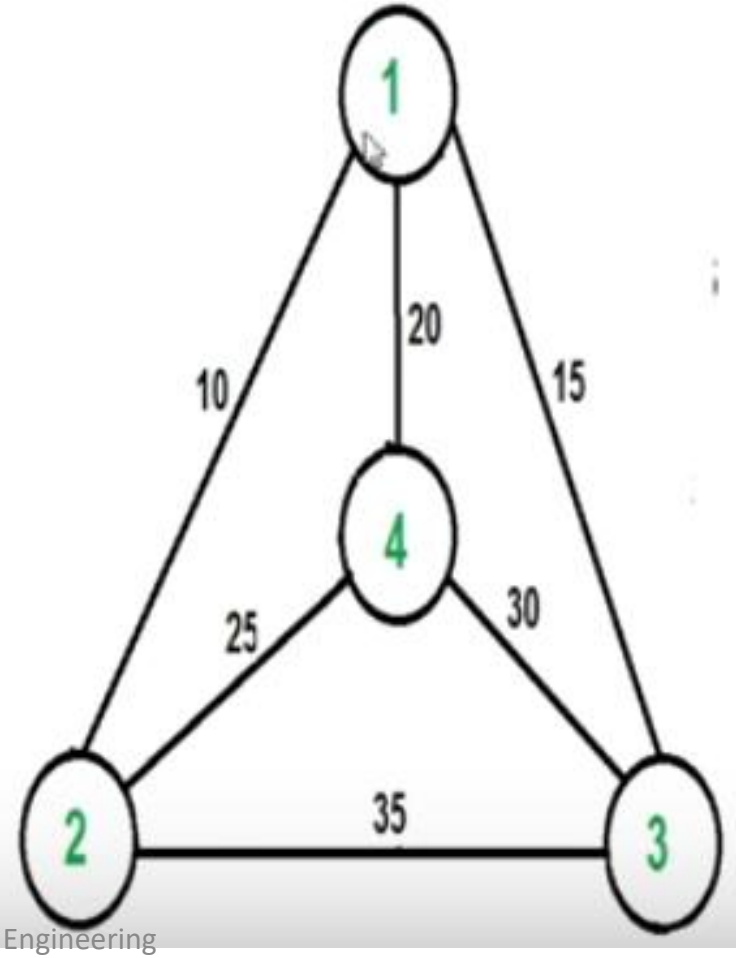
UNIT-II
TSP 30 CITIES USING ANT ALGORITHM

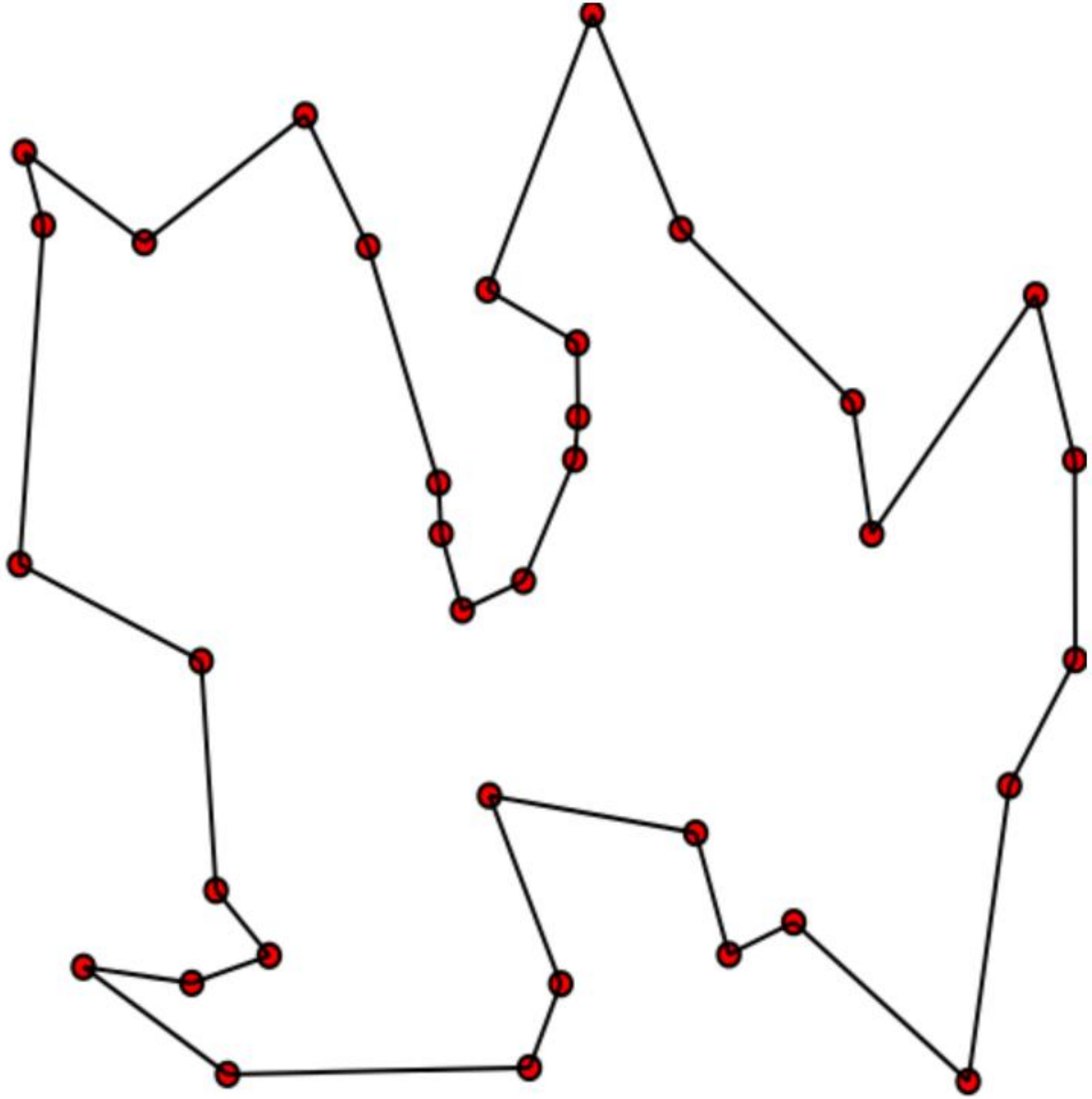
Traveling Salesman Problem

In the traveling salesman problem, a set of cities is given and the distance between each of them is known.

The goal is to find the shortest tour that allows each city to be visited once and only once.

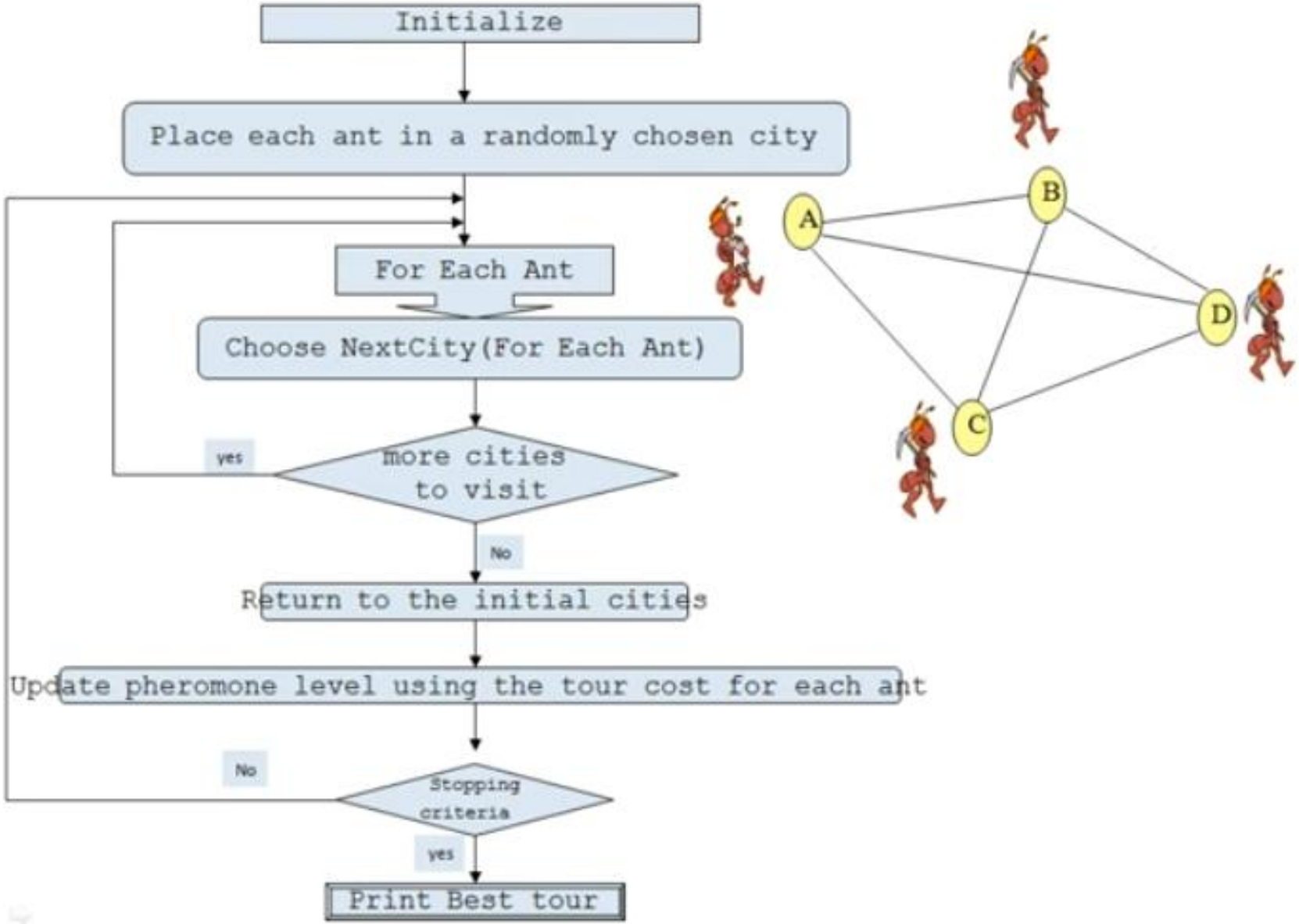
In more formal terms, the goal is to find a Hamiltonian tour of minimal length on a fully connected graph.





