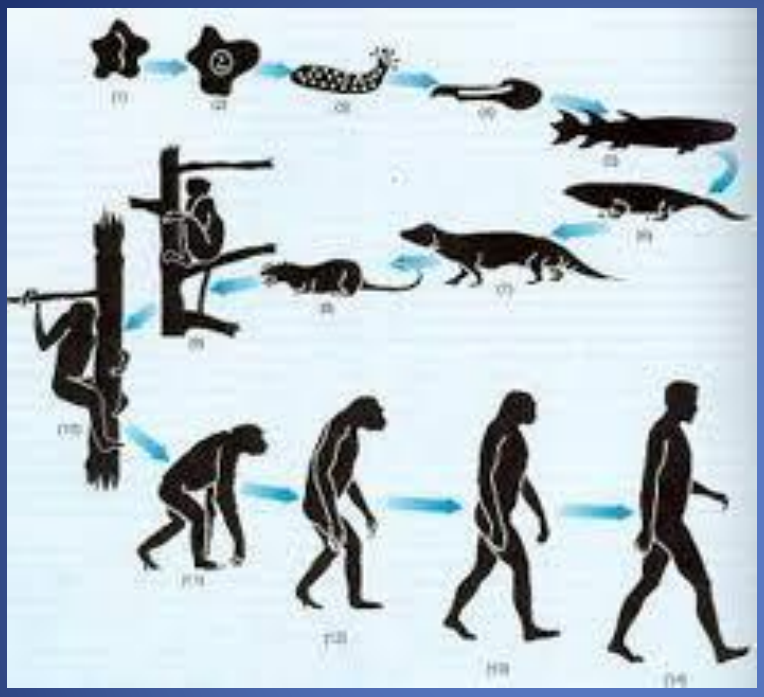
2022



Face Detection

AI REPORT

JAI SHANKER | ABBAS ALI

BSCS | 2020-43
BSCS | 2020-59

Problem Statement

This report is based on solving a template-matching problem using Evolutionary Algorithm derived from Darwin's Evolution. Instead of applying a programming approach, we must implement some logical thinking approach to detect the resemblance between two images. To do that, we will focus on the evolutionary algorithms approach and try to find an optimal solution for this problem.

What Evolutionary Algorithm is:

Evolutionary Algorithm follows mechanisms inspired by natural evolution and solves problems of the process that relate to the behavior of natural evolution of living organisms. It provides very close solutions to problems that cannot be achieved easily from other techniques.

How it works:

In Evolutionary Algorithm the problem gets analyzed by a defined "Population" which contains the possible solutions to the problems (so-called "chromosomes" in EA) and each possible solution is scored using a fitness value (using fitness function) that indicates how much close that specific solution is.

Natural Phenomenon:

The idea was inspired by a famous biological theory "Theory of Evolution by Charles Darwin", which states that "Every species is evolved from its ancestor species". But nature won't allow too quick change or evolution, but it would a species to evolve or disappear slowly and gradually. According to Darwin's theory all species are evolved from a single cell of origin, and so are humans.

The key points of the idea were that in all species there is a difference it can be large and small even in the case of chromosomes of a father and his child. A species turns into another species with some small changes as time passes by. Species who survive more time in the environment are considered as "Fittest" and the "Fitter" one has more chances of mating with whole populations and making children. The base of the Evolutionary Algorithm is about "Best Fitter" from the population.

Theory + Initialization:

Our Algorithms for face detection will use the same idea of Darwin's theory, we will initiate a random population that can be or near to possible solutions (*this population will contain (x, y) points from random points of the large group image*), every individual of the population will be representing a possible solution corresponding to its x, y coordinate that how good their solution is to the problem. The individual then evolves through successive iterations called generation. So, in the case of EA individuals who are the most petite fit will either have fewer chances of meeting with the new generation or they

will disappear from the next generation, and highly fit individuals are allowed to mate with other individuals.

```
def populationInitilization(groupRows, groupColumns, population):
    randomPopulation = []
    for i in range(population):
        randomPopulation.append((np.random.randint(groupRows), np.random.randint(groupColumns)))
    return randomPopulation
```

In the result, the population of individuals will become optimal solution to the problem. Evolutionary algorithms are efficient and able to deal with wide range of problems, including those which are complex for other techniques to solve.

Evolution – Fitness:

For the identification of best individual during the evolutionary process, a function will be used to assign a degree of fitness to each chromosome in every generation. The fitness value is assigned by computing means of intensity similarity between the current frame created using (x, y) coordinate and asked face image.

