

# A Genetic Algorithm-Based Solver for Small-Scale Jigsaw Puzzles<sup>1</sup>

## Team 29

**Kirthi Vignan** (2020122004, ECD)

**Sasanka Uppuluri** (2019102036, ECE)

**Siva Durga** (2019102038, ECE)

**Viswanadh** (2019112011, ECD)

**Course Instructors:**

**Ravi Kiran S**

**Sudiptha B**

**Mentor TA: Manaswini**



Github Repo: [Click here](#)

1. Guo, Wenjing & Wei, Wenhong & Zhang, Yuhui & Fu, Anbing. (2020). A Genetic Algorithm-Based Solver for Small-Scale Jigsaw Puzzles. 10.1007/978-3-030-53956-6\_32.

# Index

Introduction

Genetic Algorithm

GUI

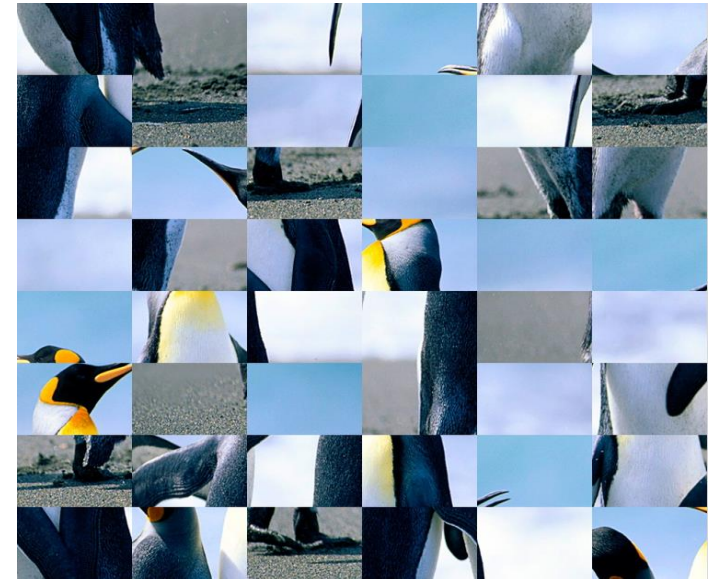
Results and Analysis

Work Division

References

# Introduction

- The concept of solving puzzles is commonly used in numerous domains such as archaeology and medical, the mending of ancient artefact pieces, and the rehabilitation of fractures and dislocated bones.
- To solve this challenge, we employ a genetic algorithm-based strategy.



