# Report

# Genetic Algorithm and its Application in Pattern Recognition

| UIT2722 ~ Bio Inspired Optimization Techniques | |
|---|---|
| **Name** | **Ashuwin P** |
| **Register Number** | **3122 22 5002 013** |
| **Department** | **Information Technology** |
| **Batch** | **2022 – 2026** |

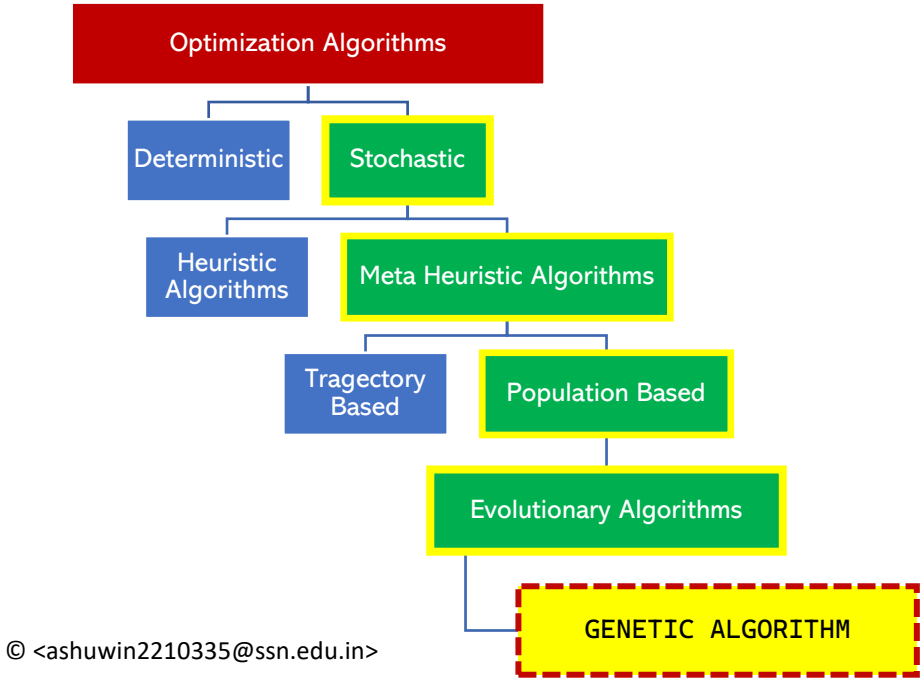
*Submitted to*
*Dr. R. Srinivasan*
*Professor*
*Department of Information Technology*
*SSNCE, Chennai*

# Table of Contents


1. **Introduction**
2. **Overview of Genetic Algorithms (GAs)**
   2.1. Key Concepts and Terminologies
3. **Working Mechanism of Genetic Algorithms**
   3.1. Initial Population Generation
   3.2. Fitness Evaluation
   3.3. Selection Process
   3.4. Crossover (Recombination)
   3.5. Mutation
   3.6. Replacement Strategy
   3.7. Termination Criteria
4. **Importance and Applications of Genetic Algorithms**
5. **Key Features and Advantages of GAs**
6. **Genetic Algorithm for Pattern Recognition**
   6.1. Implementation
   6.2. Output
   6.3. Inferences
7. **Conclusion**

## 1. Introduction

Genetic algorithms (GAs) are optimization techniques inspired by the process of natural selection. (***Natural selection*** *is a process in evolution where organisms better adapted to their environment tend to survive and reproduce, passing on their advantageous traits to the next generation. This mechanism was first proposed by Charles Darwin in the 19th century*). These algorithms are particularly effective for finding solutions to complex problems where Deterministic Algorithms may be inadequate. The Chart below shows the position of Genetic Algorithm in Optimization Algorithms.






## 2. Overview of Genetic Algorithms (GAs)

Genetic algorithms are search-based optimization methods that simulate evolutionary processes to identify optimal or near-optimal solutions. They are commonly applied in challenging and ***high-dimensional problem spaces***

**Key Concepts and Terminologies:**

- **Chromosome:** A representation of a potential solution, typically encoded as a binary string or a value vector.
- **Gene:** A single unit within a chromosome that defines a specific parameter.
- **Allele:** The value a gene can take.
- **Population:** The set of all chromosomes in the current generation.

## *3. Working Mechanism of Genetic Algorithms*

**1. Initial Population Generation:**
The algorithm begins by creating a diverse set of chromosomes that represent potential solutions to the problem.

