Using Genetic Algorithm to solve Rectangular Jigsaw Puzzles

Kalikivaya Sathvik
Electronics and Electrical Engineering
Indian Institute of Technology
Guwahati, Assam, India - 781039
s.kalikivaya@iitg.ac.in

Poluri Sriya
Chemical Engineering
Indian Institute of Technology
Guwahati, Assam, India - 781039
s.poluri@iitg.ac.in

Budarapu Shashank
Chemical Engineering
Indian Institute of Technology
Guwahati, Assam, India - 781039
budarapu@iitg.ac.in

***Abstract*—In this paper, we present a solver that can solve 'rectangular jigsaw puzzles' using Genetic algorithm by opting a unique crossover method that gives a better child solution by combining two best parent solutions. The solver identifies and merges the distorted puzzle pieces to frame and build the complete desired image. Additionally, we also tried different crossover operators to examine and evaluate their impact on the outcomes of this GA-based solver.**

***Index Terms*—Optimization, Metaheuristic Techniques, Genetic Algorithm, Jigsaw Puzzle**

I. Introduction

Jigsaw puzzles are fun to solve and are very popular all over the world. A typical jigsaw puzzle is made by slicing photographs into smaller pieces and randomly shuffling them. One needs to assemble all the distorted pieces to get back the original picture.

From the literature, it is known that there are two distinct types of jigsaw puzzles: The Canonical Jigsaw and the Rectangular Jigsaw. In this paper, we are only going to discuss the Rectangular ones and describe ways by which the puzzle can be optimized and be finished in a relatively lesser time.

In any type of jigsaw puzzle, we can majorly encounter two notable problems - one being the combinatorial problem, dealing with the number of possible ways to assemble the puzzle pieces; and the other one is the problem of evaluating and figuring out the geometrical and graphical features of combined pieces to know how well they fit together.[2]

In any canonical puzzle, border pieces can be identified and segregated, and every piece has a limited number of neighbour candidates. While in rectangular jigsaw puzzles, every individual piece can be connected to all other pieces. Also, border pieces cannot be distinguished; hence every piece is a candidate in rectangular ones which makes it difficult to solve.[2]

II. Discussion

Say we were to implement a simple brute-force approach to solving a jigsaw puzzle. This means generating all possible assignments of the individual pieces.

If the original image is cut into 'n' segments along each edge, we will have n^2 individual pieces in total, which will lead to (n^2)! possible assignments.

Furthermore, in a given assignment, each piece can be placed in exactly four different orientations, multiplying the number of potential solutions by $4^{(n)^2}$.

Let's call the function of the total number of possible assignments of a puzzle cut into segments along each edge. We then have:

$$f(n) = (n^2)! * 4^{(n)^2}$$

With a simple 2-by-2 puzzle, we have: f(2) = 6,144 Increasing the number of segments by one, we have: f(3) = 95,126,814,720.

Clearly a simple brute-force approach is not viable.

Hence we put our hopes on genetic algorithm, which can selectively find good matches among a randomly generated population at the beginning. Then these good matches will be preserved and combined with newly-found good matches in future populations. The complete solution is found when there are enough good matches.

III. Problem Statement

Firstly, we generate a square puzzle from an arbitrary image using python (using skimage package-related to scikit-learn). We let the program to randomly shuffle the puzzle parts and create a jigsaw puzzle to solve.

Now, the task at hand for us is to come up with an arrangement of these which has the least fitness value.

Between the compatible dimensions, we compute the pairwise distances. Here, we use the cosine distance because it is great for dealing with considerably high dimensional data structures and can help us compute the angle between two edge vectors of high dimensional order.[1]

We then implement GA by performing multiple crossover operations (or variations) and mutations, to get diverse progenies which provide ith an optimized solution for the puzzle.

IV. Problem Generation

For the purpose of this project, we're only considering square pictures cut into square pieces. If the desired image is of dimension m x n (where, m!=n), then we compress the image into a square before generating the puzzle. This means the solver will have knowledge regarding the size of the final solution (in terms of how many pieces there are across an edge). This also means that the method of matching the cut along the edges of two pieces is not viable, since the edges of all the pieces are cut in a straight line.

While representing images as matrices of pixels, each piece of a jigsaw puzzle is reduced to a 3D matrix (it will be of higher dimensions if we consider coloured images instead of grayscale). Furthermore, it can be quite easy to find out whether two given pieces (or matrices) can be matched together or not, computationally.[1]

Using an arbitrary grayscale image, we construct an n x n square puzzle with sides. When we remove the edge configurations, problem becomes relatively more difficult, as we now have to make comparisons based only on image content.[1]

We generate our puzzle by randomizing the indices of each puzzle block.