So, the fitness value for our individuals will be assigned by calculating the correlation formula between same size of image array.

```
    for i in range(population):
        if frame[i].shape == boothi.shape:
            frame[i] = frame[i] - frame[i].mean()
            boothi = boothi - boothi.mean()
            frame[i] = frame[i] / frame[i].std()
            boothi = boothi / boothi.std()
            value = np.mean(frame[i]*boothi)
            fitness_value.append(value)
        else:
            fitness_value.append(0)
```

Figure 1: This function returns the correlation values of all individuals.

Selection:

Selection of best fit individuals from whole population is very important to bring our generation near to the optimal solution of problem. So, here selection is based on finding those individuals who have highest correlation values, so they can participate into mating pool of next generation. In sense of programming, we will sort the individual's fitness values and then extract first 2 highest.

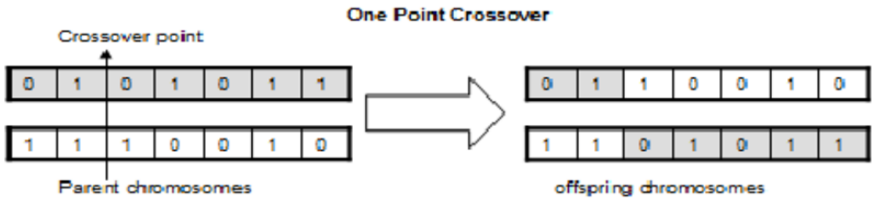
```python
def selection(currentPopulation, fitness_values):
    sort = []
    for i in fitness_values.values():
        sort.append(i)
    return sorted(sort)
```

Figure 2: This Selection function will return the sorted values of fitness.

Cross-Over:

Cross-Over implies the concept of small difference between each gen whenever it evolves to a new gen.

Our Evolutionary Algorithm will follow this process by selecting 1st two fittest individuals from population selection (sorted population) then will randomly break both individuals into 2 pieces and then do swapping between each other parts. As a result, two offspring are generated by combing the partial features of two chromosomes.




```python
for i in range(0, population, 2):
    parent1_part = concat_binary[i]
    parent2_part = concat_binary[i+1]
    parent1_part = list(parent1_part)
    parent2_part = list(parent2_part)
    random_point = np.random.randint(0, 22)

    for j in range(random_point, 22):
        parent1_part[j], parent2_part[j] = parent2_part[j], parent1_part[j]

    parent1_part= ''. join(parent1_part)
    parent2_part= ''. join(parent2_part)
    result_list.append(parent1_part)
    result_list.append(parent2_part)
return result_list
```

If a pair of chromosomes does not cross-over, then the offspring are created as exact copies of each parent, which is not good for the algorithm because we can possibly stuck on the same point every time.

Mutation:

Mutations are essential to evolution. A mutation is a change in DNA (change in chromosome). It avoids the loss of genetic diversity. Its role is to provide guarantee that the algorithm is not trapped on a local optimum when searching.

So, for implementing mutation in our algorithm we will choose a bit at a random location to flip its value from 0 to 1, or 1 to 0.

```
def mutationOfBits(crossOver):
    value = crossOver[population-1]
    value = list(value)
    random_mutation = np.random.randint(0,22)

    if value[random_mutation]=='0':
        value[random_mutation]='1'
    else:
        value[random_mutation]='0'

    value = ''.join(str(v) for v in value)
    crossOver[population-1] = value

    return crossOver
```

Figure 3: This function is doing Mutation of random individual

Test = Termination / New Generation:

If our predefined preferred threshold condition gets satisfied, then result with “face found!”, else create new population for processing.

```
def faceDetection(currentGeneration, fitnessVal, sorted_fitness):
    newlyCreatedGeneration = newGeneration(currentGeneration, fitnessVal[1], sorted_fitness)
    key = []
    for i in fitnessVal[0].values():
        if (i >= fitnessThreshold):
            print("face found!")
            key.append([j for j in fitnessVal[0] if fitnessVal[0][j]== i])
            return [key, [], i]
        else:
            return newlyCreatedGeneration
```

Figure 4: Check for desired fitness is achieved or not?

Experimentations:

Let us test how much effective and robust the Evolutionary Algorithm is.

A. Hypothesis No. 1:

- What if we just increase the No: of generation without increasing population size to check that our algorithm effective and fast our algorithm is.

