Jigsaw Puzzle Sover

Bautista Gómez Juan Pablo
pablobautista240402@gmail.com
Puente Montejano Cesar Augusto (Adviser)

Abstract - Solving a puzzle involves placing a series of pieces in the correct order, for which there are several strategies, such as separating the pieces into groups according to their color, building the edges first, among others. In this paper we propose a strategy to solve a puzzle with square pieces, through the implementation of genetic programming, which, from the generation of syntax trees provides a formula with the best combination of two metrics of image similarity to solve the puzzle.

Keywords –genetic programming, solver, jigsaw puzzle, genetic algorithm

I. INTRODUCTION

Jigsaw puzzles are a worldwide known board game, in which the goal is to correctly place the pieces to reconstruct an image. In the computational field, the resolution of puzzles has been the subject of study for decades, as some concepts of computer vision are covered, such as edge detection and extraction of characteristics of the pieces to find patterns to find the correct order of the pieces, also the problem can be scaled to apply artificial intelligence algorithms to improve the performance and accuracy of the solution. And whose application can be seen mainly in the archaeological field in the reconstruction of archaeological artifacts and reconstruction of documents. As well as in image editing and image reconstruction.

To date there have been several approaches to this problem. In 1998 Wolfson et al. **[1]** proposed an algorithm that classified parts according to their shape. Later, in 2008 Nielsen et al. **[2]** used both edge information and image features. In 2012 Gallaher **[3]** addressed the problem using square pieces, in two different approaches, pieces with unknown orientation and known location and puzzles with unknown orientation and location.

In general, the different proposals for this problem consider two main components, a similarity measure for the pieces or edges and a method to order the pieces. The main contributions of this work are the obtaining of a combined metric from genetic programming to measure the similarity between the pieces, and an ordering method based on quadrants for the ordering of the pieces. The problem is limited to 2x2 puzzles knowing the orientation of the pieces.

II. THEORICAL FRAMEWORK

The proposed solution uses the genetic programming approach, which in turn makes use of genetic algorithms, which were implemented through the DEAP**[4]** library, which offers an already defined genetic algorithm and the possibility of implementing genetic programming and configuring its parameters. These two concepts are explained below.

II.I RELATED WORK

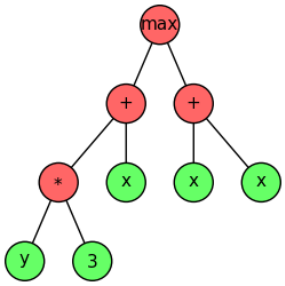
The study of how to solve a puzzle in the computational field has already been studied for a long time, and with it a variety of approaches, some use pieces of different shapes or sizes, however, others have also used square pieces such as Cho te al **[5]** in 2010 where he proposed a patch system **[6]**, which with the support of clusters and linear regression allowing you to solve 432-piece puzzles with more than 90% accuracy. Other approaches add piece orientation as a feature, as Abdullah **[7]** did where, in addition to square pieces, he addressed piece orientation. To do this, he used the comparison of edges using a combined metric, which allows minimizing the number of errors, in addition to comparing the edges adjacent to the piece to find the orientation with the greatest similarity. With this approach Abdullah managed to solve a 30-piece puzzle with 100% success.
The author of the dataset Serhii Biruk **[8]** provides a solution for the data used, where he implements a convolutional neural network (CNN) using the Keras library, where he uses a csv file containing 441 rows, each with a different

order of the pieces, and with which he trains the network in order to predict the correct order of the pieces. Subsequently it uses a similar csv with 111 rows of the same shape each with an order of the pieces, corresponding to the 110 puzzles it intends to solve. This proposed neural network correctly solves 24 puzzles out of the 110 proposed, which represents 21.8% effectiveness.

II.II GENETIC PROGRAMMING

Genetic programming **[9,10]** is an approach that aims to solve problems automatically, generally these solutions are represented by means of syntax trees and that from genetic algorithms allows to perform operations such as mutation and crossover, allowing to reach the optimal solution.
The syntax trees consist of two main elements:

- **Terminals:** They can be constant or variable numbers.
- **Primitives:** Operations that are applied on the terminals. For this work, simple arithmetic functions were used as primitives.



Syntax tree for the function **max(x + 3 * y, x + x) [DEAP]**

The syntax tree represents the function max (x + 3 ∗ y, x + x) where the green nodes represent the terminals and the red nodes the primitives.

II.III GENETIC ALGORITGHM

A genetic algorithm **[11]** is a search and optimization method based on the concept of natural evolution, where the fittest survive. And they allow solutions to be found based on their evaluation through a fitness function, where the crossover and mutation operators are applied to the best ones to generate a new population with better individuals until reaching a solution or reaching a condition such as It may be the limit of generations. And it consists of 3 main elements.

- **Mutation:** Introduces random changes in the solutions to prevent the algorithm from getting stuck
- **Crossover:** Operation that occurs between two solutions to combine them with the objective of generating a better solution
- **Fitness function:** Function that evaluates each solution.

II.IV DATA

The dataset used was obtained from the Kaggle platform **[12]**. The images contained in the dataset correspond to the TV series "Gravity Falls" and the author mentions that they were obtained from Google open sources.