A. Edge Matching

A small difference between the two columns (or vectors) means a "smooth" transition from one column of pixels to another. And if the two pieces are indeed next to each other in the final solution of the jigsaw puzzle, they should have a such smooth transition. The goal is then to match pieces that have relatively similar edges together.

B. Dissimilarity Measures

The optimal solution to the jigsaw puzzle problems is one that minimizes the pairwise error of all pieces along their respective edges. We compute the distance between each compatible edge separately. The dissimilarity between pieces is determined as the minimum cosine distance between the compatible edge vectors of two pieces. To preserve the order properties of this compatibility, we also return the orientation of the pieces that minimize distance.[1]

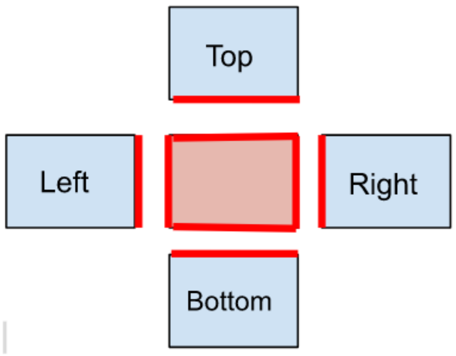

Fig.1: The 4 possible orientations while calculating the piece dissimilarity between 2 pieces

Cosine similarity measures the similarity between two vectors of an inner product space. It is measured by the cosine of the angle between two vectors and determines whether two vectors are pointing in roughly the same direction.
The formula for calculating the cosine similarity is :

$$Cos(x, y) = x.y/|x| * |y|$$

where, x . y = product (dot) of the vectors 'x' and 'y'.

$|x|$ and $|y|$ = length of the two vectors 'x' and 'y'.

The formula for calculating the cosine dissimilarity is :
$1 - Cos(x, y)$

We precompute all pairwise dissimilarities between pieces and store their values and resulting orientations[1]. This will greatly decrease the computational power of our algorithm decreasing redundant work; since all the least distances and best orientations are known, it helps the GA to optimimize the solution fast.

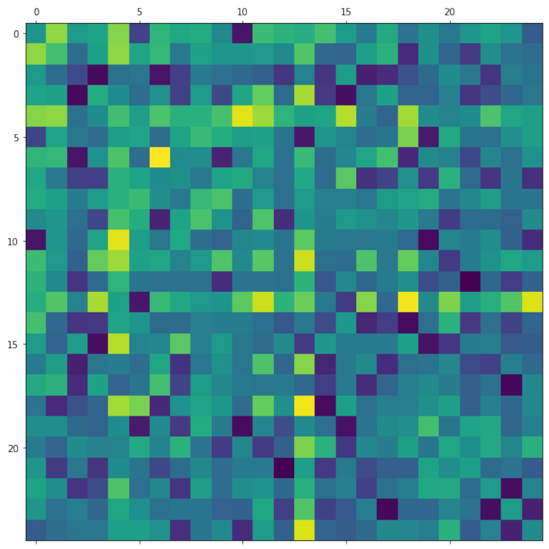

Fig.1: Plot showing the least pairwise distance between every two puzzle pieces of a 5x5 puzzle.

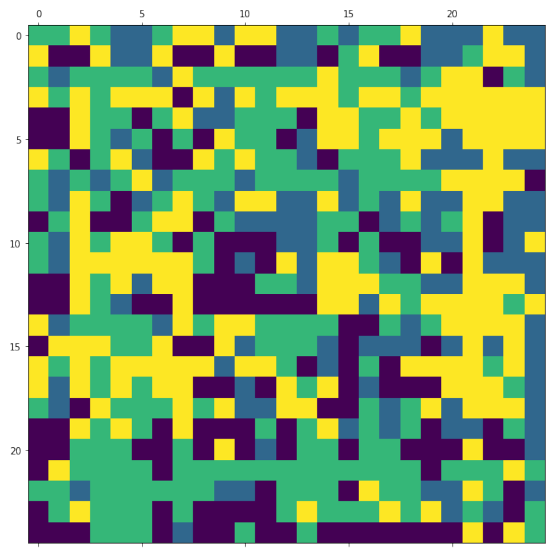

Fig.2: Plot showing the best orientation out of 4 possible or for piece matching of a 5x5 puzzle.

V. Implementation of Genetic Algorithm

A. Genetic Algorithm

Genetic Algorithm (GA) is considered one of the most used and impactful metaheuristic techniques known till now. This stochastic method is inspired by the principles of naturall genetics and selection (Darwin's principle of natural evolution). In this technique, the solutions from which new solutions are generated are called parents amd newly generated ones are called the offsprings. The solution vectors are termed as chromosomes the elements in it are called genes.