TSP-30 CITIES

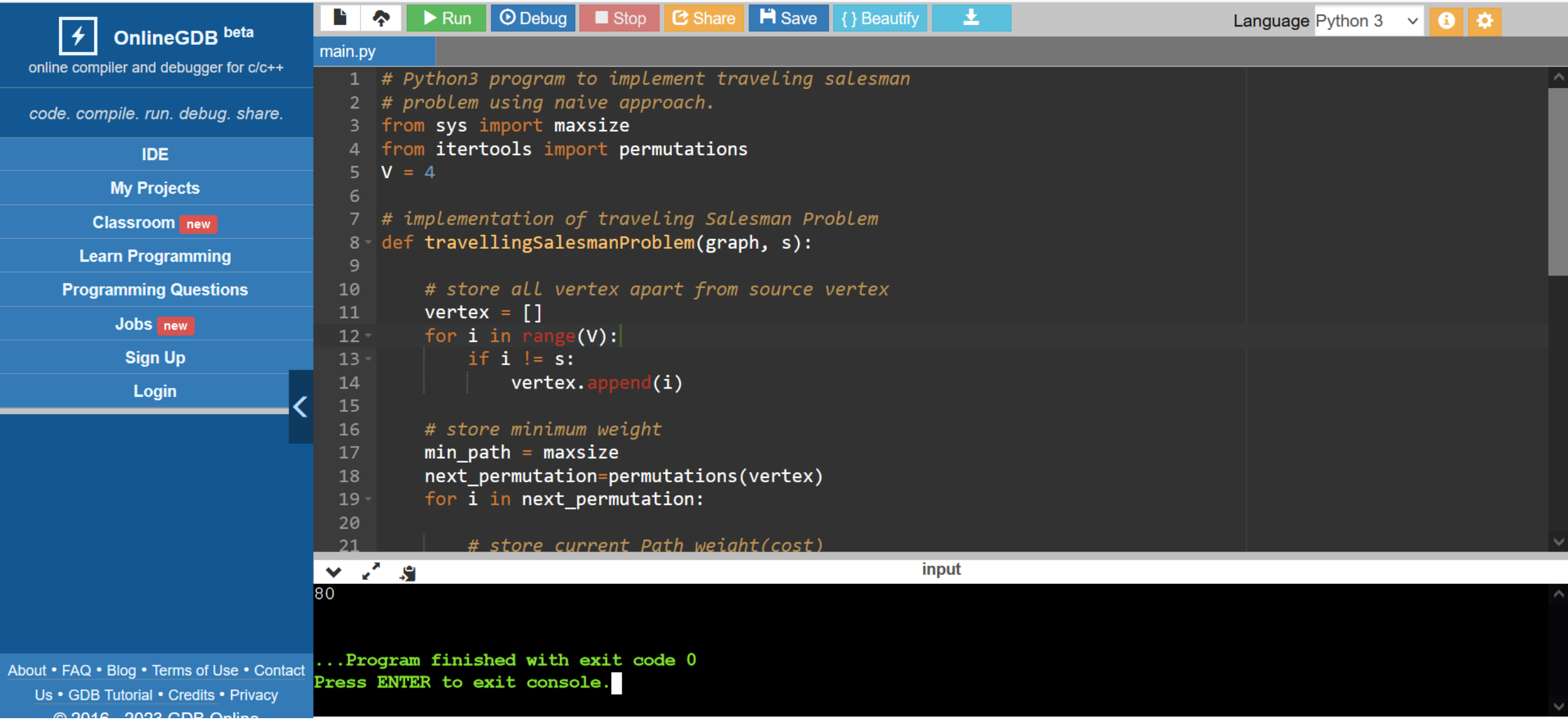
Ant Systems Algorithm for TSP



```python
# Python3 program to implement traveling salesman
# problem using naive approach.
from sys import maxsize
from itertools import permutations
V = 4

# implementation of traveling Salesman Problem
def travellingSalesmanProblem(graph, s):

    # store all vertex apart from source vertex
    vertex = []
    for i in range(V):
        if i != s:
            vertex.append(i)

    # store minimum weight Hamiltonian Cycle
    min_path = maxsize
    next_permutation=permutations(vertex)
    for i in next_permutation:

        # store current Path weight(cost)
        current_pathweight = 0

        # compute current path weight
        k = s
        for j in i:
            current_pathweight += graph[k][j]
            k = j
        current_pathweight += graph[k][s]

        # update minimum
        min_path = min(min_path, current_pathweight)

    return min_path


# Driver Code
if __name__ == "__main__":

    # matrix representation of graph
    graph = [[0, 10, 15, 20], [10, 0, 35, 25],
             [15, 35, 0, 30], [20, 25, 30, 0]]
    s = 0
    print(travellingSalesmanProblem(graph, s))
```

Output

```
80
```

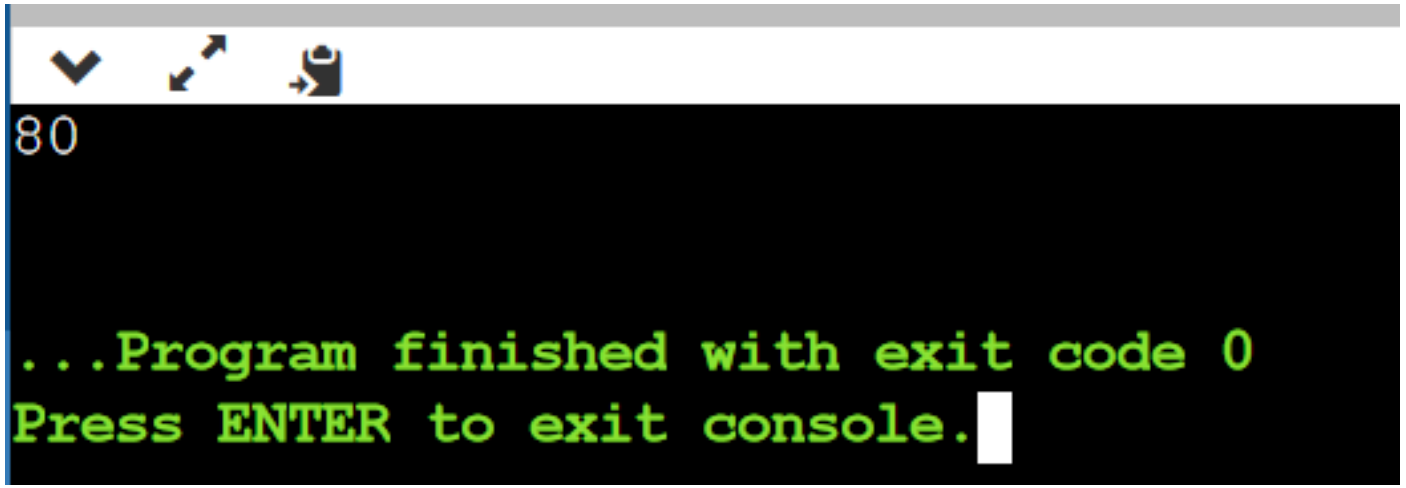

OnlineGDB beta
online compiler and debugger for c/c++
code. compile. run. debug. share.
IDE
My Projects
Classroom new
Learn Programming
Programming Questions
Jobs new
Sign Up
Login
Run
Debug
Stop
Share
Save
{ } Beautify
Language Python 3
main.py
# Python3 program to implement traveling salesman
# problem using naive approach.
from sys import maxsize
from itertools import permutations
V = 4
# implementation of traveling Salesman Problem
def travellingSalesmanProblem(graph, s):
# store all vertex apart from source vertex
vertex = []
for i in range(V):
if i != s:
vertex.append(i)
# store minimum weight
min_path = maxsize
next_permutation=permutations(vertex)
for i in next_permutation:
# store current Path weight(cost)
input
80
...Program finished with exit code 0
Press ENTER to exit console.
About • FAQ • Blog • Terms of Use • Contact Us • GDB Tutorial • Credits • Privacy

main.py

```python
# Python3 program to implement traveling salesman
# problem using naive approach.
from sys import maxsize
from itertools import permutations
V = 4

# implementation of traveling Salesman Problem
def travellingSalesmanProblem(graph, s):

    # store all vertex apart from source vertex
    vertex = []
    for i in range(V):
        if i != s:
            vertex.append(i)

    # store minimum weight
    min_path = maxsize
    next_permutation=permutations(vertex)
    for i in next_permutation:

```

```
        # store current Path weight(cost)
        current_pathweight = 0

        # compute current path weight
        k = s
        for j in i:
            current_pathweight += graph[k][j]
            k = j
        current_pathweight += graph[k][s]

        # update minimum
        min_path = min(min_path, current_pathweight)

    return min_path


# Driver Code
if __name__ == "__main__":
```

```
38 if __name__ == "__main__":
39
40     # matrix representation of graph
41     graph = [[0, 10, 15, 20], [10, 0, 35, 25],
42              [15, 35, 0, 30], [20, 25, 30, 0]]
43     s = 0
44     print(travellingSalesmanProblem(graph, s))
45
```



- The **Hamiltonian cycle problem** is to find if there exists a tour that visits every city exactly once.
- Here we know that Hamiltonian Tour exists (because the graph is complete) and in fact, many such tours exist, the problem is to find a minimum weight Hamiltonian Cycle.

  For example, consider the graph shown in the figure on the right side.
- **A TSP tour in the graph is 1-2-4-3-1. The cost of the tour is 10+25+30+15 which is 80.**

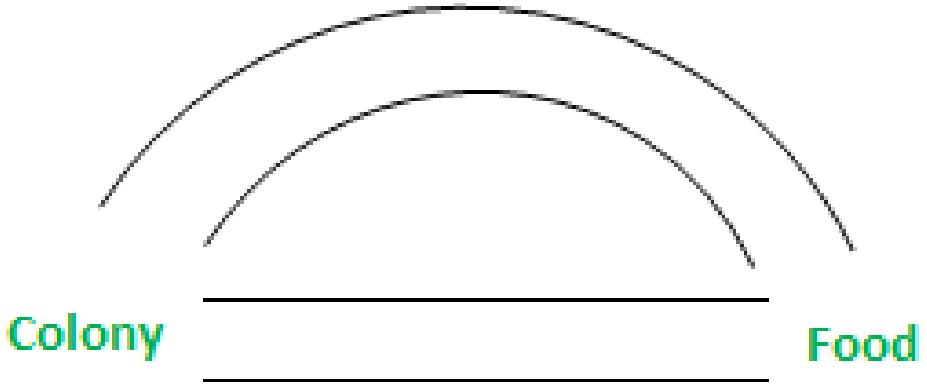
Ant Colony Optimization

- **Metaheuristic** has been derived from two Greek words, namely, **Meta** meaning **one level above** and **heuriskein** meaning **to find**.
- Algorithms such as the Particle Swarm Optimization (PSO) and **Ant Colony Optimization (ACO) are examples of swarm intelligence and metaheuristics.**
- The **goal of swarm intelligence** is to design intelligent multi-agent systems by taking inspiration from the collective behaviour of social insects such as ants, termites, bees, wasps, and other animal societies such as flocks of birds.

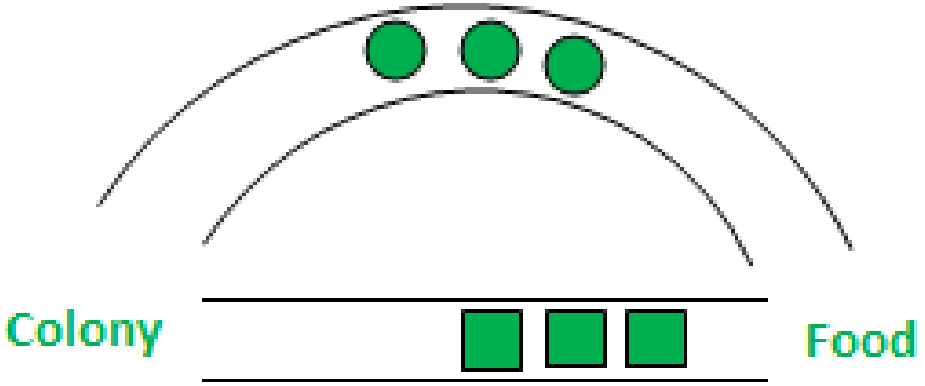
- Ants are eusocial insects that prefer community survival and sustaining rather than as individual species.
- **They communicate with each other using sound, touch and pheromone. Pheromones** are organic chemical compounds secreted by the ants that trigger a social response in members of same species.
- These are chemicals capable of acting like hormones outside the body of the secreting individual, to impact the behaviour of the receiving individuals.
- Since most ants live on the ground, they use the soil surface to leave pheromone trails that may be followed (smelled) by other ants.

- Ants live in community nests and the underlying principle of ACO is to observe the movement of the ants from their nests in order to search for food in the shortest possible path.
- Initially, ants start to move randomly in search of food around their nests.
- This randomized search opens up multiple routes from the nest to the food source. Now, based on the quality and quantity of the food, ants carry a portion of the food back with necessary pheromone concentration on its return path.

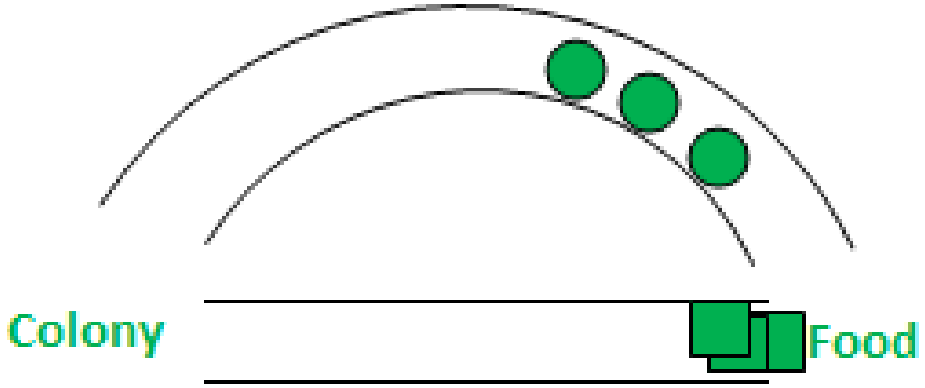
- Depending on these pheromone trials, the probability of selection of a specific path by the following ants would be a guiding factor to the food source.
- Evidently, this probability is based on the concentration as well as the rate of evaporation of pheromone.
- It can also be observed that since the evaporation rate of pheromone is also a deciding factor, the length of each path can easily be accounted for.



