An Implementation of a Genetic Algorithm Based Large-Jigsaw-Puzzle Solver

Joshi Shailesh A (14EE10023)
Electrical Engineering Department
IIT Kharagpur
Email: shailjoshi@iitkgp.ac.in

***Abstract*—In this project I have attempted to reproduce the approach given in a paper entitled "A Genetic Algorithm Based Solver for Very Large Puzzles" by D. Sholomon, O. David, and N. Netanyahu. It is the first paper in a series of paper on the same problem as published by D. Solomon et. al., where this first paper forms the basis of solving the puzzles with some simplifying assumptions. It is claimed by the author that this method of solving jigsaw puzzles has outperformed any other method at that time (2013); as he claims to solve a puzzle with 22834 pieces wheras previous best was possibly below 10000 pieces. The key feature of this algorithm is a heavily customized crossover operator. The algorithm, however, should fall in the category of single objective simple genetic algorithm. In my attempt, I believe that I have successfully implemented the algorithm as proposed by the author and have successfully solved a puzzle of 625 pieces (25x25) in less than an hour using MATLAB.**

Keywords*—*Genetic algorithm, jigsaw, puzzle.

I. INTRODUCTION

The problem of automating the solving of jigsaw puzzles is one that has been around since at least the 1950s. Jigsaw puzzles are image reconstruction problems where the image provided has been cut into non overlapping pieces and shuffled around. The problem is then to reconstruct the original image from the shuffled pieces. In computer science, this problem is proven to be of the 'NP-complete' class. The problem has multiple applications, both in and outside of image reconstruction. Puzzle solution techniques can be applied to broken tiles to simulate the reconstruction of archaeological artifacts. In 2011, DARPA held a competition, with a fifty thousand dollar prize, to automatically reconstruct a collection of shredded documents. Other applications include the molecular docking problem for drug design, DNA/RNA modeling, image based CAPTCHA construction, and speech descrambling.

A. *Previous work*

Several different approaches have been studied and applied in solving jigsaw puzzles. This includes probabilistic, particle filter, dynamic programming, patch transform, greedy algorithm and many other methods.

B. *This project*

In his work, the author of original paper has assumed the knowledge of total number of pieces, final image dimension and that all pieces are available at all times. Also assumed is that the pieces are all rectangle, of same size and are in correct orientation (no rotation or flipping). In my work, I have taken all these assumptions and also asked that the image be divided same number of pieces along its length and width. This requirement is only for programming simplicity and it neither facilitates not complicates the working of algorithm proposed. Fig. 1 shows a jumbled image and its solution is shown in Fig. 2.

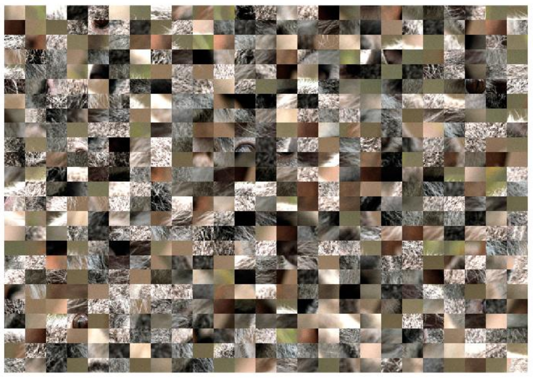

Fig. 1: Randomly generated jumbled image in 625 pieces

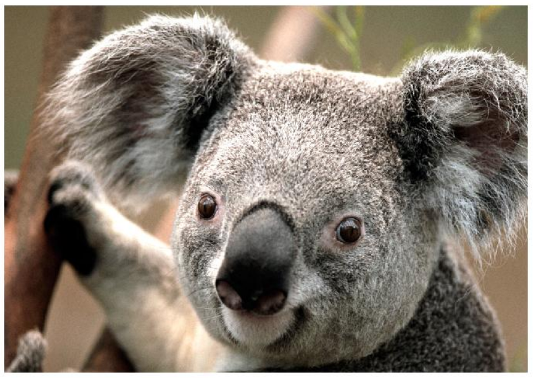

Fig. 2: Solved image using proposed algorithm

II. Technical Details

Like for any genetic algorithms we also need to define our selection, crossover, mutation, fitness calculation and other operations if used. In this solution we do not have any mutation or selection operators. We randomly choose two available parents, send them for crossover and put the child in new generation. This is repeated till new the generation is full. However, we do use elitism. In each generation a few best chromosomes are passed to the next generation without any crossover. A pseudocode for main algorithm as given in the original text is shown below:

Algorithm 1 Pseudocode of GA framework

```
1:  $population \leftarrow$ generate 1000 random chromosomes
2:  for $generation\_number = 1 \rightarrow 100$ do
3:      evaluate all chromosomes using the fitness function
4:      $new\_population \leftarrow NULL$
5:      copy 4 best chromosomes to $new\_population$
6:      while $size(new\_population) \leq 1000$ do
7:          $parent1 \leftarrow$ select chromosome
8:          $parent2 \leftarrow$ select chromosome
9:          $child \leftarrow crossover(parent1, parent2)$
10:         add child to $new\_population$
11:     end while
12:     $population \leftarrow new\_population$
13: end for
```

In order to understand the working, we need to study in detail the fitness calculation and crossover operation. This will involve an introduction to some terms like compatibility, dissimilarity measure and best buddy.

