

A Genetic Algorithm-Based Solver for Very Large Jigsaw Puzzles

Project Proposal

Project ID: 20

Team Name: Dip chick

Team Members:

- Gunjan Gupta 2019111035
- Shivaan Sehgal 2018111026
- Akash Verma 2018101011
- Vedant Mundheda 2018112006

Github Link : [Link \(https://github.com/Digital-Image-Processing-IIITH/dip-project-dip-chick\)](https://github.com/Digital-Image-Processing-IIITH/dip-project-dip-chick)

Problem Statement

The puzzles are part of everyone's childhood. Solving very large puzzles automatically would lead to advantage of many fields. A vast amount of research on topic has been conducted and the methods proposed before like state-of-the-art solver work for very few large images. The problem with these methods is that they use greedy methods for the estimation function, thus optimize at the local values. This achieve good results but not best for small puzzles leaving scope of improvement. We will try to implement the automated puzzle solver for very large jigsaw puzzles to attempt puzzles fastly with more accuracy.

| Input Given | Result Expected |
|---|--|
|  |  |

Goals and Approach

- **Goal**

The goal of the project is to generate a Genetic Algorithm based puzzle solver for very large jigsaw puzzles such that it doesn't compromise speed and performance on the basis of *piece orientation*¹ and puzzle dimensions.

Note: [1] Here piece orientation doesn't imply rotation.

In this project, we will follow the work of *Dror Sholomon*, *Omid David* and *Nathan S. Netanyahu*.

• Approach

For the implementation, let's understand the Genetic Algorithm first.

Genetic Algorithm

It is the optimization algorithm inspired from the theory of natural selection.

The initial population will be randomly generated (random arrangement of puzzle's pieces), then we'll select some chromosomes to produce offsprings using fitness function. Then the selected chromosomes will perform crossover to have offsprings with the traits of both the parents. Then, the children are mutated and then this become new population and this process is followed until the expected results(correct image) is not obtained.

This is certain that it will give expected output because the fitness function will correctly detect chromosomes containing promising solution parts to be passed on to the next generations.

We'll start with 1000 random chromosomes in the initial population.

Fitness Function

In this, we tend to predict the probability of 2 pieces to be adjacent. To achieve this, we'll find the dissimilarity measure between 2 pixels in the particular placement.

$$D(x_i, x_j, r) = \sqrt{\sum_{k=1}^K \sum_{b=1}^3 (x_i(k, K, b) - x_j(k, 1, b))^2}$$

here, it is calculated for x_j in the right of x_i .

The pieces are compatible when their dissimilarity is minimum.

So the **Fitness Function** is:

$$\sum_{i=1}^N \sum_{j=1}^{M-1} (D(x_{i,j}, x_{i,j+1}, r)) + \sum_{i=1}^{N-1} \sum_{j=1}^M (D(x_{i,j}, x_{i+1,j}, d)) + \sum_{i=1}^N \sum_{j=2}^M (D(x_{i,j}, x_{i,j-1}, l)) + \sum_{i=2}^N \sum_{j=1}^M (D(x_{i,j}, x_{i-1,j}, u))$$

As the down and left can be easily deduced from remaining 2, we only find up and right dissimilarity to reduce the computancy cost.

We'll choose the best 4 chromosomes from the current population.

Crossover

As some pieces might repeat, so we need to think of such crossover function that it transfers best traits but also don't repeat the pieces.

So, we'll start with a piece and place other pieces at the boundaries by the following ways:

- If some piece is at the same position on both the parents w.r.t position, then place it there; otherwise continue.
- If the some piece fulfills the criterion of best buddies, then place it otherwise continue.
Condition for the best-buddies is

$$\forall x_k \in Pieces, C(x_i, x_j, R_1) \geq C(x_i, x_k, R_1) - eq(1)$$

$$\forall x_p \in Pieces, C(x_j, x_i, R_2) \geq C(x_j, x_p, R_2) - eq(2)$$

x_i and x_j are said to be best-buddies if eq (1) and (2) are true simultaneously.

- Randomly choose a boundry and place most compatible available piece there.