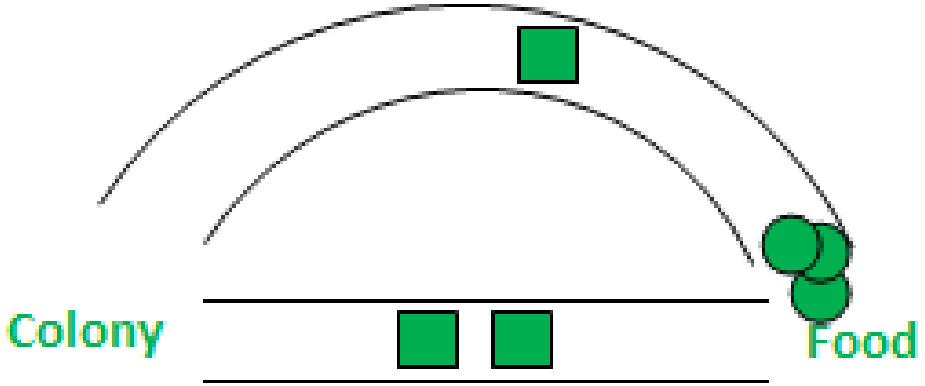
1



2

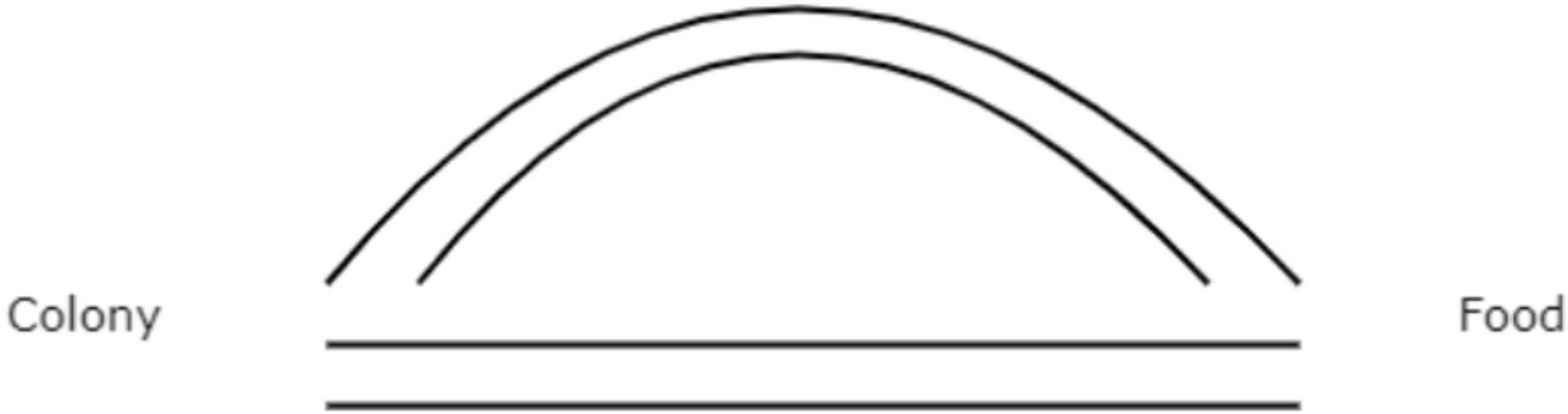


3



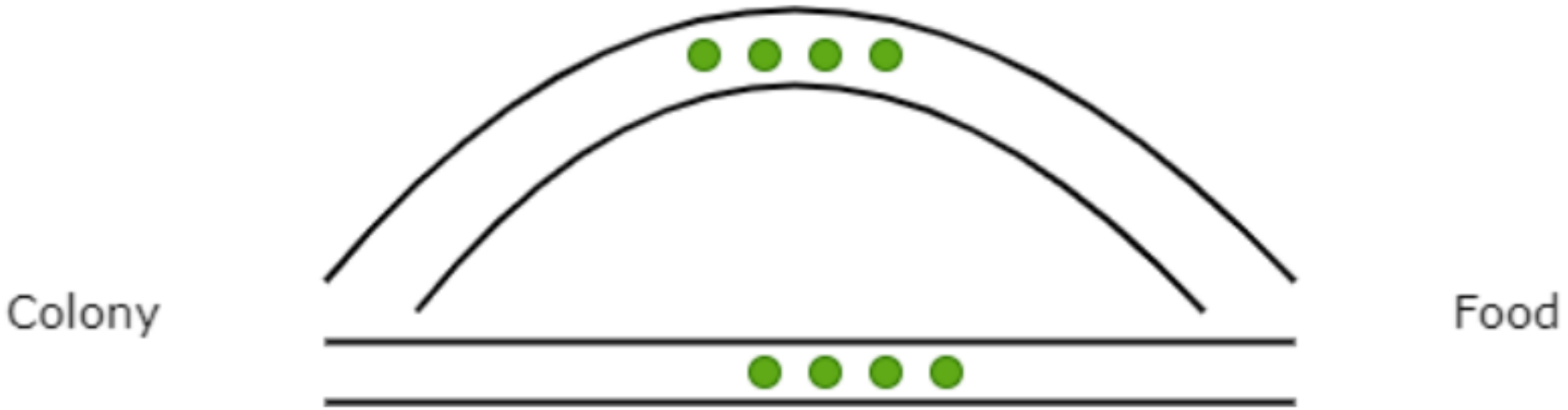
4

Stage 1:



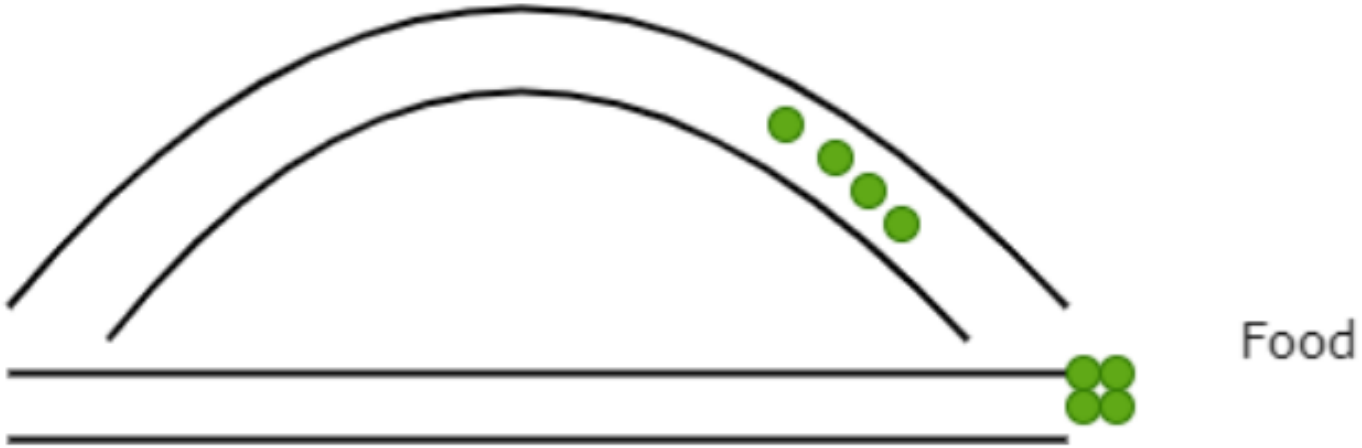
In this stage, there is no pheromone in the path, and there are empty paths from food to the ant colony.

Stage2:



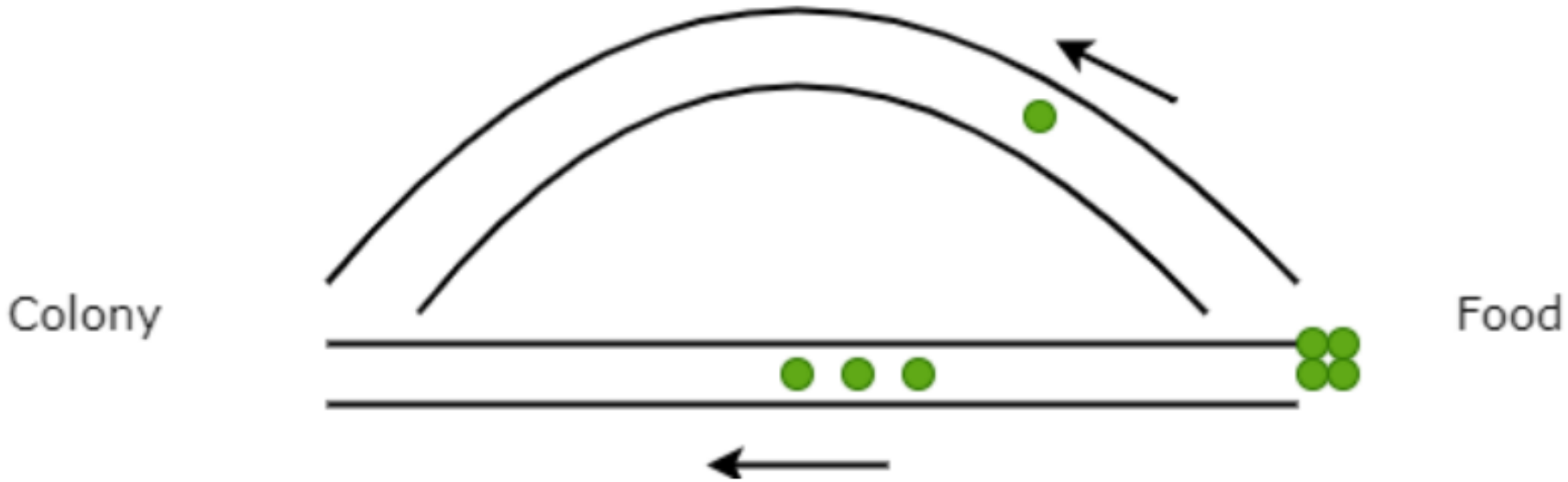
In this stage, ants are divided into two groups following two different paths with a probability of 0.5. So we have four ants on the longer path and four on the shorter path.

Stage 3:



Now, the ants which follow the shorter path will react to the food first, and then the pheromone concentration will be higher on this path as more ants from the colony will follow the shorter path.

Stage 4:



Now more ants will return from the shortest path, and the concentration of pheromones will be higher. Also, the rate of evaporation from the longer path will be higher as fewer ants are using that path. Now more ants from the colony will use the shortest path.