A. *Fitness*

Fitness in any genetic algorithm is the performance measure of the chromosome. In a simple genetic algorithm we want to either maximize or minimize the fitness. Here, the fitness function that we have chosen measures the distance of each piece with its neighbouring pieces. It is defined in terms of compatibility and dissimilarity.

1) Compatibility and Dissimilarity: We define a measure which predicts the likelihood of two pieces to be adjacent in the original image as compatibility, denoted by C. Given two puzzle pieces x_i, x_j and a spatial relation between them $R\epsilon\{l, r, u, d\}, C(x_i, x_j, R)$ denotes the compatibility of piece x_j when placed to the left, right, up or down side of piece x_i, respectively.

Of many possible compatibility measure it was shown in previous work that dissimilarity measure is the most discriminative. Our measure is based on difference between colors of pixels of neighbouring piece's edges. For a correct match sum of difference of pixel colors in edges of neighbouring pieces should be minimum. If x_i, x_j are represented in normalized L*a*b* space by a $K \times K \times 3$ matrix, where K is the height/width of a piece (in pixels), their dissimilarity where x_j is to the right of x_i, is given as:

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

Now, with a fitness measure defined as such, the concern of run time and calculation complexity does arrive because we need to calculate this for each piece, every relation and each chromosome in every generation. However, as we know all the pieces beforehand we can form a look-up-table containing the dissimilarity measure calculated for all pieces with respect to all other pieces for all four relations. However, we know dissimilarity of piece A below B calculated for B is same as dissimilarity of piece B above A calculated for A. So, we can store the dissimilarity only for right and down relations, reducing the size of the look-up-table by half. The final size of the table is $2.(N.M)^2$ where N and M are number of pieces to be placed along the length and width of the image.

2) Fitness of a chromosome: Finally, the fitness function of a given chromosome is the sum of pairwise dissimilarities over all neighboring pieces (whose configuration is represented by the chromosome). Representing a chromosome by an $(N \times M)$ matrix, where a matrix entry $x_{i,j}(i = 1..N, j = 1..M)$ corresponds to a single puzzle piece, we define its fitness as

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

B. *Crossover*

As mentioned previously, this method of solving jigsaw puzzle depends majorly on its heavily customized crossover operation. But before we see crossover we need to define one more term - best buddy.

1) Best Buddy: Simply put two pieces are said to be best-buddies if each piece considers the other as its most compatible piece. The pieces x_i and x_j are said to be best buddies if:

$$\forall x_k \epsilon Pieces, C(x_i, x_j, R_1) \geq C(x_i, x_k, R_1)$$

and

$$\forall x_p \epsilon Pieces, C(x_j, x_i, R_2) \geq C(x_j, x_p, R_2)$$

where $Pieces$ is a set of all pieces of puzzle and R_1 and R_2 are complementary relations like left and right or up and down.

2) Crossover operation: Our crossover follows a kernel building approach. We start from some piece selected and placed at center of child. This is our initial kernel around which we will build an entire new image. Next we need to choose a piece for any of the available empty position around the kernel using the pieces that are not yet placed (available pieces). Placement of this next pieces is decided by three selectors (phases). In first selector, empty neighbours are analyzed one by one, if we find a piece for an empty place, such that its filled neighbours are same in both the parent chromosome and in exact same relation (left, right, up and down), we place this piece. As soon as we place this piece around the kernel in its appropriate position, we start over from finding the available empty neighbours again. However, it may so happen that we cannot find any suitable piece that is in agreement with both parents for any of the available empty places. In that case, we send the list of empty places to second selector.

Second selector is our list of best buddies. We try to find, for any of the empty place, an available piece such that it can be placed next to its best buddies (all the placed neighbours must be best buddies). Again as soon as we place one such piece we start over from finding the empty places again with this hope that this time the first selector (parent agreement) will find an agreement. However, if the second selector also fails to place a piece at any empty neighbour, we go to third selector.

Third selector is the last option to place and so it makes sure it places one piece which avoids the algorithm to get stuck at this place. We simply analyze all the available pieces by replacing them one by one at all the empty places and measure their compatibility. The one with best compatibility wins the position and gets placed. This process is continued till the child is full.