**2. Fitness Evaluation:**
Each chromosome's quality is assessed using an ***objective function*** that quantifies how well it meets the problem's requirements.

**3. Selection Process:**
Chromosomes are selected for reproduction based on their fitness scores, with higher-performing individuals having a higher chance of being chosen.

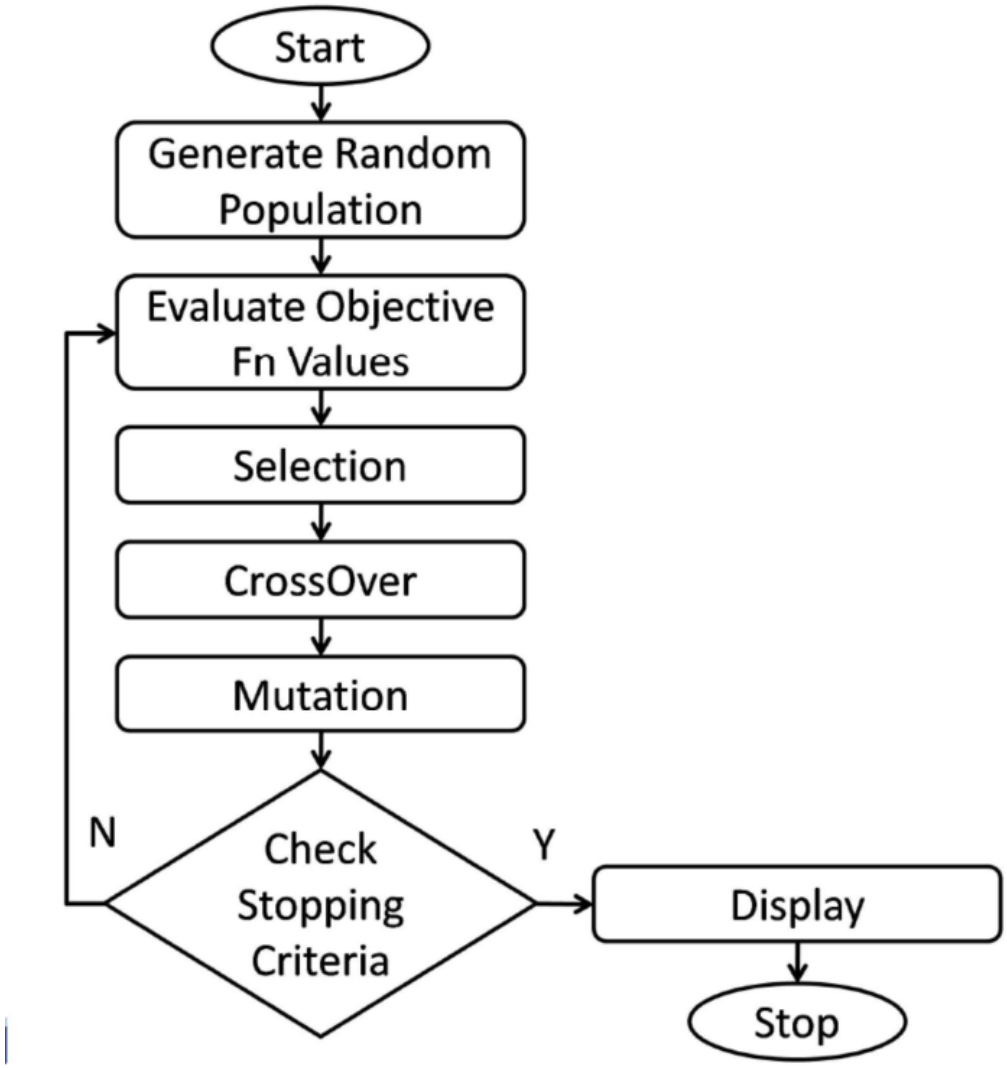
**4. Crossover (Recombination):**
Selected chromosomes are paired, and portions of their genetic information are exchanged to produce new offspring, enhancing diversity.

**5. Mutation:**
Random changes are introduced into some offspring to ensure genetic diversity and prevent premature convergence to suboptimal solutions.

**6. Replacement Strategy:**
Newly created offspring replace some or all of the population, depending on the chosen strategy.