Offsprings are generated through ”reproduction” - by selecting good solutions for mating; and by ”variation” - by performing crossover and mutation operations. Selection of a good solution is done after performing the variation.

In this technique, good solutions are retained and bad solutions are eliminated; which means better candidate has more chance to survive in an environment of limited resources. From the demonstrations of some previous works, it is known that GA can act as a very powerful tool and becomes handy for solving difficult search problems with enormous solution spaces.

B. Problem Representation

Jigsaw puzzles may also be considered as an extension to a classical Travelling Salesman Problem, whereby we seek a minimally expensive ordered path through a set of positions.

Genetic Algorithm represent solutions to an optimized problem as “Chromosomes”. In this case, we generate random arrangement of each puzzle piece along the set of n x n positions.

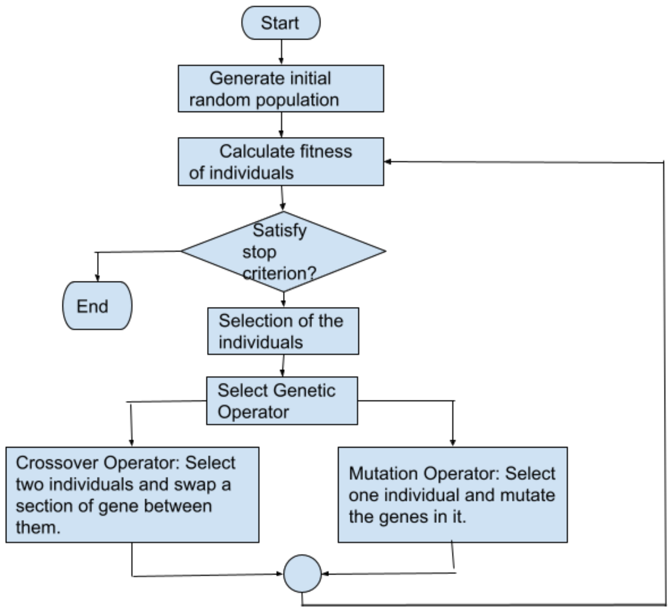
Start
Generate initial random population
Calculate fitness of individuals
Satisfy stop criterion?
End
Selection of the individuals
Select Genetic Operator
Crossover Operator: Select two individuals and swap a section of gene between them.
Mutation Operator: Select one individual and mutate the genes in it.

Fig.3: Flowchart showing the overview of the algorithm along each iteration.

C. Fitness Function

All genetic algorithms define a ’fitness function’ to evaluate possible solutions and drive the natural selection process.

If we consider a puzzle consisting n x n pieces, then we represent a chromosome (or solution vector) by an array of length n^2, in which every gene corresponds to a piece number,which essentially denotes an assigned number that is allotted according to the initial location of genes in the puzzle. Thus, each chromosome represents a complete solution to the jigsaw puzzle; it gives an array with optimized locations of all puzzle pieces.

Every chromosome comes up with a proposed location array for the puzzle problem. Since there is no assumption made about the original image in the problem variation at hand, it is impossible to assess with ease if the absolute locations of the puzzle pieces are correct. Instead, it computes the pairwise compatibility of each pair of neighbouring components.

Our fitness function measures the total dissimilarity of a given configuration. Let the nxn matrix formed by reshaping the chromosome be ‘a[n-1][n-1]’.Then the fitness value of a chromosome is determined by the following equation:

$$\sum_{i=0}^{n-1}\sum_{j=0}^{n-2} Ke[a(i,j),a(i,j+1)] + \sum_{i=0}^{n-2}\sum_{j=0}^{n-1} Ke\Big[a\big(i,j\big),a\big(i+1,j\big)\Big]$$

Where, Ke[m,n] represent the least piece dissimilarity between the pieces m,n.

D. Initialization

We begin by establishing a population of random solutions. These will be evolved over many generations by our GA to identify the optimum solution to the puzzle configuration problem.[1]

TABLE I
PARAMETERS

Table	Parameters
Population Size(Np)	1000
Dimension of the problem	2
Size of the population matrix	1000 X 25
Size of the Chromosome array	25 X 1
Crossover probability	Pc
Mutation probability	0.05
Maximum iterations	400

We do not fix a Crossover Probability (Pc) because crossover occurs until all the solutions are filled in the next generation matrix after retaining, randomly selecting, and mutating. It varies after every cycle.