The following experiments will test this hypothesis:

1) Experiment No. 1:

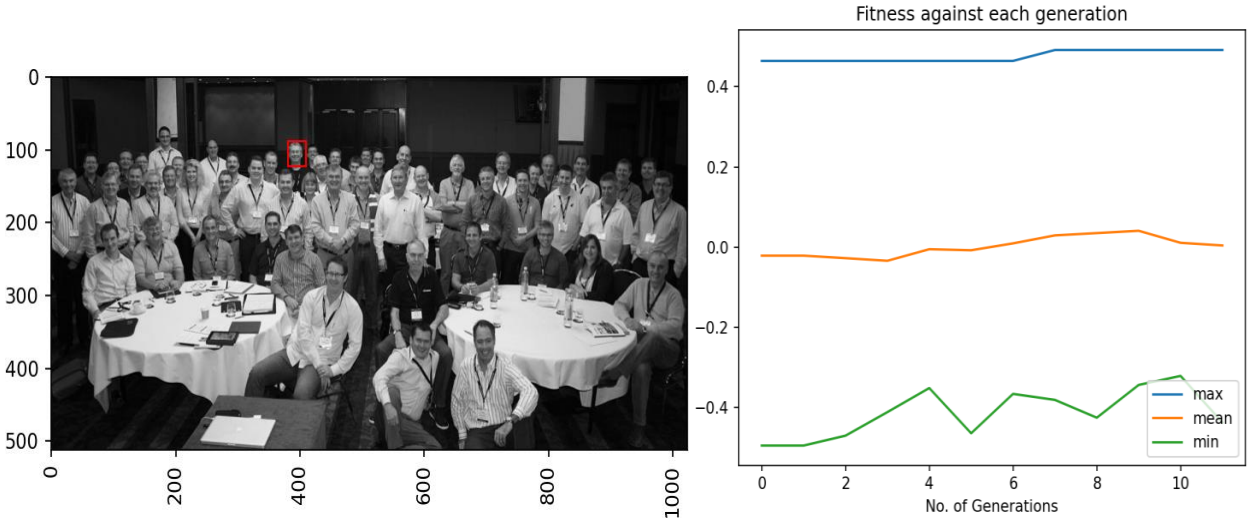
- **Attempt 01:**
  - **a.** Given population size = 100

b. Threshold = 0.6

c. Generations = 1000

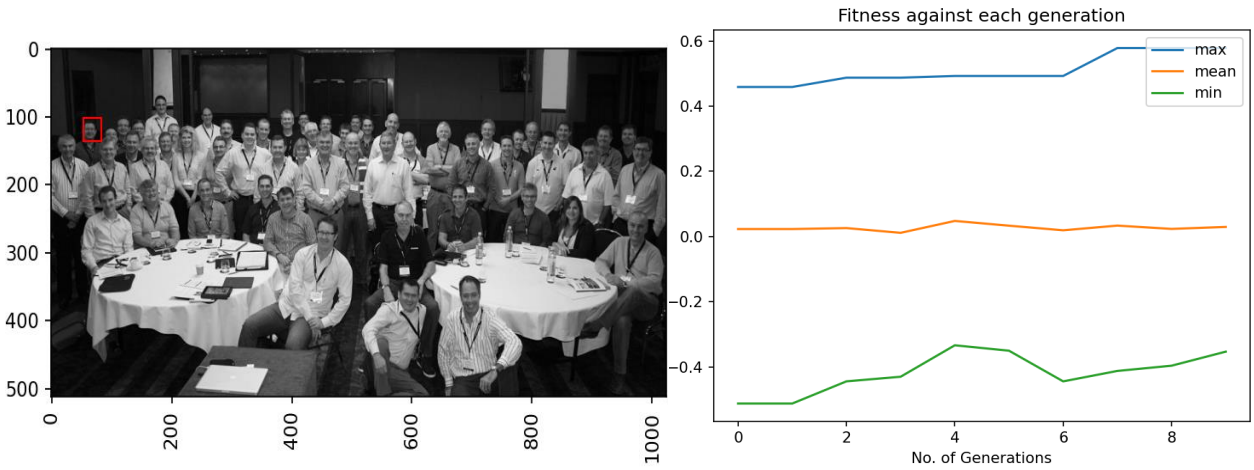
Result:

```
3399088181056356, 0.35195779579321607, 0.38467801354647857,
4911596089858281, 0.4915166530009189, 0.7142391983945197]
Number of generations:  12
```



- **Attempt 02:**
  - **a.** Given population size = 100
  - **b.** Threshold = 0.6
  - **c.** Generations = 1000

```
49259488061606954, 0.49354375552758667, 0.5782985165173712, 0.6256753639515322]
Number of generations:  10
```



- **Analysis of above 2 experiment attempts over hypothesis 1:**
  - **a.** It is observed on the fitness threshold 0.6 that however the generations are increasing it isn't affecting the result our fitness threshold is getting satisfied within 50 generations.
  - **b.** Its result is quick, which shows how robust our algorithm is!
  - **c.** It didn't give exact location because at the fitness level 0.6 there can be many possibilities of similar looking locations.