**7. Termination Criteria:**
The process continues until a specified condition is met, such as reaching the maximum number of generations or finding an acceptable solution.

The Flowchart given below represents the Working Mechanism of Genetic Algorithm




**Roulette Wheel Selection**

$$p(i) = \frac{f(i)}{\sum_{j=1}^{n} f(j)}$$

**Where,**

**P(i) represents Probability of $i^{th}$ Chromosome for Selection**

**F(i) represents Fitness Value of $i^{th}$ Chromosome evaluated based on Objective function (For Pattern Recognition Manhattan Distance is used)**

- **Roulette Wheel brings Randomness in Selection Process**

- **The Fittest Chromosome (The Chromosome that has higher fitness value in the current population) gets High chance to become Parent for Crossover process to create Offsprings for Next generation. The offsprings thus produced will be probably "Good"**
- **The Chromosome that has lower fitness value in the current population gets lesser chance to become Parent for Crossover process to create Offsprings for Next generation. This may initially seem to be "Bad" but over generations, It may become "Good" !**
- **A Random Number 'n' is generated [0,1]**
- **Cumulative Distribution Function can be formed and the 'n' value can be used to select a chromosome.**

## *4. Importance and Applications of Genetic Algorithms*

Genetic algorithms are particularly valuable for tackling complex problems with high-dimensional search spaces. **Genetic Algorithms (GAs)** can be used to solve **multimodal problems**. Multimodal problems are optimization problems that have **multiple local optima** or "peaks" in the search space, meaning there are several solutions that can be considered optimal depending on the criteria or constraints. They can effectively handle multiple objectives and constraints and have applications in areas such as:

- Pattern recognition
- Machine learning
- Scheduling and optimization

## *5. Key Features and Advantages of GAs*

**Key Features:**

- **Population-based Search:** GAs explore multiple solutions at the same time, enabling a broader exploration of the search space.
- **Stochastic Approach:** Randomness in selection, crossover, and mutation helps prevent local optima and promotes the discovery of better solutions.

- **Fitness-driven Selection:** Ensures that higher-quality solutions have a higher likelihood of being carried into the next generation.

**Advantages:**

- **Robustness:** Less likely to get stuck in local optima.
- **Parallelism:** Easy to parallelize for faster performance on modern multi-core systems.
- **Adaptability:** Can be tailored to different types of problems by modifying encoding, objective functions, and genetic operators.

## 6. Genetic Algorithm for Pattern Recognition

**Pattern Recognition**

**Pattern recognition** is the process of identifying regularities, patterns, or trends within data. It is a fundamental task in artificial intelligence (AI) and machine learning and is widely used across various fields such as:

- **Computer Vision**: Identifying objects or features within images or videos.
- **Natural Language Processing (NLP)**: Recognizing patterns in text, such as identifying sentiment, speech, or named entities.
- **Data Mining**: Discovering patterns and relationships in large datasets.

Pattern recognition techniques can be applied to both structured and unstructured data, including images, sound, and text, with the goal of classifying or predicting based on the observed patterns.

**Digital Images and Pattern Recognition**

- **Digital Images**: A digital image is a representation of visual information, where the image is composed of **pixels** arranged in a matrix format. Each pixel represents a specific point in the image and holds information about its Colour and intensity.
- **Pixels and Colour Depth**: The number of possible images increases **exponentially** with the number of pixels and the Colour depth. The more pixels in the image and the higher the Colour depth, the more complex the

image data becomes. This makes pattern recognition in images challenging but also rich in potential for extracting meaningful information.

- **Pattern Recognition in Images**: In image processing, pattern recognition involves identifying specific arrangements or configurations of pixel values that correspond to a **target pattern**. For example, recognizing faces, text, or objects involves detecting certain spatial arrangements of pixel values that match a known shape, object, or texture.