Algorithm Design

- We can consider the ant colony and food source as the node or vertex of the graph and the path as the edges to these vertices. Now the pheromone concentration can be assumed as the weight associated with each path.
- Suppose there are only two paths which are P_1 and P_2.
- C_1 and C_2 are the weight or the pheromone concentration along the path, respectively.
- So we can represent it as graph G(V, E) where V represents the Vertex and E represents the Edge of the graph.
- Initially, for the i^{th} path, the probability of choosing is:

$$Pi = \frac{Ci}{C1+C2}; \; where\, i = 1, 2$$

- f $C_1 > C_2$, then the probability of choosing path 1 is more than path 2. If $C_1 < C_2$, then Path 2 will be more favorable.

For the return path, the length of the path and the rate of evaporation of the pheromone are the two factors.

1. Concentration of pheromone according to the length of the path:

$$Ci = Ci + \frac{K}{Li}$$

Where L_i is the length of the path and K is the constant depending upon the length of the path. If the path is shorter, concentration will be added more to the existing pheromone concentration.

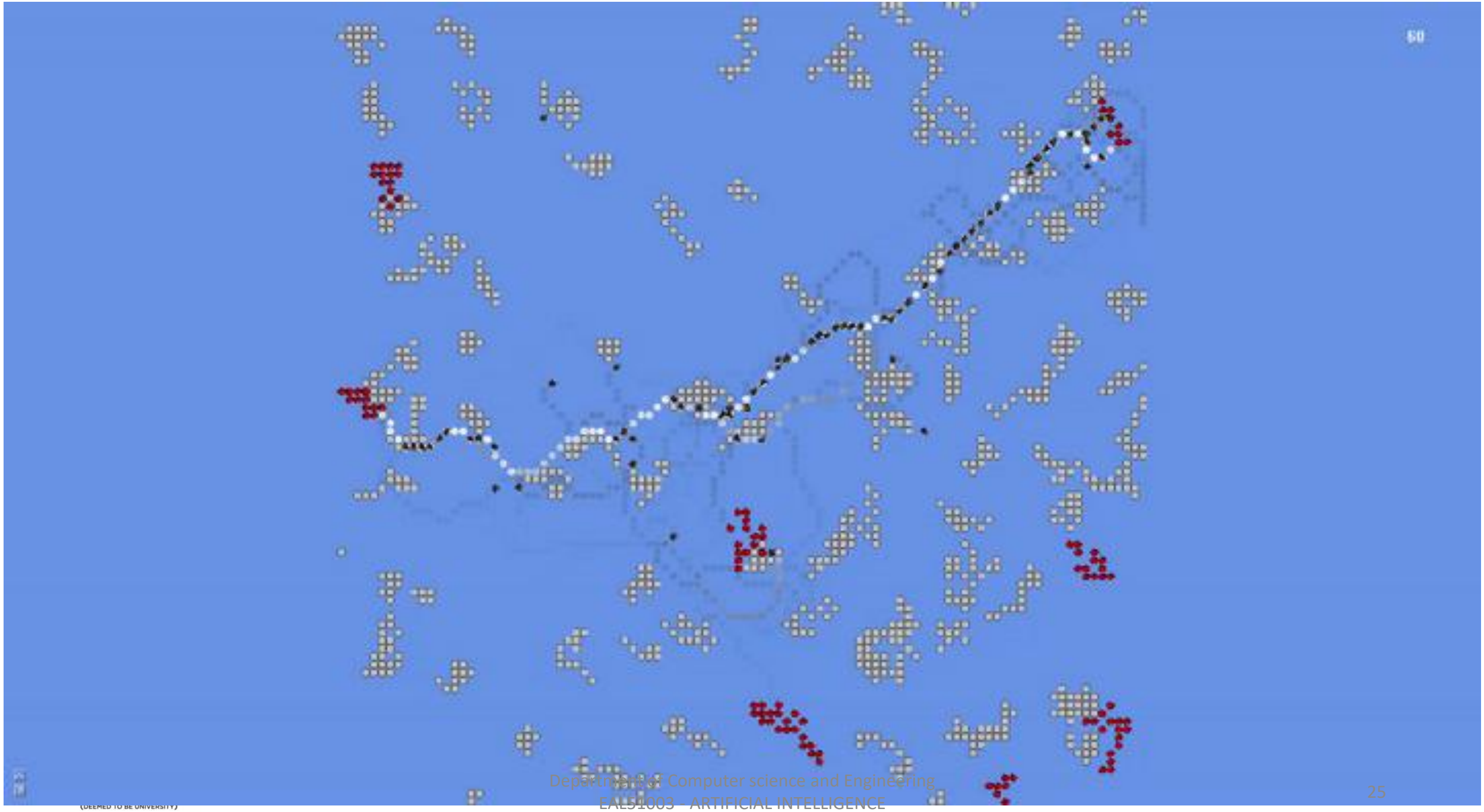
2. Change in concentration according to the rate of evaporation:

$$Ci = (1 - v) * Ci$$

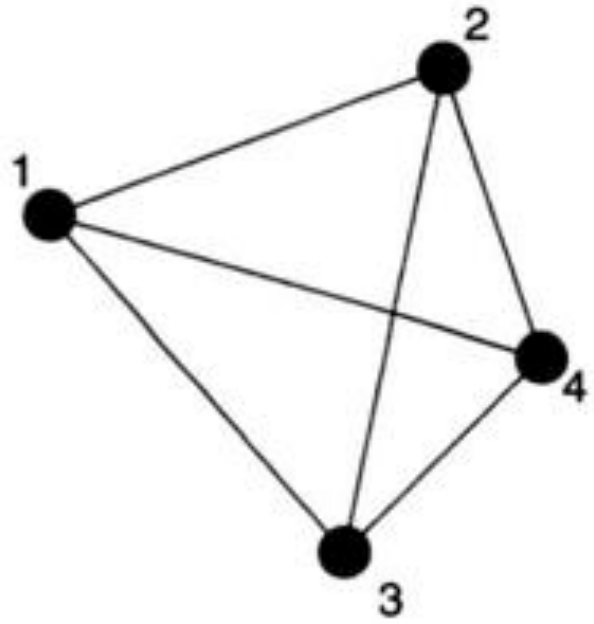
Here parameter v varies from 0 to 1. If v is higher, then the concentration will be less.

Pseudo Code:

```
Procedure ACO:
    Initialize the necessary parameters and pheromone concentration;
    while not termination do:
        Generate initial ant population;
        Calculate the fitness values for each ant of the colony;
        Find optimal solution using selection methods;
        Update pheromone concentration;
    end while
end procedure
```


60

Example of Fully Connected Bidirectional graph V with edge set E



Graph with Vertex set V = {1,2,3,4}

Edge set E ={{1,2},{1,4},{1,3},{2,3},{2,4},{3,4}}

HINDUSTAN
INSTITUTE OF TECHNOLOGY & SCIENCE
(DEEMED TO BE UNIVERSITY)