An important thing to note here is that by following this kernel bound building approach the children do not depend on accuracy of positioning of the first piece. A piece placed first can end up at any position when the child is full. Without this feature the entire development of the generations will depend on the accuracy of placement of the first piece.

This way we successfully develop a child from its parents by taking only good and agreed traits from them. The entire crossover operation is explained in the flowchart on the side.

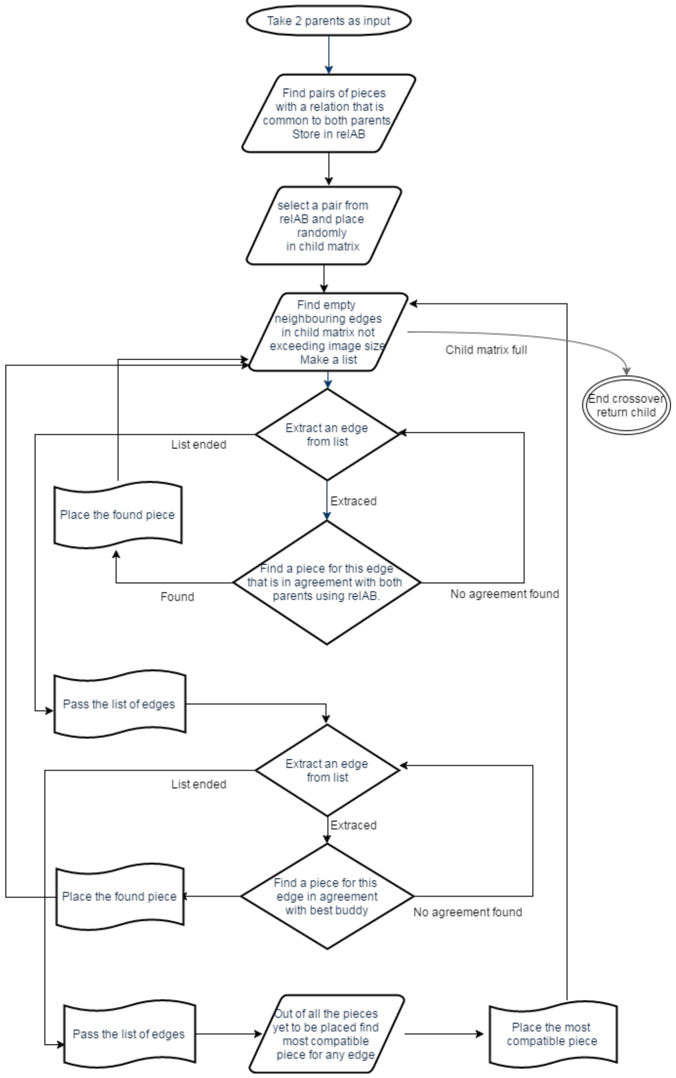
Take 2 parents as input
Find pairs of pieces with a relation that is common to both parents Store in relAB
select a pair from relAB and place randomly in child matrix
Find empty neighbouring edges in child matrix not exceeding image size Make a list
Child matrix full
End crossover return child
Extract an edge from list
List ended
Extracted
Place the found piece
Find a piece for this edge that is in agreement with both parents using relAB.
Found
No agreement found
Pass the list of edges
Extract an edge from list
List ended
Extracted
Find a piece for this edge in agreement with best buddy
Place the found piece
No agreement found
Pass the list of edges
Out of all the pieces yet to be placed find most compatible piece for any edge
Place the most compatible piece

Fig. 3: Crossover operation flowchart

III. Results and Conclusion

Simulations are run and tested on MATLAB for different images and different configurations. In all the cases tested, 100 % reconstruction of the image is achieved. The original work discusses an exception case where some pieces are more compatible with wrong neighbour, resulting in a solution having better fitness (less fitness value) than the original image. However, this only affects the cases where piece sizes are small and number of pieces is very large. In my simulations, I went till 625 pieces only due to time and computational power constraints and thus did not witness the case where a solution having better fitness that of original image is generated.

I am sharing the details of two of my simulations on two different images. The image of a Koala taken from the internet was run with 625 pieces and the image of a temple taken by me was run with 400 pieces.

The best fitness for the 625 pieces was achieved in 7 generations only with complete reconstruction as shown in Fig. 4. Fig. 5 shows an initial random arrangement, Fig. 6 shows an intermediate chromosome and the final image is Fig. 7. This simulation was run with 25 member population size, 1 elite and for 15 generations.

The best fitness for the 400 piece image was achieved in 3 generations only with complete reconstruction as shown in Fig. 8. Initial and final states are shown in Fig. 9 and Fig. 10. This simulation was run with 15 member population, 1 elite and for 8 generations.