## *6.1. Implementation*

https://github.com/ashuwin-p/Bio-Inspired-Optimization-Techniques/tree/main/Genetic_Algorithm_Pattern_Recognition

```
import numpy as np
import random
import matplotlib.pyplot as plt
import math
class Chromosome:
    def __init__(self, chromosome):
        self.chromosome = chromosome
        self.fitness = None
        self.prob = None

    @staticmethod
    def random_chromosome(size_):
        return np.random.choice([0, 255], size=size_)


    def manhattan_distance(self, other):
        return np.sum(np.abs(other - self.chromosome))

    def __lt__(self, other):
        return self.fitness < other.fitness

class GeneticAlgorithm:
    def __init__(self, solution, population):
```

```
        self.population = population
        self.size = len(self.population)
        self.solution = solution
    def evaluation(self):
        for chromosome in self.population:
            manhattan_distance =
chromosome.manhattan_distance(self.solution)
            chromosome.fitness = 1 / (1 + manhattan_distance)

        if len(self.population) > self.size:
            self.population = sorted(self.population, key=lambda x:
x.fitness, reverse=True)[:self.size]
        return

    def build_roulette_wheel(self):
        total_fitness = sum(chromosome.fitness for chromosome in
self.population)
        for chromosome in self.population:
            chromosome.prob = chromosome.fitness / total_fitness

        cumulative_distribution = []
        cdf = 0
        for chromosome in self.population:
            cdf += chromosome.prob
            cumulative_distribution.append(cdf)
        return cumulative_distribution

    def selection(self, cumulative_distribution):
        ch1, ch2 = None, None
        def select():
            n = random.random()
            for i, chromosome in enumerate(self.population):
                if n <= cumulative_distribution[i]:
                    return chromosome

        parent1 = select()
        parent2 = select()
        while np.array_equal(parent2.chromosome,
parent1.chromosome):
            parent2 = select()
        return parent1, parent2
```

```
    def crossover(self, parent1, parent2):
        # Generate a random crossover point
        crossover_point = random.randint(1, len(parent1.chromosome)
- 1)

        offspring_chromosome = np.concatenate(
            (parent1.chromosome[:crossover_point],
parent2.chromosome[crossover_point:])
        )

        offspring = Chromosome(offspring_chromosome)
        return offspring


    def mutation(self, offspring):
        num_mutation = 5
        m = len(offspring.chromosome)
        for i in range(num_mutation):
            idx = random.randint(0, m-1)
            if offspring.chromosome[idx] == 0 :
                offspring.chromosome[idx] = 255
            else:
                offspring.chromosome[idx] = 0
        return offspring

    def best_value(self):
        return max(chromosome.fitness for chromosome in
self.population)


def Run_GA(solution, population, num_iter):
    GA = GeneticAlgorithm(solution, population)
    values = []
    for _ in range(num_iter):
        GA.evaluation()
        if _ == num_iter - 1:
            break
        values.append(GA.best_value())
        cdf = GA.build_roulette_wheel()
        for k in range(GA.size//2):
            p1, p2 = GA.selection(cdf)
```

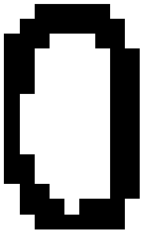
```
            offspring = GA.crossover(p1, p2)
            mutated_offspring = GA.mutation(offspring)
            GA.population.append(mutated_offspring)
    return GA.population, values


import cv2
def preprocess_image(image_path):
    src_image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    original_shape = src_image.shape
    _, bw_image = cv2.threshold(src_image, 127, 255,
cv2.THRESH_BINARY)
    flattened_image = bw_image.flatten()
    return flattened_image, original_shape


src_image_path = r".\zero1.jpg"


image, osize = preprocess_image(src_image_path)


plt.title("Solution Chromosome --> Target Image")
plt.imshow(image.reshape(osize), cmap='gray')
plt.axis('off')
plt.show()

population_size = 12
population = []
for _ in range (population_size):
    random_chromosome = Chromosome.random_chromosome(image.shape)
    population.append(Chromosome(random_chromosome))


solutions, logs = Run_GA(image, population, 75000)

plt.plot(logs)
plt.title("Fitness Over Generations")
plt.xlabel("Generation")
plt.ylabel("Fitness (higher is better)")
plt.show()


# Determine grid size for displaying images
```

```
num_images = len(solutions)
cols = 4
rows = math.ceil(num_images / cols)

# Create subplots
fig, axes = plt.subplots(rows, cols, figsize=(15, rows * 3))
fig.suptitle("Solutions with Fitness Scores", fontsize=16)

# Flatten axes array for easy indexing
axes = axes.flatten()

for i, img in enumerate(solutions):
    axes[i].imshow(img.chromosome.reshape(osize), cmap='gray')
    axes[i].set_title(f"Fitness: {img.fitness:.9f}")
    axes[i].axis('off')

# Hide any remaining empty subplots
for j in range(i + 1, len(axes)):
    axes[j].axis('off')

plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()
```