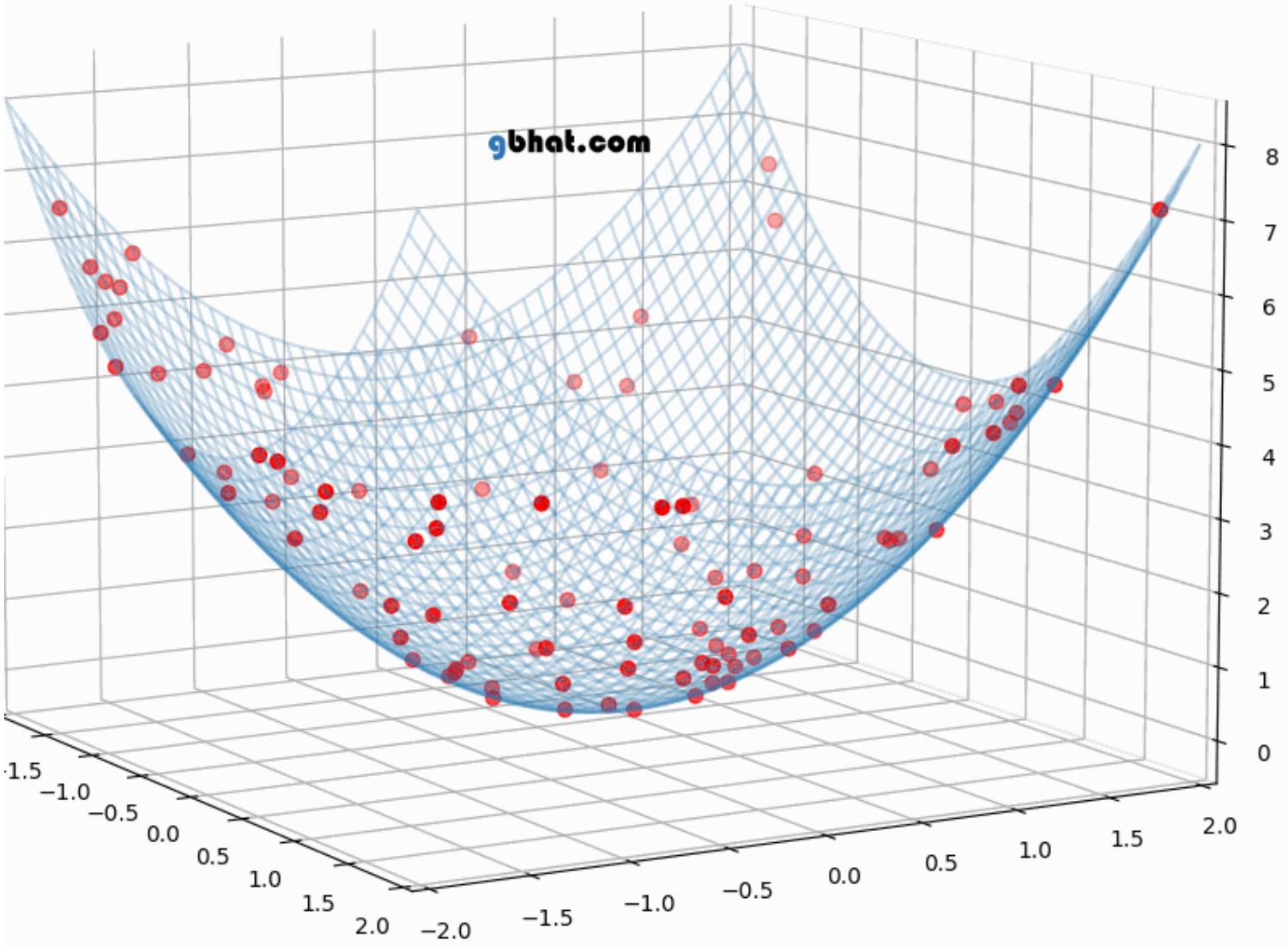
EAL51003 – ARTIFICIAL INTELLIGENCE
B.Tech[AIML] – III Semester

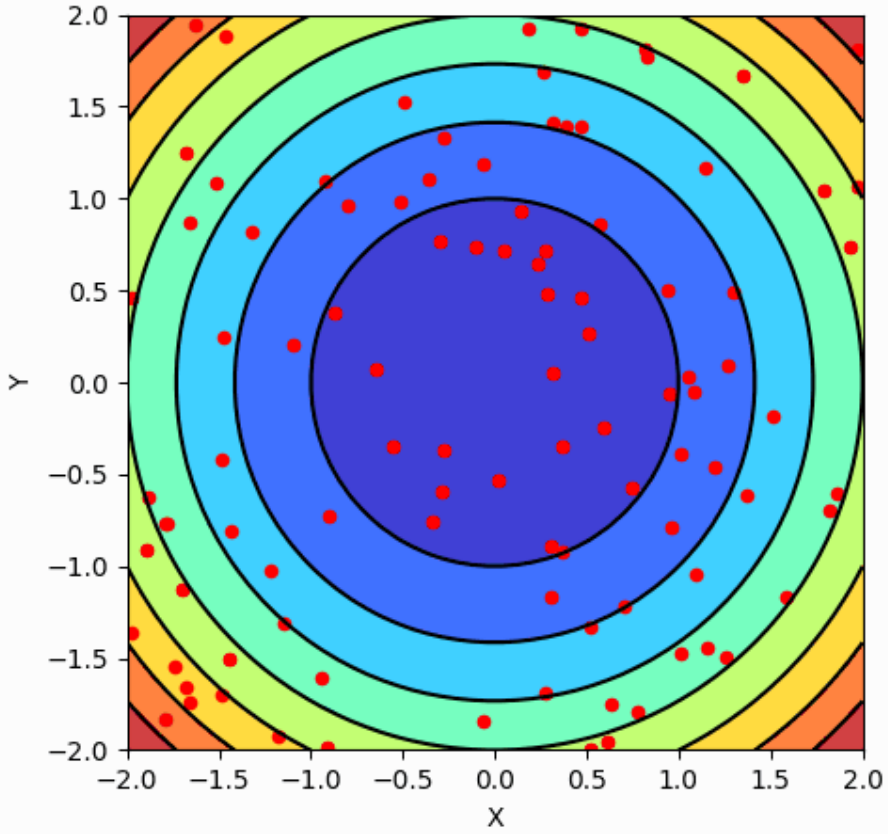
K.Kowsalya
Assistant Professor (SS)
School of Computing Sciences,
Department of Computer Science and Engineering

UNIT-II
PARTICLE SWARM OPTIMIZATION

- The Particle Swarm Optimization (PSO) algorithm is a computational technique inspired by the collective behavior of natural organisms, such as birds or fish, that move together to achieve a common goal.
- In PSO, a group of particles (representing potential solutions) navigates through a problem's solution space to find the best possible solution. Each particle adjusts its position based on its own best-known solution (personal best) and the best solution discovered by the entire group (global best).

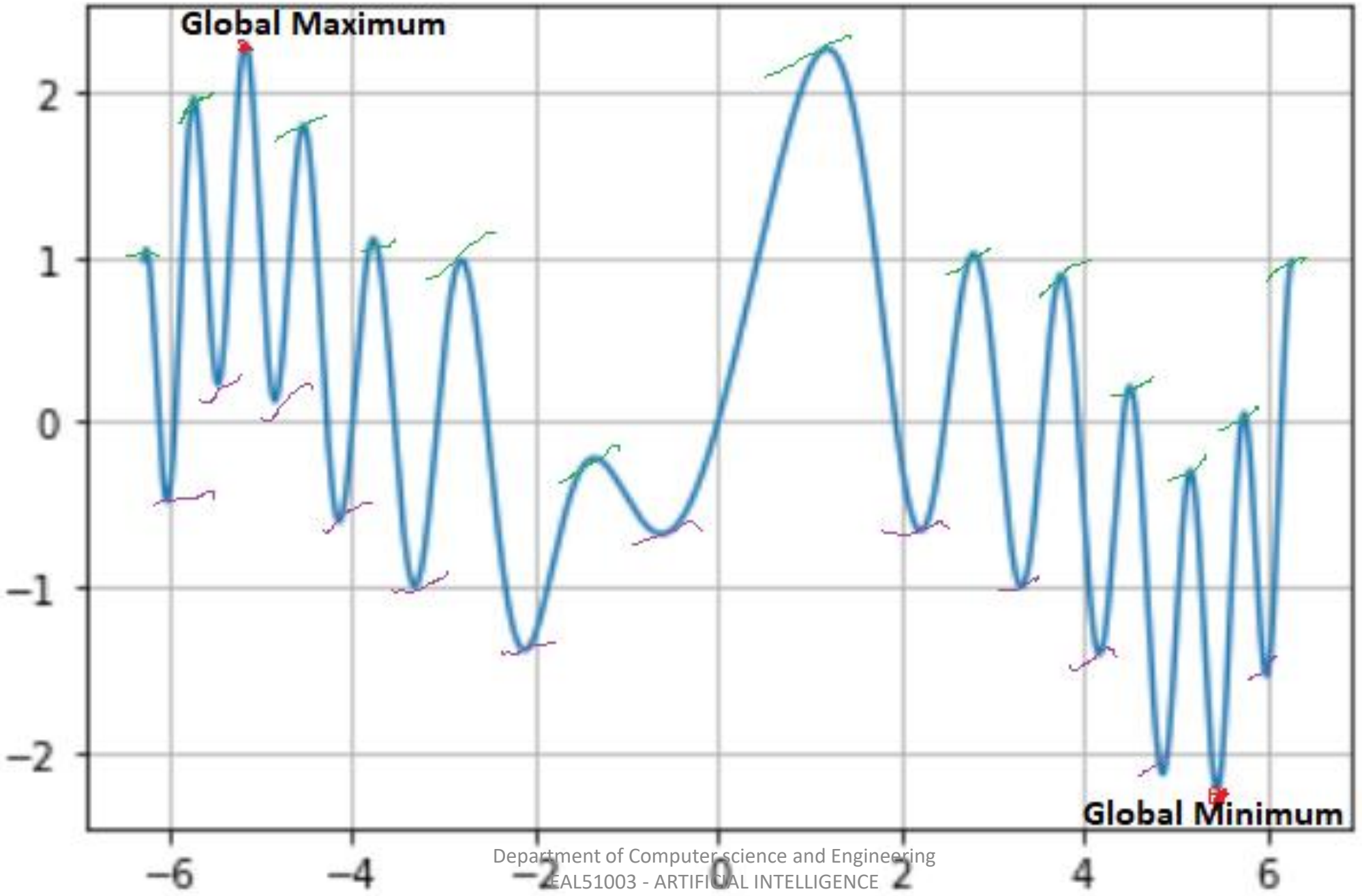
- This collaborative movement enables particles to converge toward optimal solutions over iterations.
- PSO is widely used for optimization problems in various fields, leveraging the power of collective intelligence to explore complex solution spaces and find optimal outcomes efficiently.


gbhat.com
8
7
6
5
4
3
2
1
0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
Y
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
X

- **The Problem of Optimization**
- The concept of swarm intelligence inspired the POS. Here we are speaking about finding the optimal solution in a high-dimensional solution space. It talks about Maximizing earns or minimizing losses. So, we are looking to maximize or minimize a function to find the optimum solution.