Pseudocode of GA Framework used in our test cases

```
1: population ← generate 1000 random chromosomes
2: for generation_number = 1 → 400 do
3: evaluate all chromosomes using the fitness function
4: new_population ← NULL
5: copy 400 best chromosomes to new_population
6: while size(new_population) ≤ 1000 do
7: parent1(male) ← select chromosome
8: parent2(female) ← select chromosome
9: child ← crossover(parent1, parent2)
10: add child to new_population
11: end while
12: population ← new_population
13: end for
```

E. Crossover and Mutation

Crossover describes the process where two parent solutions produce an offspring by swapping ”genetic material”. In problems which are similar but without topological constraints, this may be achieved by random substitutions.

To further improve performance of GA, instead of tournament selection, Elitism is implemented by selecting some percent of the population directly to the mating population by their fitness ranking in current population.

In order to handle the topological constraints of a jigsaw puzzle, we employ the following procedure for performing crossover[1]:
- If the relative positions of the pieces are agreed upon by both parents, the kid will automatically inherit the same attribute.
- Pick a boundary piece at random, then locate its closest neighbour and set that piece in the boundary position.

This type of crossover stands out as the best one as it gives us an optimized solution in relatively lesser time. As the iterations pass, the identical parents and children go on to the next generation, increasing the popularity of their kind. It means that the best or fittest solution trains other solutions to undergo crossover and mutation to form its reflection. This lead to the more similar solutions which inturn leads to less computational power, as we skip through the solutions if they are same.

The other crossover operations we implemented are:[2]
Self-reversion crossover: In this approach, two random locations on a chromosome are chosen. The genes between them are now in reverse order.
One Point Self-crossover: In this approach, a random location is chosen. After that, genes are switched around it.
Two Point Self-crossover: In this approach, a chromosome’s two nonoverlapping fragments are chosen at random. Afterward, their genes are switched.

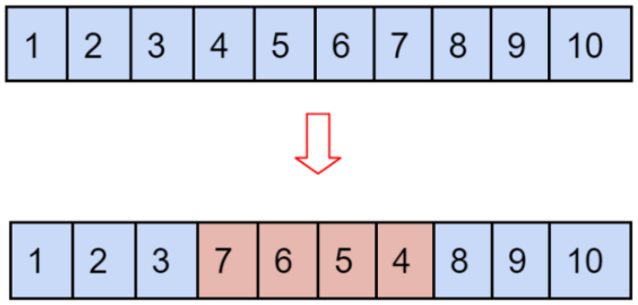

Fig.4(A)

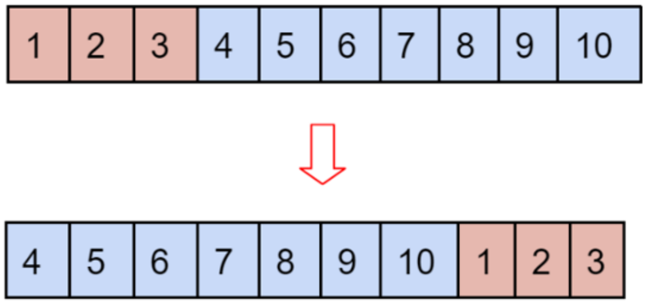

Fig.4(B)

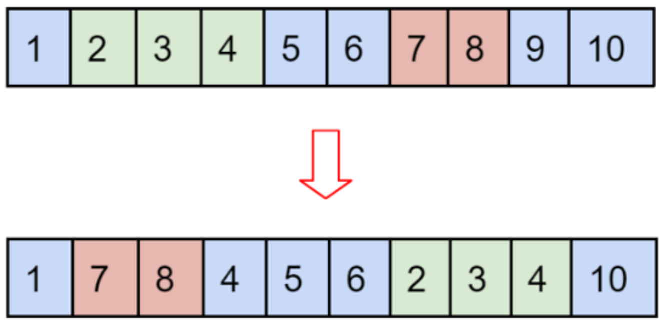

Fig.4(C)

Representation of multiple crossover operations used. Fig.4(A): Self-reversion crossover, Fig.4(B):One point self-crossover, Fig.4(C): Two point self- crossover

While mutating, only individual genes of a chromosome can be altered, so we might encounter an instance where we could loose a better solution on mutating. So, instead of changing a gene and taking the risk of pulling a good piece out of it's position where we might need it, we roll the solution. It basically means we shift it in the chromosome, so instead of being broken apart, continuous pieces are most likely to stay together and then get iterated.

As we iterate, though we get the correct solution for most solutions, but to few solutions, we observed that few portions of the image end up as vertical slices and not combining. To overcome this, we rolled along the columns of the nxn matrix of the chromosome to reach the correct solution.