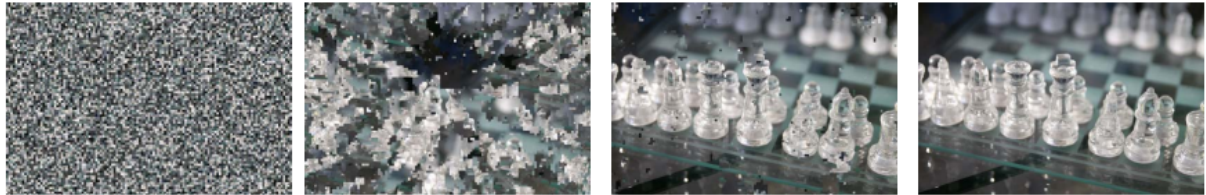
Then restart process for the placed piece.

We are not directly placing pieces agreed by both parents in the final solution. If the placement is due to randomness, that can be disqualified in later stages. And for the best buddy, we can say that the adjacency to the parent makes it more realiable to use. The GA will propogate the overlapping segments to child as it has the high probability of being the correct segment without considering the placement of it.

For the remaining 996 chromosomes, we'll perform crossover with the mutation rate of 5%.

Results

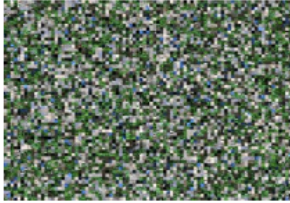


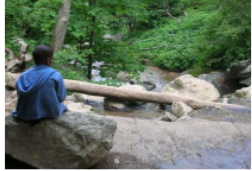
The expected result is the solved image in some generations of the Genetic Algorithm and reach the expected output. The below mentioned example is for 10375 pieces.



We'll run GA for almost 1000 generations. To measure the accuracy, we'll use the neighbor comparison instaed of direct pixel by pixel comparison as some images can be shifted in one particular direction, which will lead to almost 0% accuracy in the direct comparison.



So let's consider case of 5015 pieces puzzle.

| Original Puzzle | Generation 1 | Generation 2 | Final Generation |
|--|--|---|---|
|  |  |  |  |
| Puzzle generated by dividing image into 5015 pieces and randomly placing them but not rotating or scaling them | After Generation one the pieces on the boundaries are getting settled in the right segment | Almost all the pixels start to settle down at the right place but many segments can be seen shifted | As we can see that the results of better than perfect i.e. dissimilarity is less than the correct img |

The results of GA solver is far better performance than greedy algorithms for both smaller and larger images. For some images, the accuracy is 100%.

We will also be comparing the results of the given paper with our implementation for different dimensions of the image.

Dataset

In the paper, there is mention of availability of dataset and results may time. But the link mentioned shows page not found. We have mailed the owners and waiting for the reply.

If we doesn't get any response, we will try to find 3-4 images with different level of difficulty.

Note: Here difficulty can be defined by the variety of textures and colors.

Link: [Dataset](http://www.cs.biu.ac.il/%E2%88%BCnathan/Jigsaw) (<http://www.cs.biu.ac.il/%E2%88%BCnathan/Jigsaw>).

Expected Deliverables

1. **Code:** A working puzzle solver for a given puzzle.
2. **Documentation:** A Readme.md (<http://Readme.md>) with the instructions to run the code.
3. **Requirements:** A requirement.txt to include any missing dependencies.
4. **Sample Inputs:** A zip file containing images used for training.
5. **Sample Outputs:** A zip file containing images obtained as output.
6. **Comparison:** Experiment report for performance comparison with the paper and other methods.
7. **Presentation:** Presentation, description of the project along with instructions to run the demo.
8. **Report:** Detailed report of the code, the outputs obtained and the conclusions from the outputs.

Milestones & Timeline

| Timeline | Milestone |
|-----------|---|
| 5 Nov | Project Allocation |
| 9 Nov | Project Propoal Submission |
| 10-11 Nov | Mentor Review |
| 13 Nov | Look into Genetic Algorithm and implement basic of it |
| 17 Nov | Implement fitness function |
| 25 Nov | Implement Crossover function and Mutation function |
| 27 Nov | Testing and readying deliverables |
| 29 Nov | Code Submission |
| 2 Dec | Project Presentations |

Work Distribution

| Work | Done By |
|-----------------------------------|---------------------------|
| Project Proposal | All 4 members contributed |
| Initial Research | Equal and Independent |
| Fitness Function | Pair 1 |
| Crossover Function | Pair 1 |
| Mutation | Pair 2 |
| Debugging, testing and compilling | Pair 2 |
| Getting Deliverables Ready | Pair 1 |
| Documentation | Pair 2 |

Pair 1: Shivaan+Gunjan

Pair 2: Akash+Vedant