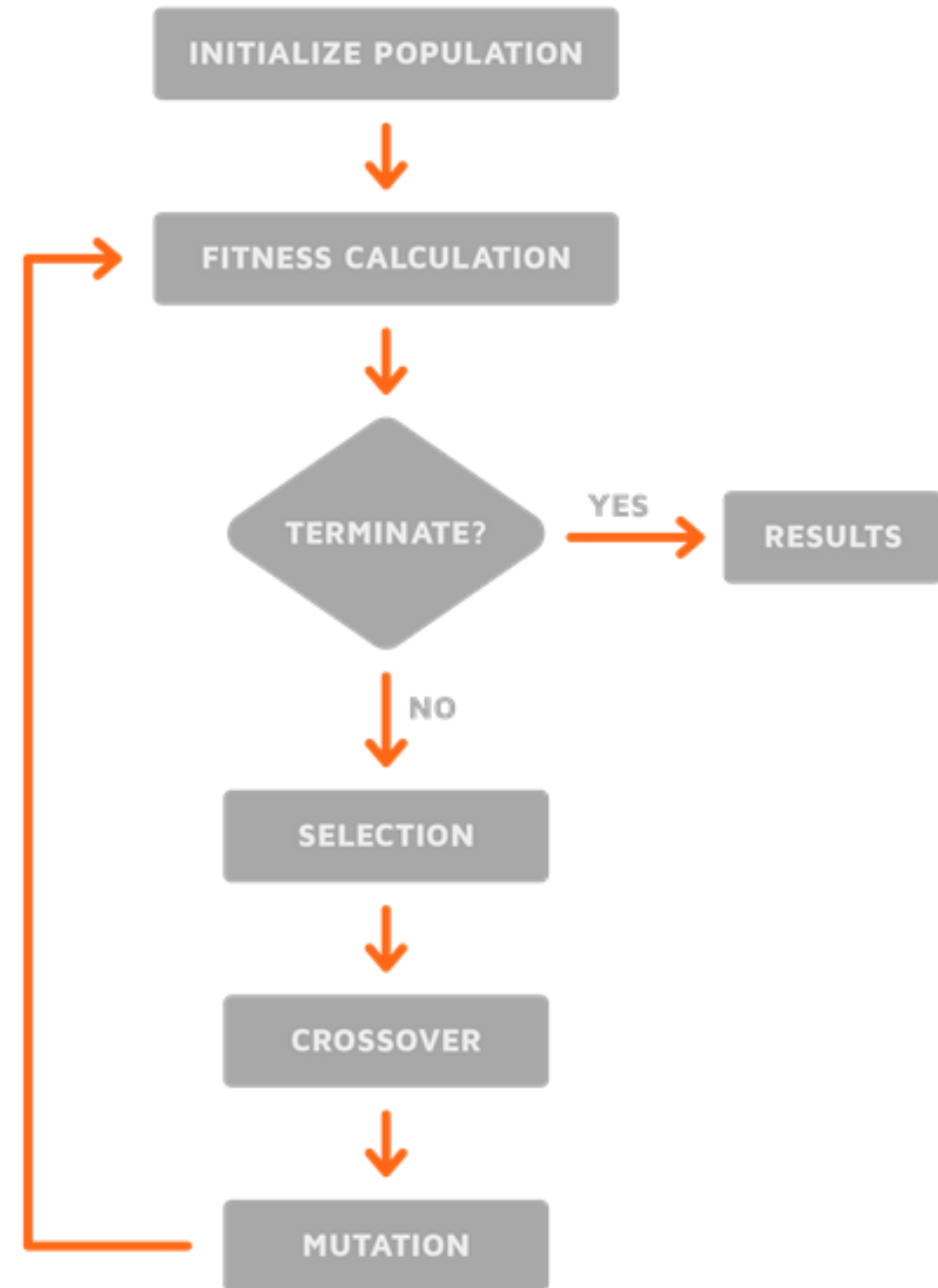
# Genetic Algorithm

Overview

Fitness function

Crossover

Mutation



# Overview

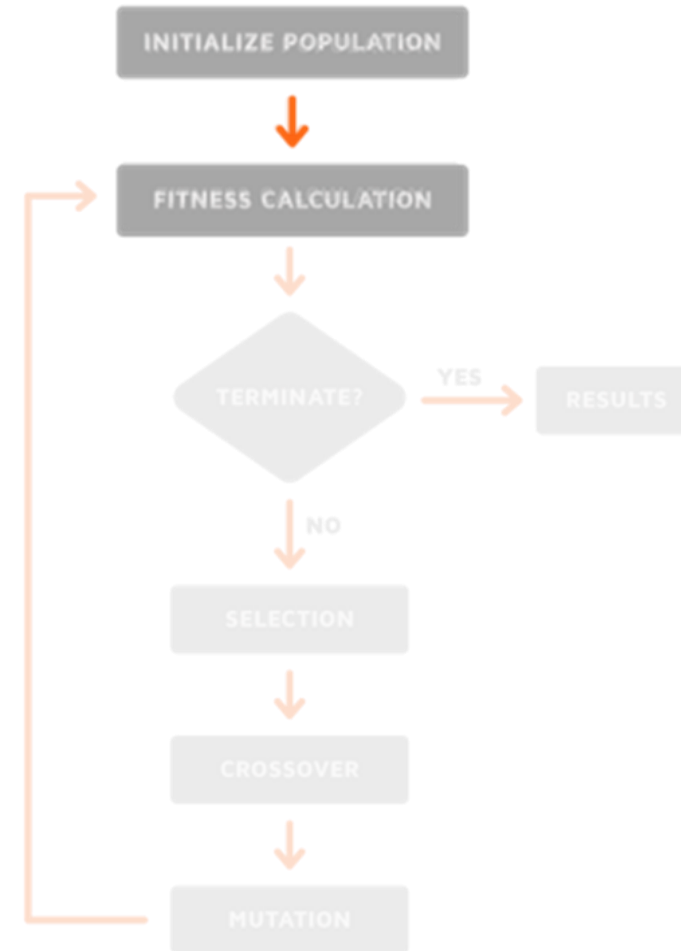
- Randomly generate initial population(chromosomes). A chromosome is a suggested arrangement of the puzzle pieces.
- We evaluate the 'fitness' of the chromosomes using a fitness function to choose the 'best two parents' among the 500 or so generated chromosomes.
- These two parents undergo 'crossover' and 'mutation' in the process of reproduction to give a good child and it is added to the new population.
- Repeat the above steps.

```
1: population = generate 500 random chromosomes
2: for generation number = 1:100
3:     evaluate all chromosomes using the fitness function
4:     new population = NULL
5:     copy 2 best chromosomes to new population
6:     while size(new population) <= 1000
7:         evaluate fitness function of the new population
8:         parent1, parent2 = select best 2 chromosomes
9:         child = crossover(parent1, parent2)
10:        child = mutation(child)
11:        add child to new population
12:    population = new population
```

# Fitness Function

- We define the possibility of two pieces being adjacent pieces as compatibility. The greater the compatibility, the greater the probability of two pieces are adjacent pieces.
- Define the difference measurement function between  $x_i$ , and  $x_j$  to be  $C(x_i, x_j, r)$  where  $r$  denotes the spatial direction - top, bottom, left or right - of  $x_j$  with respect to  $x_i$ , each of size  $K \times K \times 3$ .
- Note that if difference between two pieces is less, the compatibility is maximum
- Define the color space distance between them in the R direction to be

$$D(x_i, x_j, r) = \sqrt{\sum_{k=1}^K \sum_{b=1}^3 x_i(k, K, b) - x_j(k, 1, b)}$$