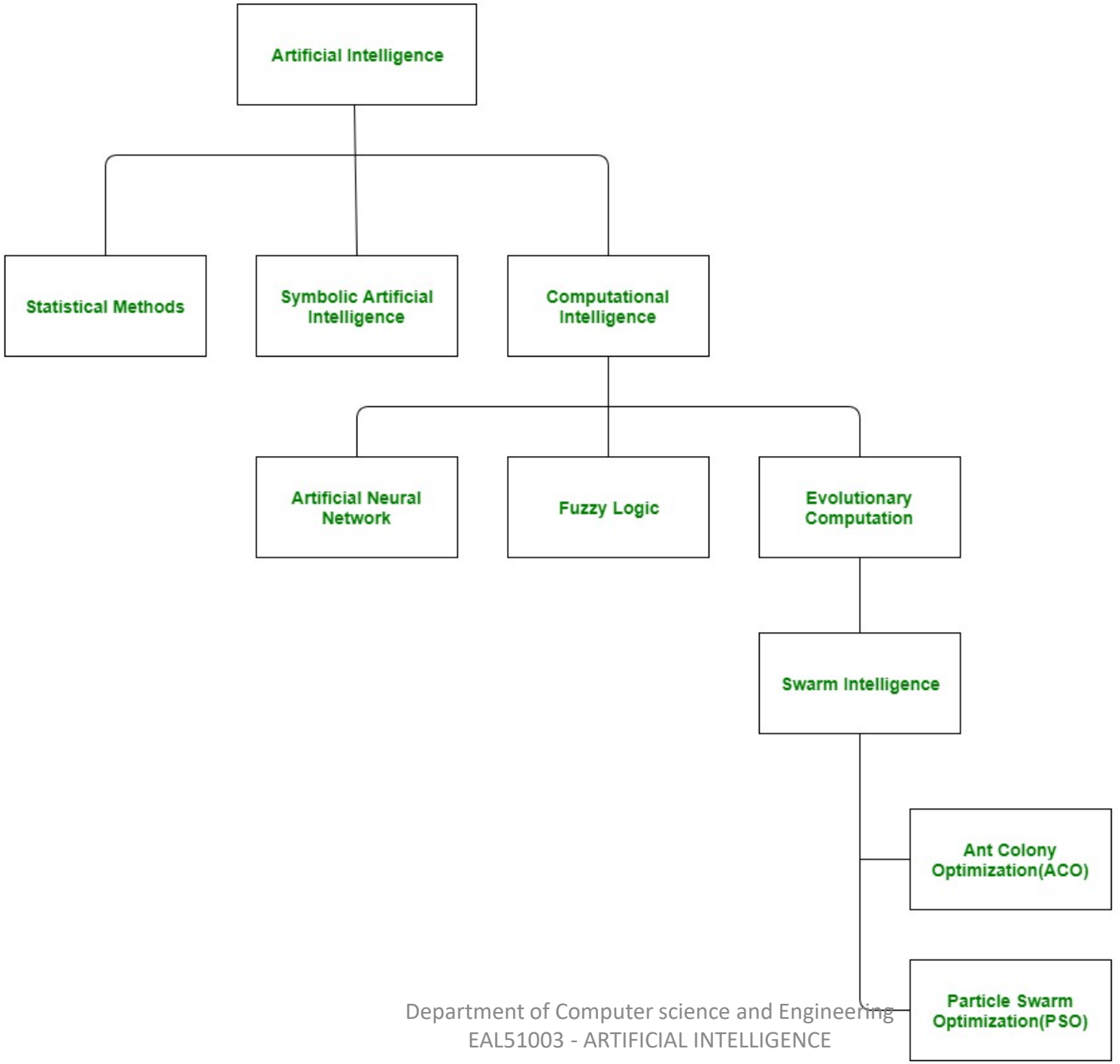
- A function can have multiple local maximum and minimum. But, there can be only one global maximum as well as a minimum. If your function is very complex, then finding the global maximum can be a very daunting task. PSO tries to capture the global maximum or minimum.
- Even though it cannot capture the exact global maximum/minimum, it goes very close to it. It is the reason we called PSO a heuristic model.
- **Finding of Global Maximum or Minimum**
- Let me give you an example of why the finding of global maximum/minimum is problematic.
- Check the below function : $y=f(x)=\sin x+\sin x^2+\sin x\cos x$


Global Maximum
Global Minimum
2
1
0
−1
−2
−6
−4
−2
0
2
4
6

- ***These techniques are designed basis two categories*:**
- The first study includes how biological phenomena can be studies using computation.
- The second one shows how biological phenomena can help understand computation problems. While *studying the PSO Technique, we deal with in the second category*.
- ***There are three approaches of the artificial intelligence:***
- Statistical Methods
- Symbolic Artificial Intelligence
- Computational Intelligence

- ***Computational Intelligence can be implemented using either of the three methods:***
- Artificial Neural Network
- Fuzzy Logic
- Evolutionary Computation

- Swarm Intelligence is a branch of Artificial Intelligence where we observe nature and try to learn how different biological phenomena can be imitated in a computer system to optimize the scheduling algorithms.
- In swarm intelligence, we focus on the collective behavior of simple organisms and their interaction with the environment.


Artificial Intelligence
Statistical Methods
Symbolic Artificial Intelligence
Computational Intelligence
Artificial Neural Network
Fuzzy Logic
Evolutionary Computation
Swarm Intelligence
Ant Colony Optimization(ACO)
Particle Swarm Optimization(PSO)

- ***There are two types of Optimization algorithms in Swarm Intelligence:***
- The first one is *Ant Colony Optimization(ACO)*. Here the algorithm is based on the collective behavior of ants in their colony.
- The second technique is *Particle Swarm Optimization(PSO).*
- In PSO, the focus in on a group of birds. This group of birds is referred to as a '*swarm*'.
- Let's try to understand the Particle Swarm Optimization from the following scenario.

- **Example:** Suppose there is a swarm (a group of birds). Now, all the birds are hungry and are searching for food.
- These hungry birds can be correlated with the tasks in a computation system which are hungry for resources.
- Now, in the locality of these birds, there is only one food particle.
- This food particle can be correlated with a resource. As we know, tasks are many, resources are limited.
- So this has become a similar condition as in a certain computation environment.
- Now, the birds don't know where the food particle is hidden or located. In such a scenario, how the algorithm to find the food particle should be designed.

- If every bird will try to find the food on its own, it may cause havoc and may consume a large amount of time.
- Thus on careful observation of this swarm, it was realized that though the birds don’t know where the food particle is located, they do know their distance from it.
- Thus the best approach to finding that food particle is to follow the birds which are nearest to the food particle.
- This behavior of birds is simulated in the computation environment and the algorithm so designed is termed as Particle Swarm Optimization Algorithm.

- Particle Swarm Optimization (PSO) is a powerful meta-heuristic optimization algorithm and inspired by swarm behavior observed in nature such as fish and bird schooling.
- PSO is a Simulation of a simplified social system. The original intent of PSO algorithm was to graphically simulate the graceful but unpredictable choreography of a bird flock.
- In nature, any of the bird's observable vicinity is limited to some range. However, having more than one birds allows all the birds in a swarm to be aware of the larger surface of a fitness function.

- Let’s mathematically model, the above-mentioned principles to make the swarm find the global minima of a fitness function

- **Mathematical model**
- Each particle in particle swarm optimization has an associated position, velocity, fitness value.
- Each particle keeps track of the particle_bestFitness_value particle_bestFitness_position.
- A record of global_bestFitness_position and global_bestFitness_value is maintained.