An important observation from the plots of best fitness in each generation is that initially during crossover there are very less agreements between the parents and still we have the sharpest descent in the initial phase. This sharp descent can be largely credited to the best buddy based selection in the second phase when the first phase had failed. However, one may say that as best buddy contains a deterministic information for final possible image, it does not truly comply with the philosophy of genetic algorithm. This is there, but one should note that after one huge descent the solution has not yet converged. Although not visible clearly due to the scale of the plots, the fitness continues to improve over next few generation as the pool is filled with good images having slightly misplaced pieces as shown in Fig. 6. This fine tuning is largely credited to the inheritance from the parents.

The original work is one of the most cited and remarkable one in the domain and possibly the most successful one among those who used genetic algorithms to solve the puzzle. More paper in a series were published that discussed the cases of flipped, rotated pieces and initially unkown nuber of pieces.

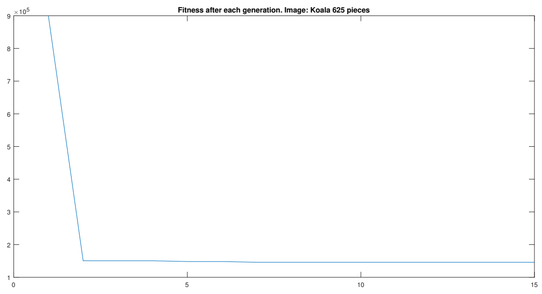

Fig. 4: Fitness of best image with generation

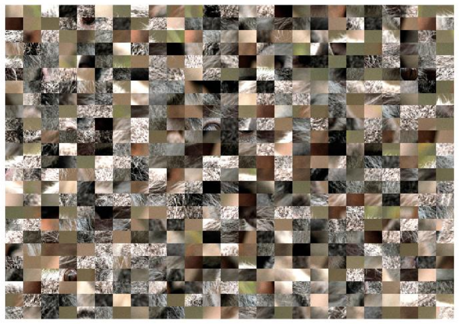

Fig. 5: Initial random chromosome

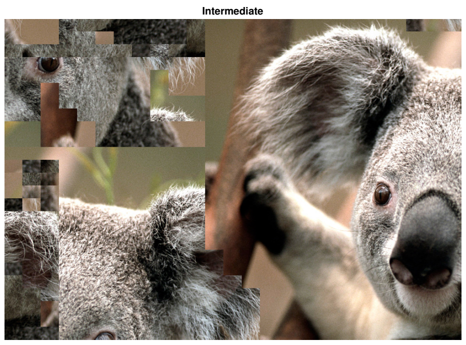

Fig. 6: Intermediate generation

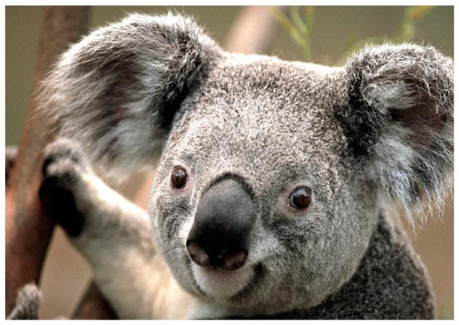

Fig. 7: Final image

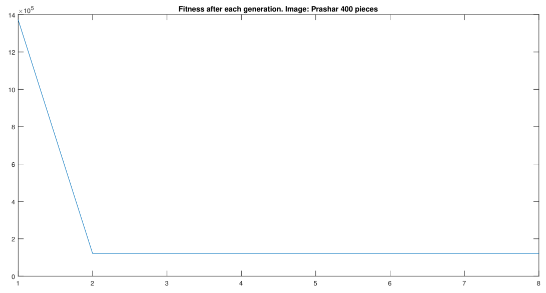

Fig. 8: Fitness of best image with generation

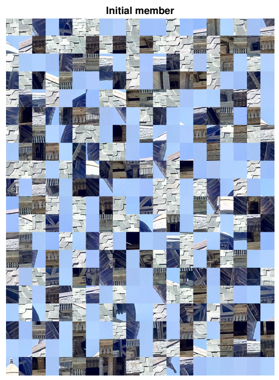

Fig. 9: Initial random chromosome

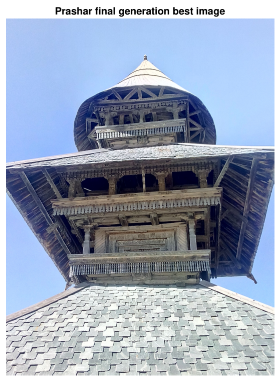

Fig. 10: Final image

IV. Resources

My MATLAB code and it's development can be followed on github on the following url https://github.com/ShailJoshi/GAjigsawpuzzle It contains all the codes and results. Results for the 625 piece puzzle is stored in the file koala_625.mat.

References

[1] D. Sholomon, O. David, and N. S. Netanyahu *A Genetic Algorithm-Based Solver for Very Large Jigsaw Puzzles*