# Fitness Function

- Mahalanobis distance is defined as:

$$DG(x_i, x_j, r) = \sum_{s=1}^K (\Lambda_{LR}^{ij}(s) - E_{LR}^{ij}(s)) V_{LR} - (\Lambda_{LR}^{ij}(s) - E_{LR}^{ij}(s))^T$$

$$L_{LR}^{ij}(s) = \delta_j(s, 1) - \delta_i(s, K)$$

$$E_{LR}^{ij}(s) = 1/2(\delta_i(s, K-1) - \delta_i(s, K-1) + \delta_j(s, 2) - \delta_j(s, 1))$$

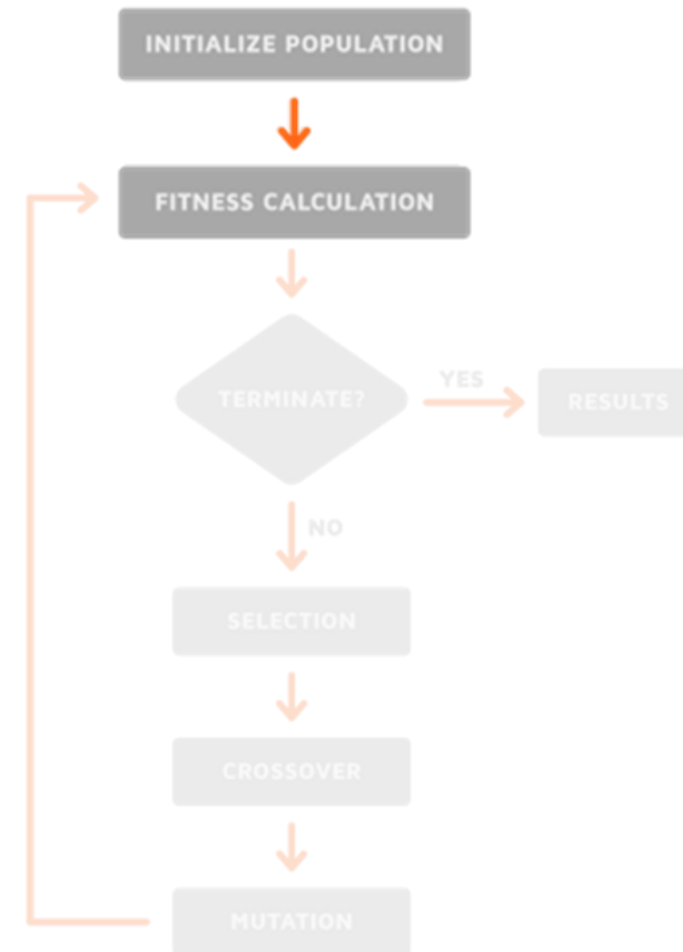
- Now, the difference measurement function is the sum of both these distances

$$C(x_i, x_j, r) = D(x_i, x_j, r) + DG(x_i, x_j, r)$$

- For a given chromosome of size NXM calculate

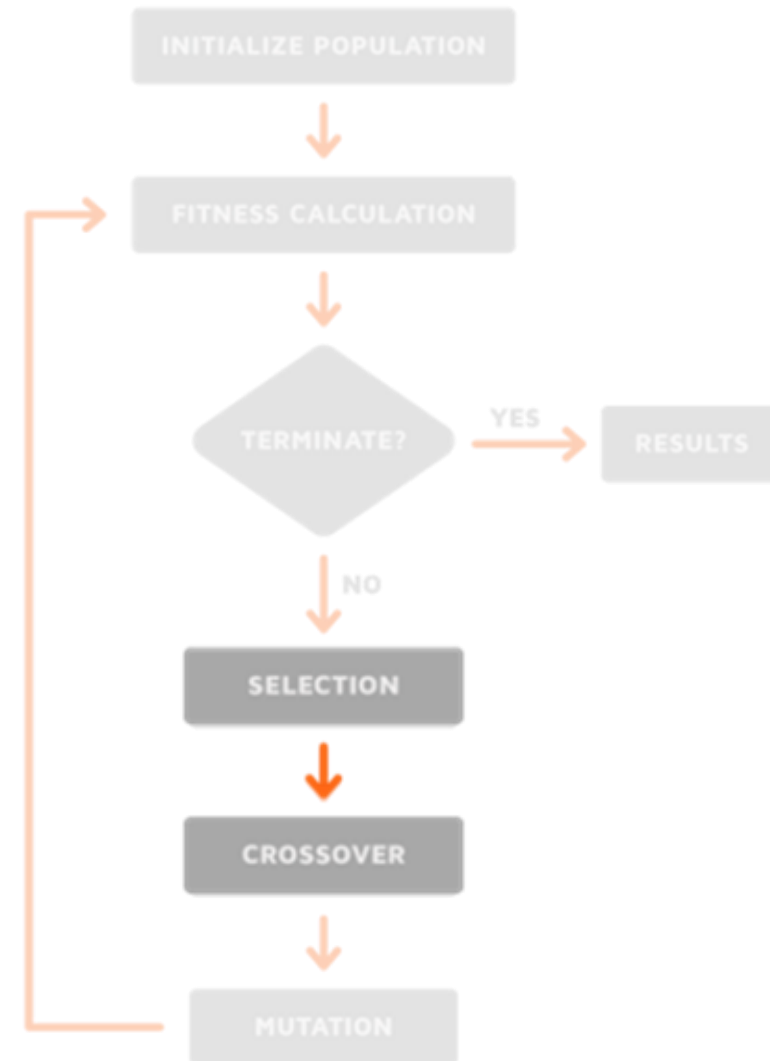
$$\sum_{i=1}^N \sum_{j=1}^{M-1} C(x_{i,j}, x_{i,j+1}, r) + \sum_{i=1}^{N-1} \sum_{j=1}^M C(x_{i,j}, x_{i+1,j}, r)$$