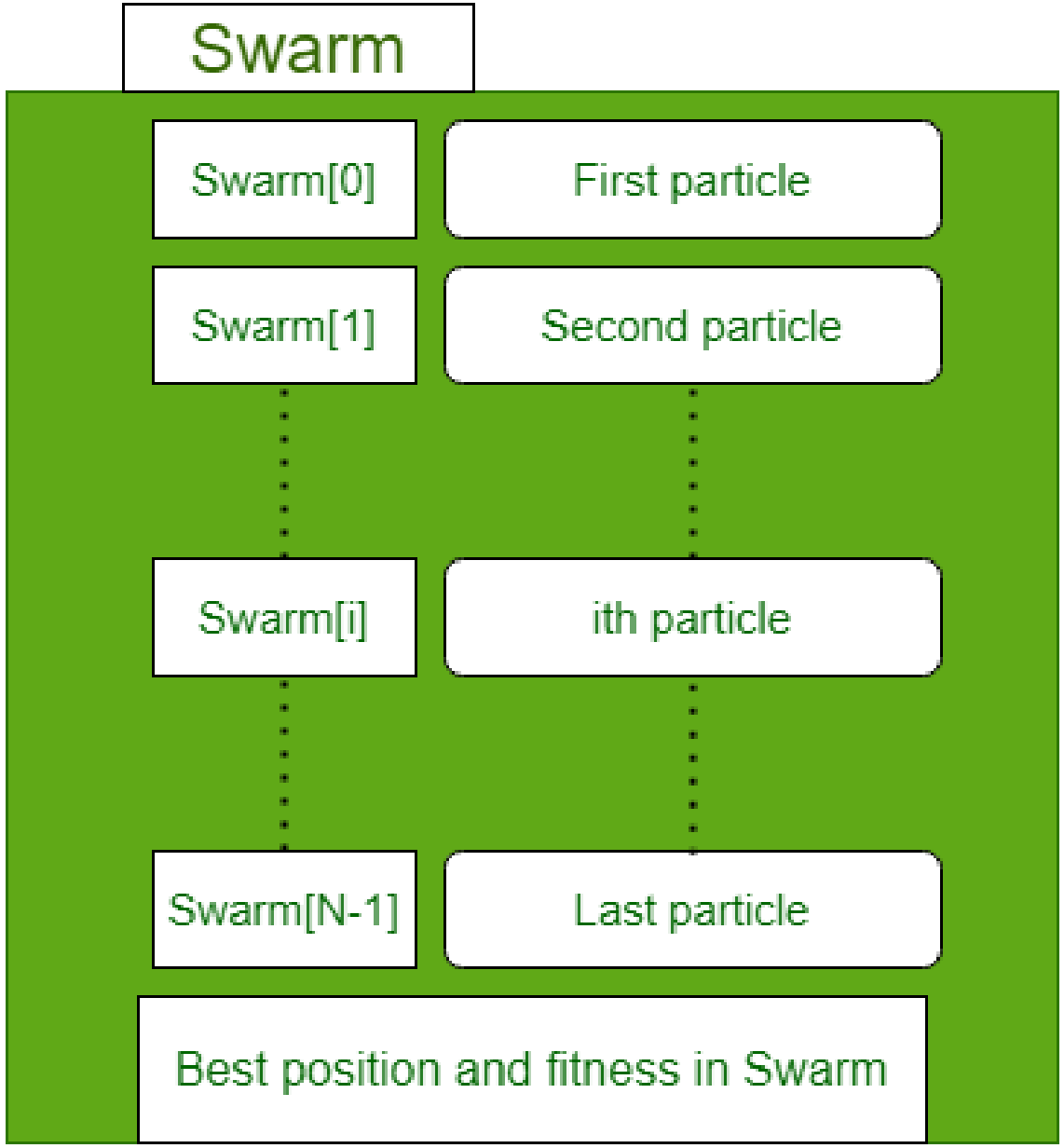


Fig 1: Data structure to store Swarm population

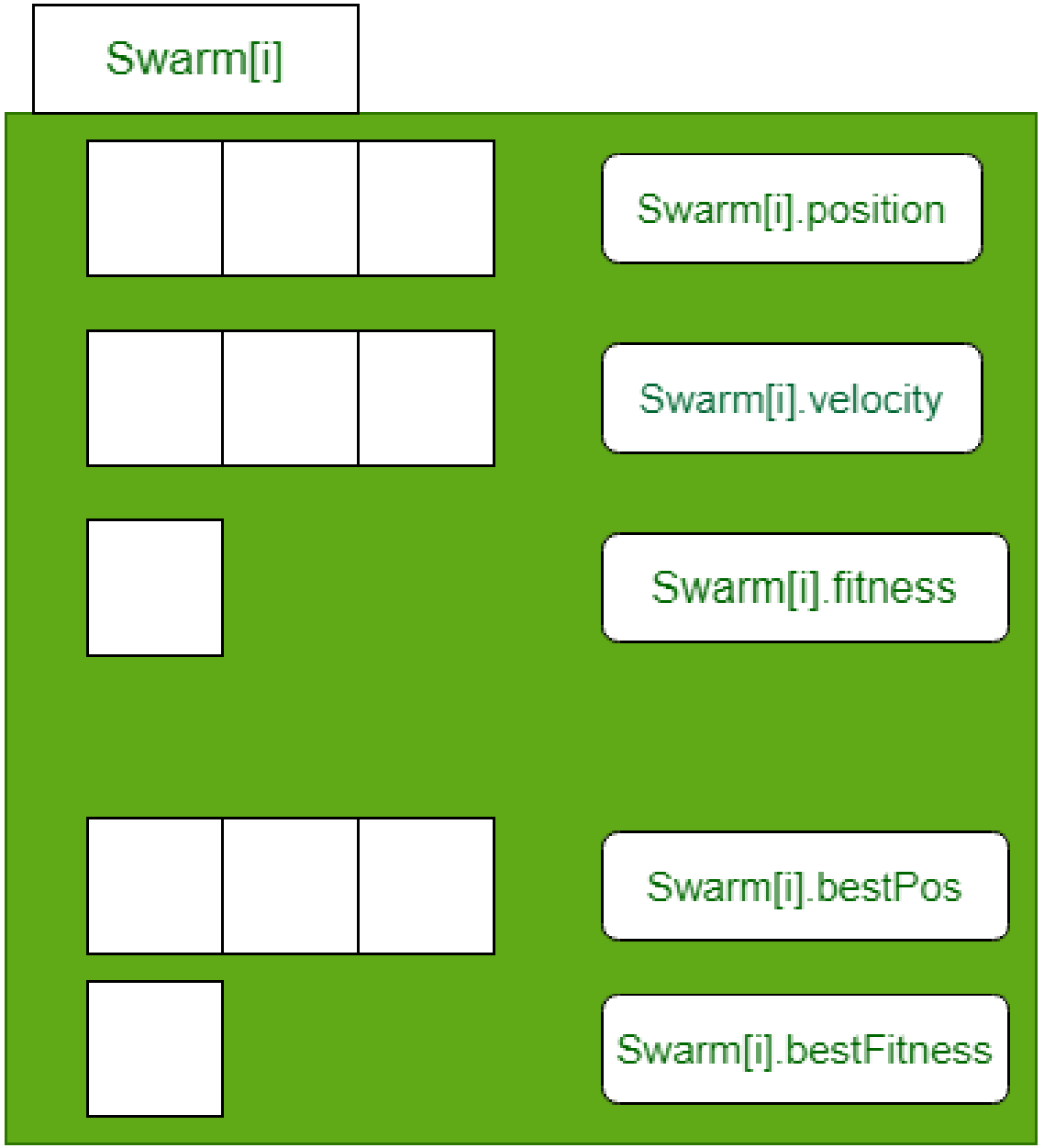


Fig 2: Data structure to store ith particle of Swarm

Algorithm

- Parameters of problem:
- Number of dimensions (d)
- Lower bound (minx)
- Upper bound (maxx)
- Hyperparameters of the algorithm:
- Number of particles (N)
- Maximum number of iterations (max_iter)
- Inertia (w)
- Cognition of particle (C1)
- Social influence of swarm (C2)

Pseudocode

```
for each particle i = 1, ..., S do
  Initialize the particle's position with a uniformly distributed random vector: x_i ~ U(b_lo, b_up)
  Initialize the particle's best known position to its initial position: p_i ← x_i
  if f(p_i) < f(g) then
      update the swarm's best known  position: g ← p_i
  Initialize the particle's velocity: v_i ~ U(-|b_up-b_lo|, |b_up-b_lo|)
while a termination criterion is not met do:
  for each particle i = 1, ..., S do
    for each dimension d = 1, ..., n do
      Pick random numbers: r_p, r_g ~ U(0,1)
      Update the particle's velocity: v_i,d ← ω v_i,d + φ_p r_p (p_i,d-x_i,d) + φ_g r_g (g_d-x_i,d)
    Update the particle's position: x_i ← x_i + v_i
    if f(x_i) < f(p_i) then
      Update the particle's best known position: p_i ← x_i
      if f(p_i) < f(g) then
        Update the swarm's best known position: g ← p_i
```

Advantages of PSO:

1. Insensitive to scaling of design variables.
2. Derivative free.
3. Very few algorithm parameters.
4. Very efficient global search algorithm.
5. Easily parallelized for concurrent processing.

Disadvantages of PSO:

1. Slow convergence in the refined search stage (Weak local search ability).