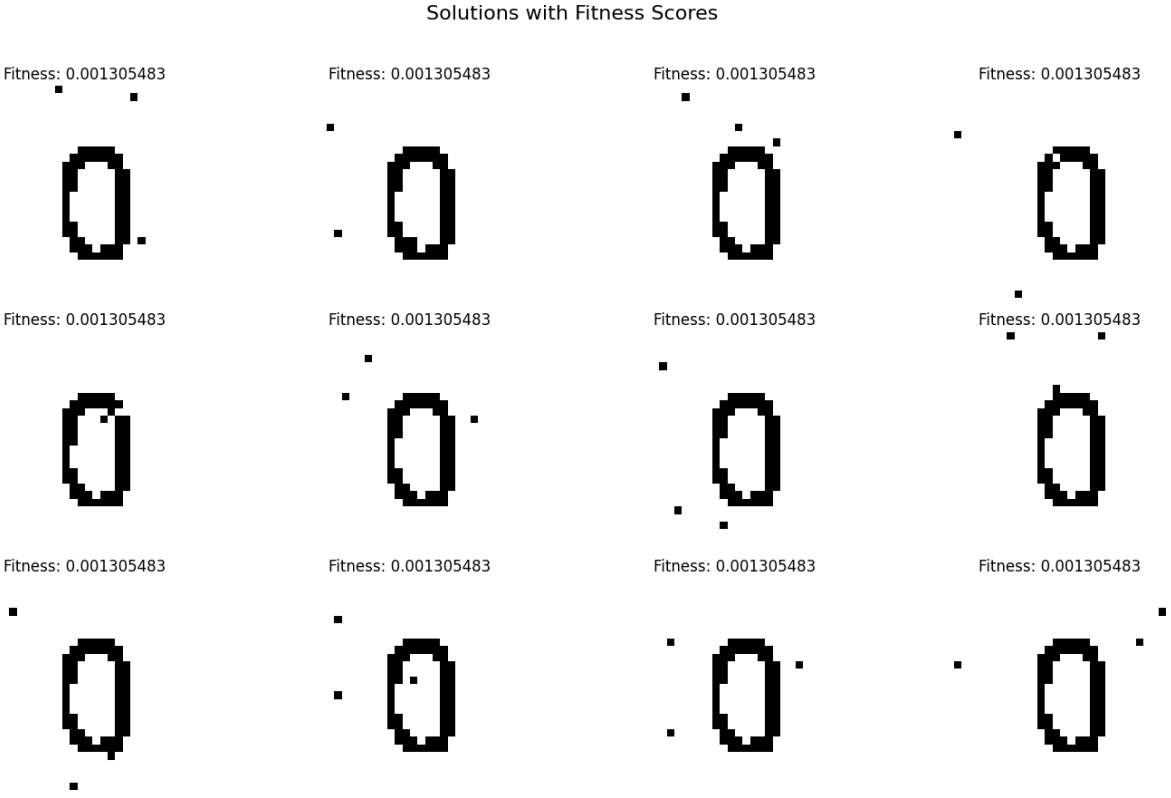
## 6.2. Output

### 1) INPUT IMAGE (28 x 28)

Solution Chromosome --> Target Image

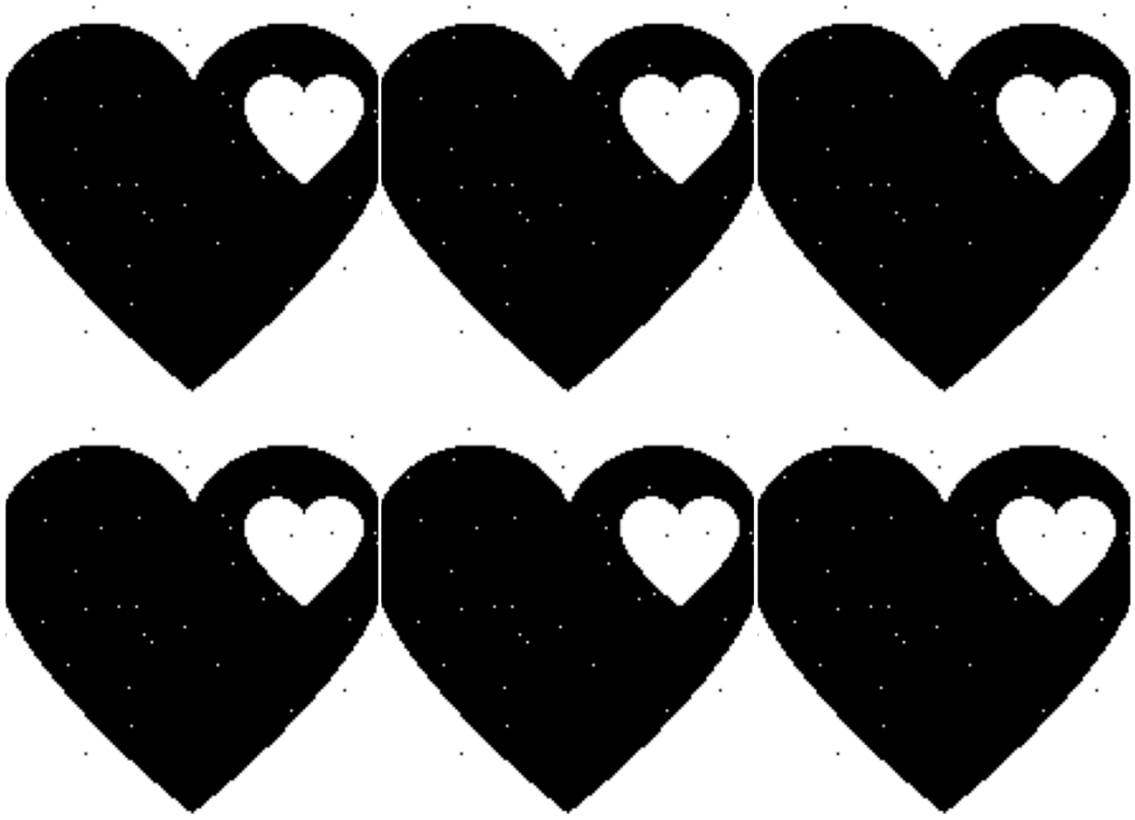


### 1) OUTPUT IMAGES



## 2) INPUT IMAGE (157 x 157)

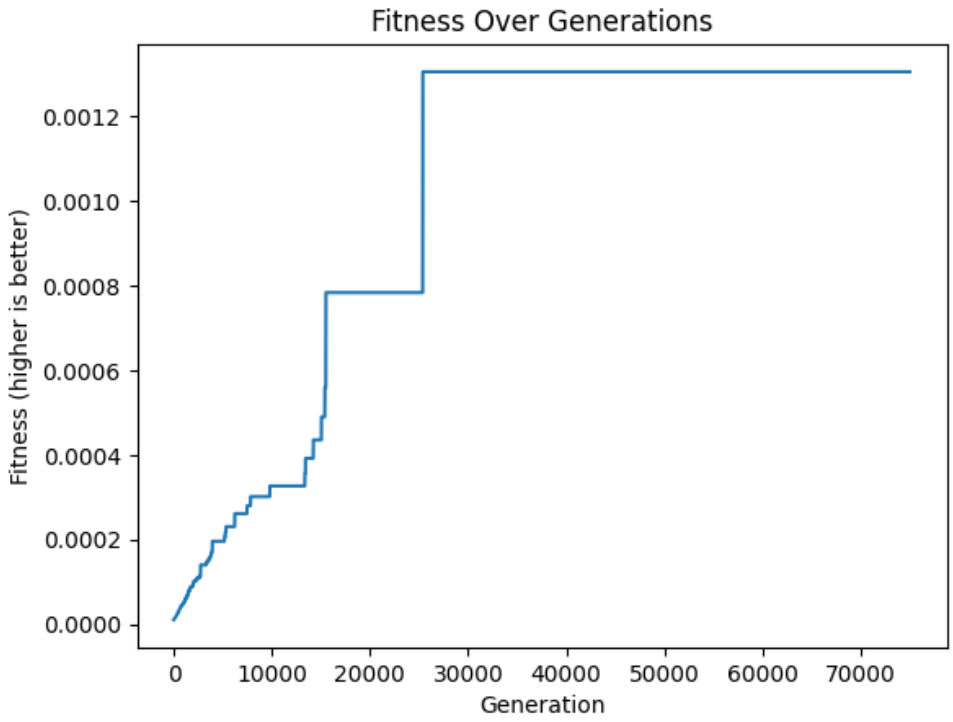
## 2) OUTPUT IMAGES



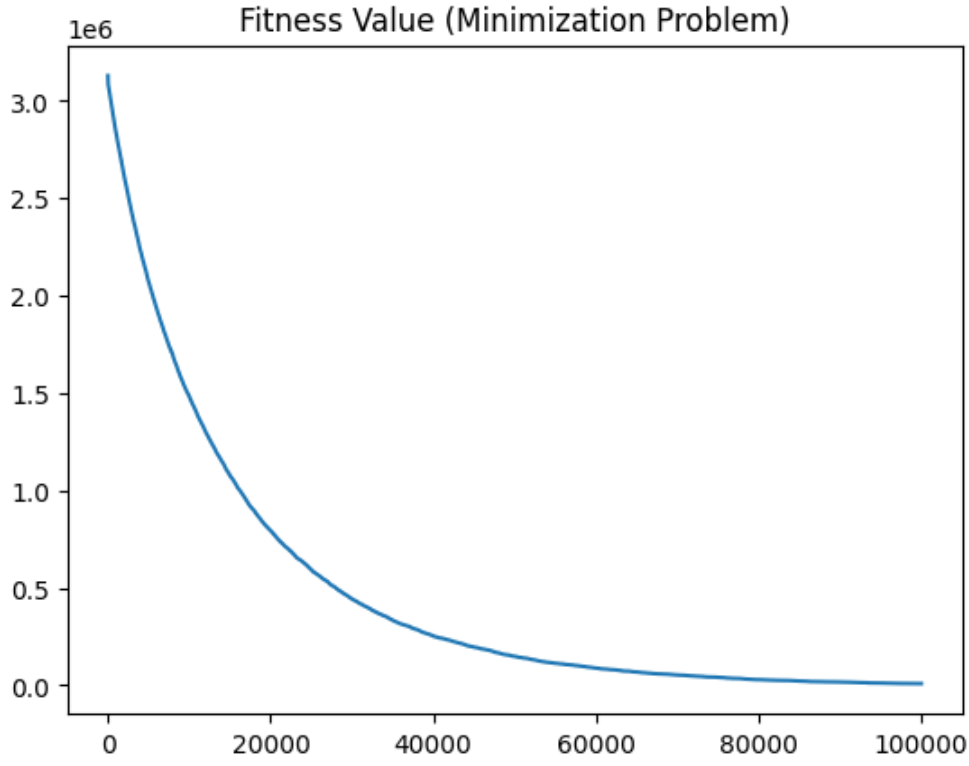
## 6.3. Inferences

**(Inferences are given at every paragraph of the report)**

**Fitness Value over Generations for INPUT 1**




**Fitness Value over Generations for INPUT 2**



- ⇨ **The Fitness Value Monotonically Converged**
- ⇨ **Global Optimum can be achieved by increasing the Number of Iterations (Generations)**
- ⇨ **Randomness in selection, crossover, and mutation helps prevent local optima**
- ⇨ **Since Genetic Algorithm is Population Based we got Multiple Solutions matching the Pattern of the Input Image.**