- And take its inverse to obtain the fitness value.



# Cross over

- We push the two chromosomes with the highest fitness value into a new population.
- Now we need the child of these chromosomes
- **What traits to retain is the key!!!**
- **Also, will it be a valid child?(with no duplicate pieces and same size as parent)**





# Cross over

## Issues:

- (1) How to select a suitable piece from the candidate piece ?
- (2) where to place it in the child generation ?

## Crossover computing steps

### Step-1:

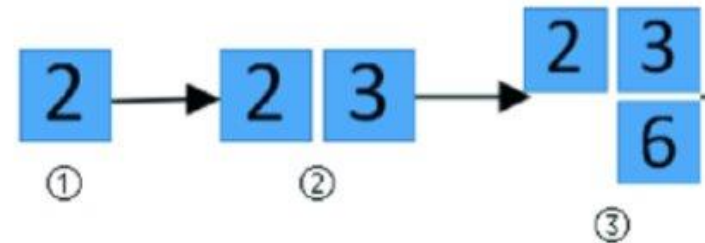
- Check whether there exists a piece boundary for which both parents agree on a piece  $x_j$  (meaning, both contain this piece in the spatial direction  $R$  of  $x_i$ ).
- If such a piece exists, then it is placed in the correct location.

2	3	1
4	6	5
7	8	9

parent. A

9	7	4
2	3	5
1	6	8

parent. B



# Cross over

## Step-2:

- If there is no agreement between the parents on any piece at any boundary, add the 'best buddy' match piece.
- The pieces  $x_i$  and  $x_j$  are said to best-buddies if

$$\begin{aligned} \forall x_k \in \text{Pieces}, C(x_i, x_j, R_1) &\geq C(x_i, x_k, R_1) \\ &\text{and} \\ \forall x_p \in \text{Pieces}, C(x_j, x_i, R_2) &\geq C(x_j, x_p, R_2) \end{aligned}$$

where Pieces is the set of all given image pieces and R1 and R2 are "complementary" spatial relations (e.g. if R1 = right, then R2 = left)

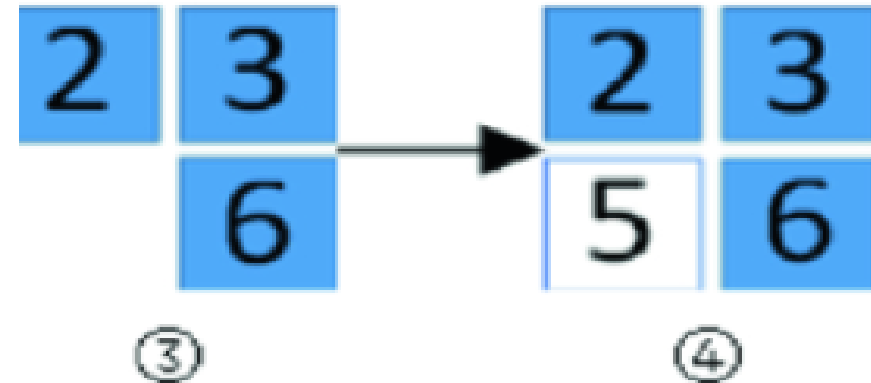
- If a best-buddy piece is found but was already assigned, it is ignored and the search continues for other best-buddy pieces.