F. Evolution Cycle

This main steps that we usually do in the evolution cycle are: Evaluate an entire population fitness function, then select a subpopulation and perform crossover to develop the next generation. This is the master function that governs the interactions of all other evolutionary sub-functions.

In our case, we evaluate the fitness of the initial or previous population, and sort the members by their fitness. We then retain any desired percent of the population and take the most fit ones into the next generation, and kill off the other solutions, which then become the parents. We randomly select some of the parents to increase the genetic diversity in the population. We also set a mutation probability to mutate some random individuals. We iterate, crossover and then create a next generation and keep doing it until we exhaust the total number of generations.

VI. Results and Conclusion

For our project, we generated twenty-five different puzzles and tested the algorithm on them. We solved them for same puzzle size and population size and performed all four crossovers and mutation operators described above. We defined the stopping criterion as running 400 cycles; but noticed that a major share of them were solved in less than 100 cycles; only a few took more than 200 cycles.

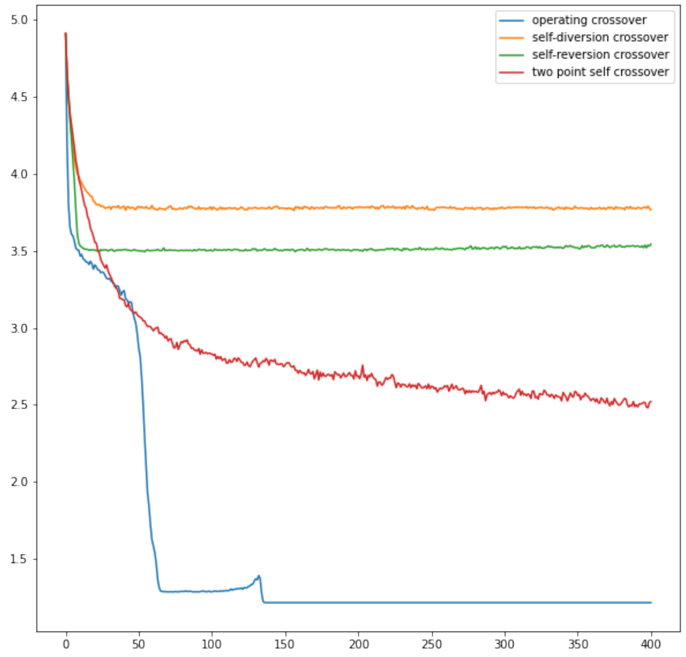

Fig.5: Plotting showing fitness curves using different crossover operations

From the plot above, it is very clear that the efficiency of the working of a genetic algorithm depends very much on the type of crossover and mutation operators that we chose. We can notice that the crossover that we used by retaining the pieces similar to both the parents works more efficiently than the other three crossover operators. It gives a proper and optimized solution in relatively less time than others and also requires less computational evaluation.

The graph shows that the the operating crossover has the least fitness value in less number of iterations, when compared to the other three operators. This happens so because in the operating crossover, once a good solution is found, it trains the other solutions to follow a similar path and converge to an optimized solution at the end. The plateaus and sudden drops in the graph show that some crossover/mutation took place which led to a major improvement in the process. While in the other crossover operators, there is no impact of a good solution on the other solutions. Hence, their performance is relatively lesser than the operating crossover.

Future Improvements

(i) Implement on RGB scale and compare the results between grayscale and non-gray scale.

(ii) Improve the code to solve any generic rectangular puzzle, without restricting to nxn.

(iii) More thorough analysis on the optimal way to calculate threshold for good matches.

(iv) Look for better crossover and mutation operations to further reduce the search and time complexity.

Acknowledgment

We are grateful to Prof. Prakash Kotecha, IIT Guwahati, for teaching us a memorable and interesting course on "Computer-Aided Applied Optimization Techniques". We also thank the teaching assistants, Rajani Kant Baro and Nikhil Anant Purao for guiding us whenever needed.

References

[1] Foxworthy, Tyler. Solving Jigsaw Puzzles with Genetic Algorithms. www.youtube.com/watch?v=6DohBytdf6I.

[2] Ghasemzadeh, Hamzeh. "A metaheuristic approach for solving jigsaw puzzles." Iranian Conference on Intelligent Systems (ICIS), 2014.

[3] Sholomon, Dror, Omid E. David, and Nathan S. Netanyahu. "An automatic solver for very large jigsaw puzzles using genetic algorithms." Genetic Programming and Evolvable Machines 17.3 (2016): 291-313.