## *7. Conclusion*

Genetic algorithms (GAs) have proven to be powerful tools for solving complex optimization problems due to their adaptive and evolutionary nature. Their application in pattern recognition underscores their ability to handle the intricacies of recognizing patterns within data that may not be straightforward or linear. By simulating natural evolutionary processes, GAs provide robust search mechanisms that explore vast solution spaces, making them suitable for high-dimensional and non-convex problems where traditional algorithms might struggle.

In the context of pattern recognition, GAs are particularly advantageous because they do not rely on gradient-based information and are flexible enough to adapt to different objective functions, allowing them to handle non-differentiable or complex search spaces.This flexibility allows them to find optimal or near-optimal solutions in image analysis, speech recognition, and biometric systems, among other fields. The iterative process of selection, crossover, and mutation ensures a comprehensive exploration of potential solutions, promoting diversity and reducing the risk of premature convergence to suboptimal solutions.

Overall, the implementation of genetic algorithms in pattern recognition demonstrates significant potential for advancing technologies that rely on accurate and efficient data interpretation. The algorithm's ability to self-improve through generations and its inherent parallelism make it a suitable choice for modern pattern recognition challenges. As computational power continues to grow, the application of GAs is likely to expand, contributing to more sophisticated and intelligent systems capable of processing complex data patterns effectively.