2	3	1
4	6	5
7	8	9

parent. A

9	7	4
2	3	5
1	6	8

parent. B



# Cross over

## Step-3:

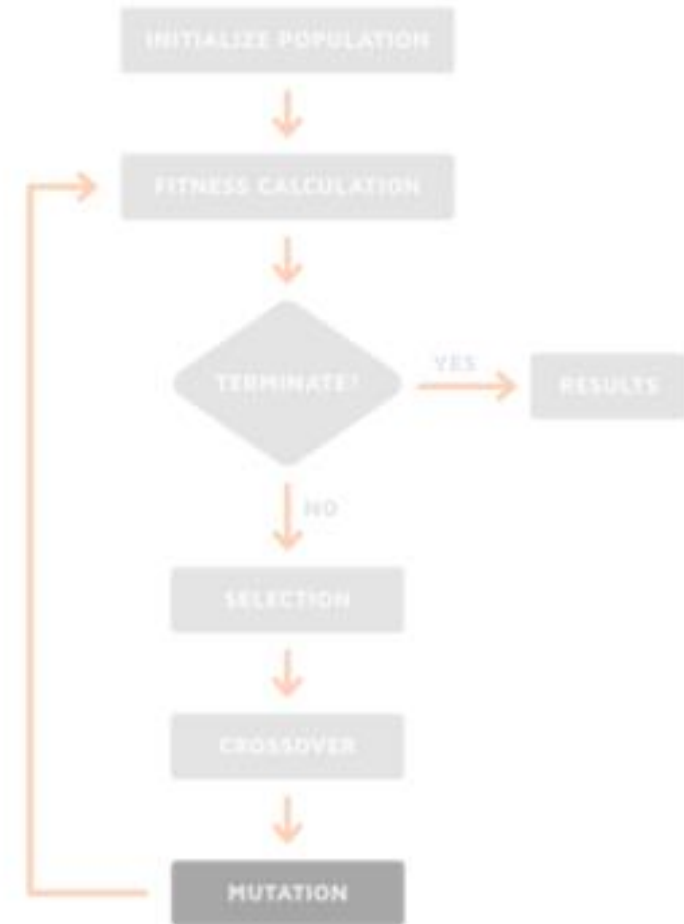
If no best-buddy piece exists, the operator randomly selects a boundary and assigns it the most compatible piece available.

Now, we have a child ready after cross over which may look as follows due to random assignment in step-3

1	2	3
6	5	4
7	8	9

# Mutation

- For each piece in the child chromosome, there is a chance to mutate. The mutation will select a piece and a direction.
- We will find the best matching piece, and exchange the best matching piece with the adjacent pieces in the same direction. If the fitness of the offspring increases after this exchange, it means the mutation succeeds, and the result will be retained.
- If the fitness after the exchange decreases, it means the mutation failed, and we will restore this mutation.




# GUI

- A wrapper Graphical UI is designed using Python's *Tkinter*<sup>1</sup> to execute the genetic algorithm without the need of passing arguments to the code.
- GUI taken an input image and user is given choice to enter "*population size*", "*number of generations*" and "*pieces size*".
- Once the inputs are taken, a random puzzle can be created and then be solved.
- Output at the end of each generation is displayed.
- Finally, user can save the final solved image and view its fitness function's plots