B. Hypothesis No. 2:

- If size of population gets increased than the image recognition, chances will also increase but as over algorithm follows the step of cross-over and mutation which can be time consuming for large amount of population.
- Let's find out the result pattern when population size increases.

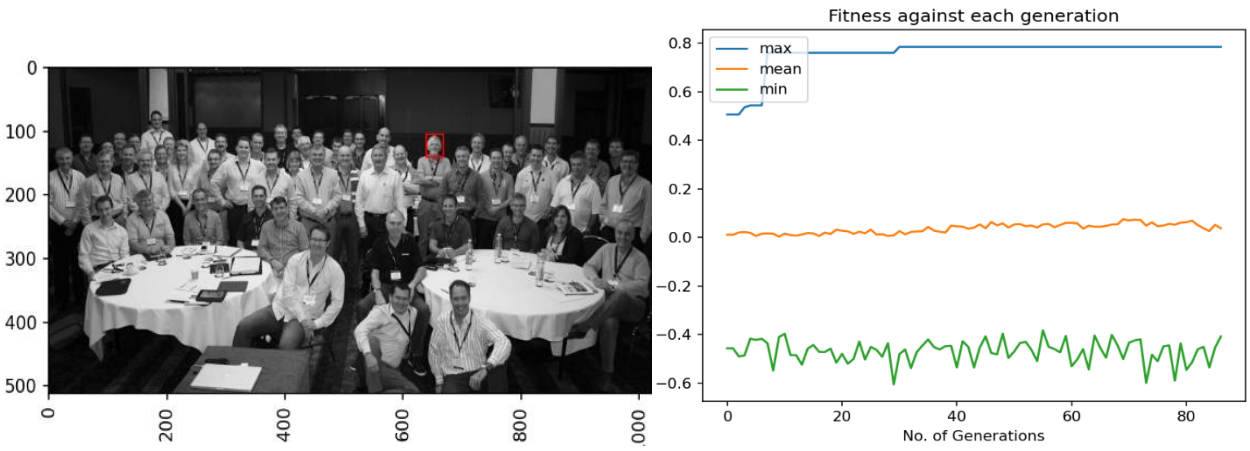
The following experiment will test this hypothesis:

Experiment No. 2:

- **Attempt 01:**
  - **a. Given population = 400**
  - **b. Given generation = 1500**
  - **c. Threshold = 0.9**

Result:

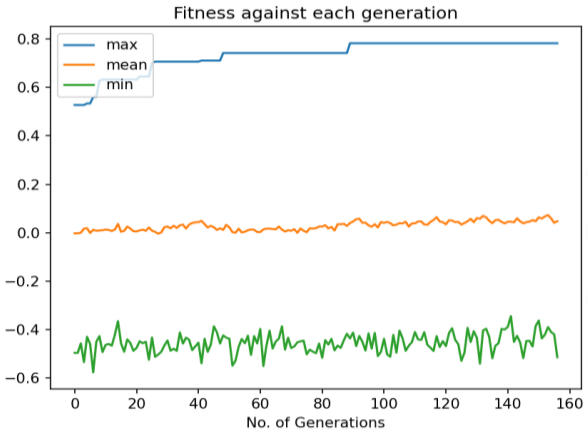
```
7602938608352121, 0.7614718407917639, 0.7847916967718389, 0.9999223384947458]
Number of generations:  87
```



- **Attempt 02:**
    a. **Given population = 600**
    b. **Given generation = 1500**
    c. **Threshold = 0.9**

Result:

```
7634426145198 39, 0.7805301902076813, 0.7830391775336372, 0.9095324576399947]
Number of generations:  157
```



- **Analysis of above 2 experiment attempts over hypothesis 2:**
    a. It is observed that by increasing the size of population the tasks also increase which will obviously take time in executing. Like, here Threshold where 0.9 generations were up to 1000 and population was 400, 500 and it took 5 mint to output the result. The result was correct.

Limitation Experience:

As this algorithm is based on random selection of coordinates, mutation. So, it is possible in one movement you can get the correlation 1 in just 40 generation and at the same time you can also get 0.6 threshold in 150 generations.

The is no assurance that the algorithm will converge. This can be stuck somewhere.

References:

Header image: https://www.rtflash.fr/l-evolution-est-elle-seulement-darwinienne/article

Crossover image: https://www.researchgate.net/figure/Single-point-crossover_fig1_220485962