1) <https://docs.python.org/3/library/tkinter.html>



Puzzle Solver



Enter pop value

750

Enter Gen value

15

Piece size between 16-128

28


Puzzle

Solve

Plot

Total puzzle Pieces=12X18

Execution time=60.49 sec



Input Image

Save Image



Total Generations

Number of Initial Population

Size of each puzzle piece

Resultant Image

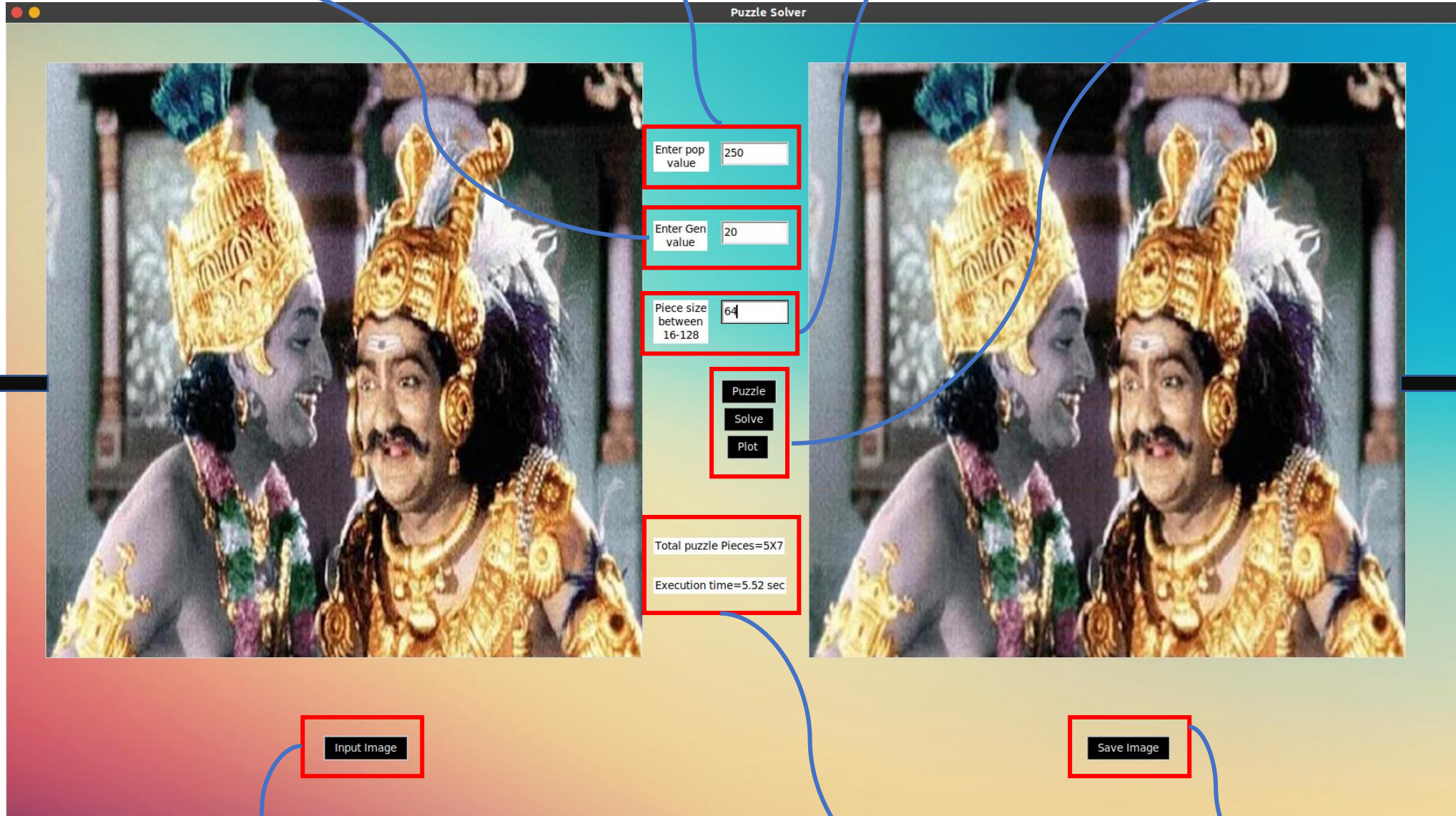
Input  
image

Solved  
output  
image  
(or)  
shuffled  
image

To give Input to GUI

Total puzzle pieces  
and execution time

To Save output image





# Results and Analysis

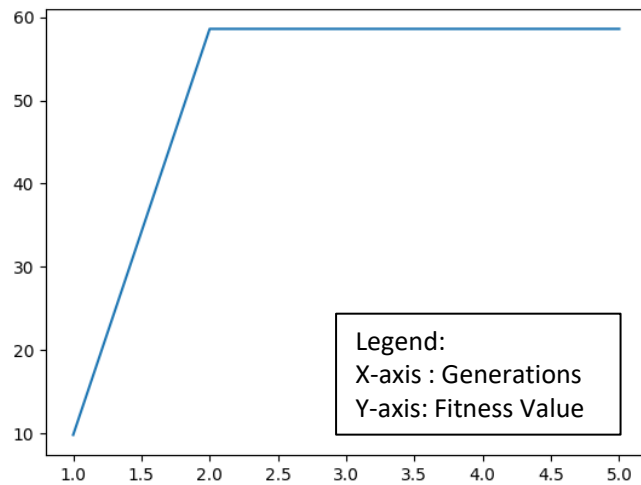
We will see impact of the parameters on time complexity and Fitness scores.

---

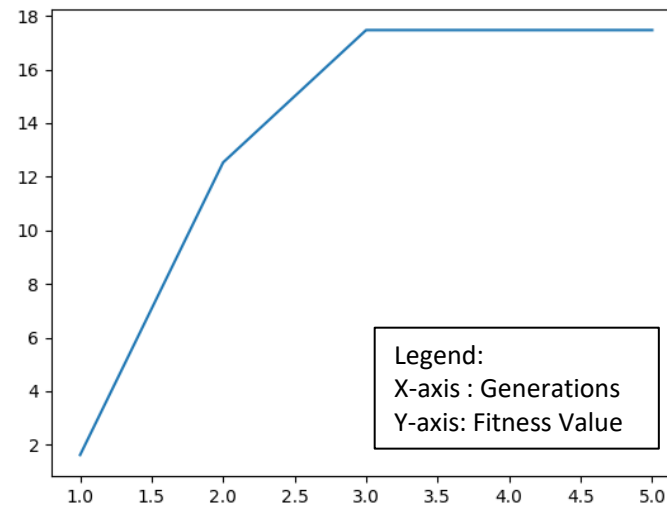


# Impact of Piece Sizes on fitness function

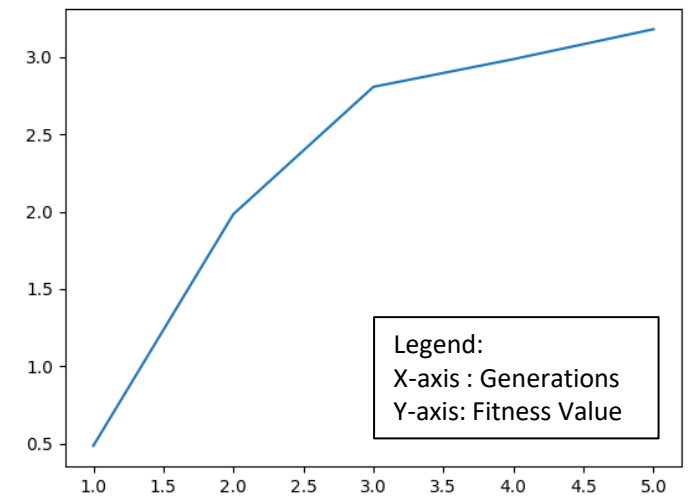
Population Size	Total Generations	Piece Size	Execution Time
400	5	128	2.57s
400	5	64	4.71s
400	5	32	26.56s



Piece Size=128



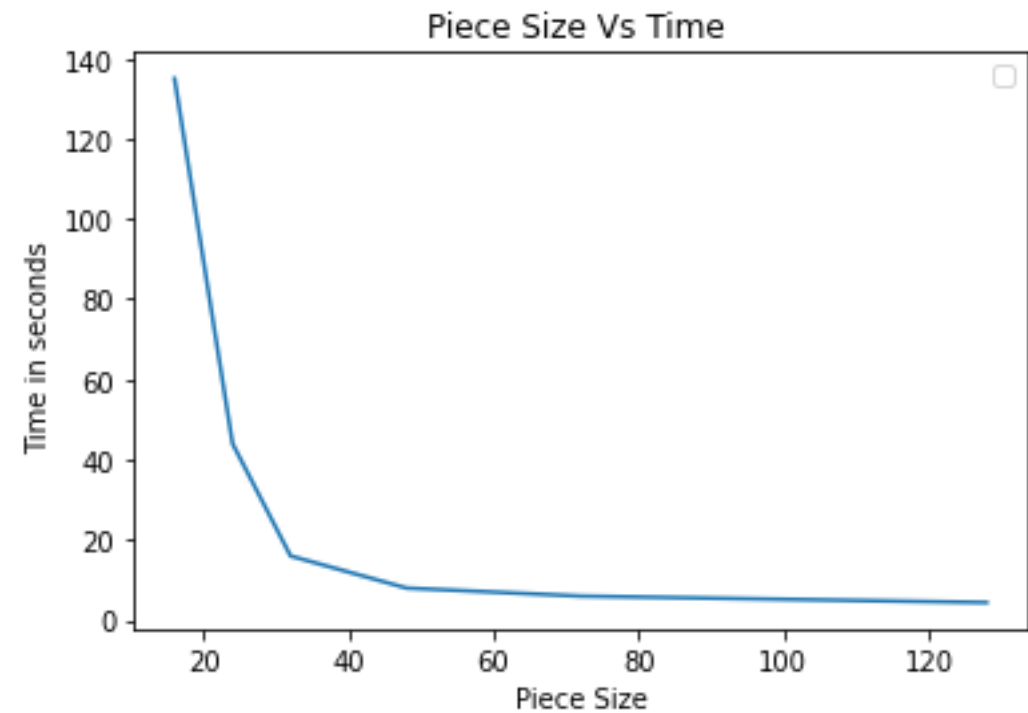
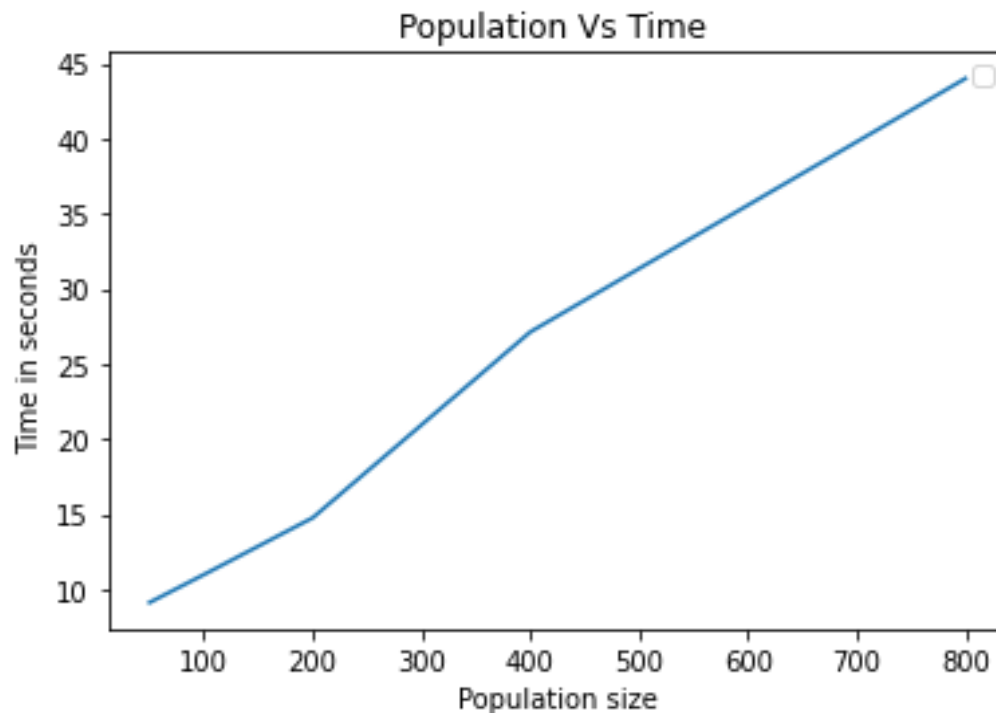
Piece Size=64



Piece Size=32

# Time Complexity Analysis

Test Image : *Mayabazar.jpg*

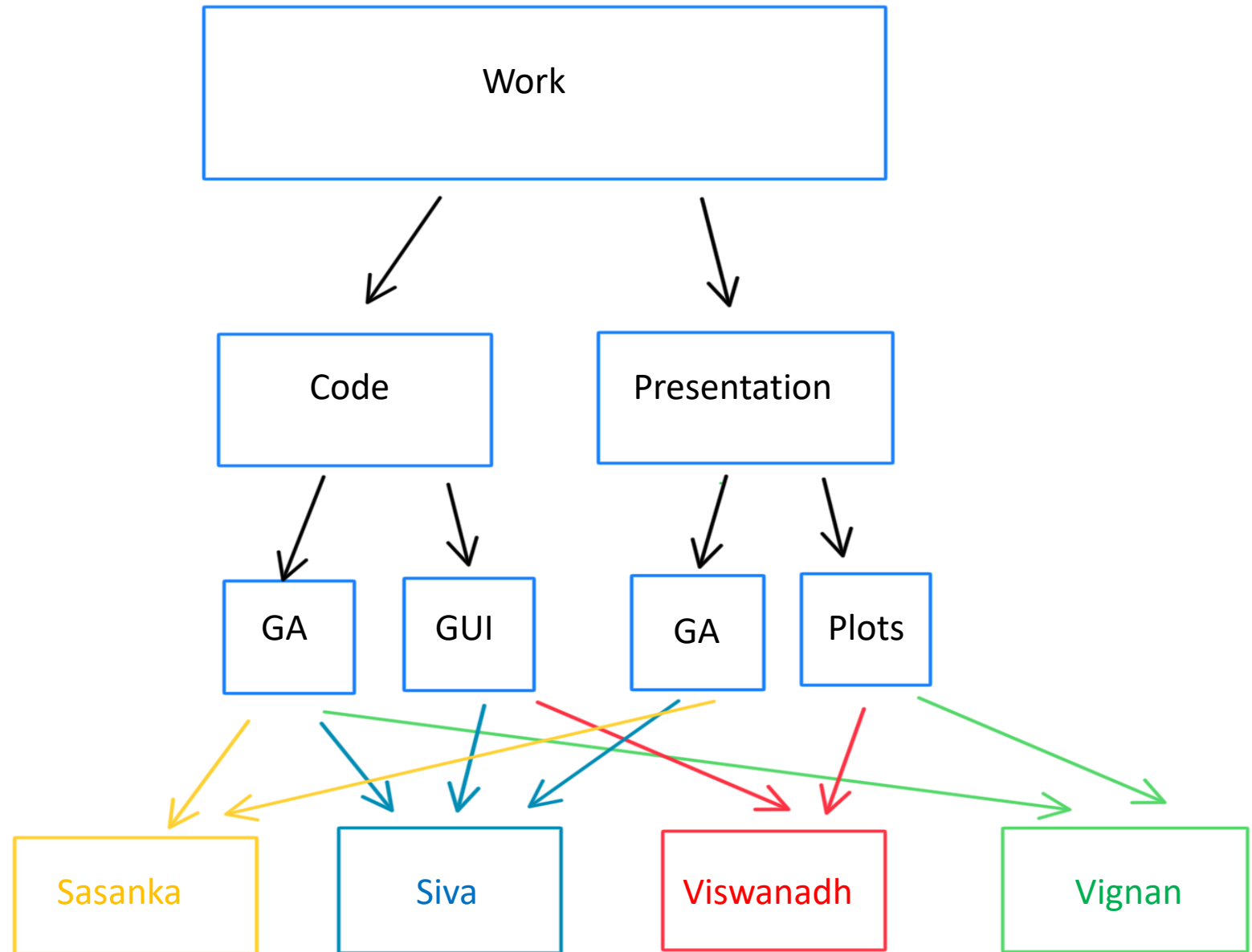


Piece Size=36

Population Size=250

# Division of work

---



# References

- [https://openaccess.thecvf.com/content\\_cvpr\\_2013/papers/Sholomon A Genetic Algorithm-Based 2013 CVPR paper.pdf](https://openaccess.thecvf.com/content_cvpr_2013/papers/Sholomon_A_Genetic_Algorithm-Based_2013_CVPR_paper.pdf)
- <https://www.machinelearningplus.com/statistics/mahalanobis-distance/>
- [https://www.tutorialspoint.com/genetic\\_algorithms/genetic\\_algorithms\\_fitness\\_function.htm](https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_fitness_function.htm)
- [https://www.tutorialspoint.com/genetic\\_algorithms/genetic\\_algorithms\\_mutation.htm](https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_mutation.htm)
- <https://www.youtube.com/watch?v=6DohBytdf6I>



Thank You

---