The dataset contains 109 images, which are divided into three subsets according to the size of the puzzles it contains (2x2, 4x4, 8x8). Although in this article we will focus only on the 2x2 size puzzles.
In addition, the data contains the images of the solved puzzles. These are used to compare the puzzles generated by the algorithm with the real solution and to obtain the number of puzzles solved with success.

III METRICS

For the comparison of edges, the combination of the results of two metrics was used, FSIM and DHash, however, following the same quadrant methodology explained later, tests were carried out using different metrics to assess their individual performance, therefore Here we will only briefly explain the operation of the two metrics that will be used in the genetic algorithm.

III.I DYNAMIC HASH (DHash)

This metric **[13]** converts the image to grayscale and reduces the size of the images to 9x9. An average value of the color intensity of the image is calculated and compared with all the pixels in the image generating a series with 0 and 1, from which a hash for the image is generated. In this case a hash is generated for the edges of each piece. The library allows to determine the size of the hash to be generated in a range from 8 to 16, where a size of 16 represents takes more features of the image to generate a more accurate hash but consequently takes more time to generate the hash.

To calculate the similarity between the edges, a subtraction is performed between the two hashes obtained, called hamming distance, where originally a value closer to 0 indicates that the images are more similar to each other. However, because the genetic algorithm takes into account also the FSIM metric, the value was normalized to a range from 0 to 4, where a value closer to 4 indicates that there is a better similarity between the edges.

III.II FEATURE SIMILARITY INDEX (FSIM)

This metric use of other methods to identify the characteristics of the image. As a first step the method makes use of Gradient magnitude (GM) and Phase Congruency (PC) techniques to obtain the image characteristics. Which are explained as follows

A. Phase congruence

Phase congruence **[14]** is a similarity method based on the calculation of the Fourier transform to obtain those points where there is a maximum phase, which represents important features in the image, such as edges, among others. The result is a map of image characteristics. This method is applied to both images, where a map is obtained for each image.

B. Gradient magnitude

Gradient magnitude **[14]** is often used to obtain the edges of the image from a convolution mask. Some methods such as sobel or the prewitt operator can be found here. However, in FSIM the scharr operator is used, which provides better performance. As a result, an image with the edges of the image is obtained.

FSIM gives greater importance to the values obtained by the PC method, however, it relies on GM to obtain better results. The result of the metric is calculated from the following formulas:

$$S_{PC}(\mathbf{x}) = \frac{2PC_1(\mathbf{x}) \cdot PC_2(\mathbf{x}) + T_1}{PC_1^2(\mathbf{x}) + PC_2^2(\mathbf{x}) + T_1}$$

$$S_G(\mathbf{x}) = \frac{2G_1(\mathbf{x}) \cdot G_2(\mathbf{x}) + T_2}{G_1^2(\mathbf{x}) + G_2^2(\mathbf{x}) + T_2}$$

Where T_1 and T_2 are a positive constant wich depends of the dynamic range of phase congruency values in S_{PC} or dynamic range of gradient magnitude values in S_G

Subsequently, the metrics are combined into a single metric called S_L

$$S_L(\mathbf{x}) = S_{PC}(\mathbf{x}) \cdot S_G(\mathbf{x})$$

SL contains the similarity in different points of the image, and to give a weight to its importance the following formula is used

$$PC_m(\mathbf{x}) = \max(PC_1(\mathbf{x}), PC_2(\mathbf{x}))$$

And finally, the following formula is applied to obtain the value of FSIM

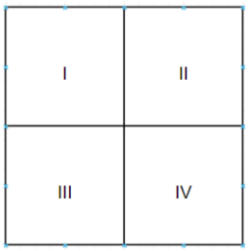
$$\mathrm{FSIM} = \frac{\sum_{\mathbf{x}\in\Omega} S_L(\mathbf{x}) \cdot PC_m(\mathbf{x})}{\sum_{\mathbf{x}\in\Omega} PC_m(\mathbf{x})}$$

To make use of this metric we used the "Pytorch Image Quality" **[15]** library in Python, which follows the above process for the calculation of the result.

IV. PROPOSED SOLUTION

IV.I FITNESS FUNCTION

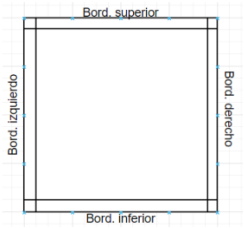
Fitness function is based on the comparison between the edges of each piece of a 2x2 puzzle. Where each puzzle is divided into four quadrants, that is, each piece represents a quadrant, as shown below.



Quadrants for a 2x2 puzzle

To obtain the best piece, the algorithm compares the edges of the pieces, obtaining as a result a floating number, which represents the similarity between the compared edges. Once all the edges have been compared with their respective similarity value, the one with the highest value in a range from 0 to 4 is chosen, where a value closer to 0 indicates that there is very little similarity and a value closer to 4 indicates a higher similarity.

The algorithm takes a piece and places it in each of the quadrants, where in each quadrant it searches for its adjacent pieces. For example, in quadrant I, the algorithm obtains the pieces of quadrants II and III. For quadrant II, it gets the pieces from quadrants I and IV. Up to this point only two pieces have been calculated, the last one is assumed to be in the remaining quadrant, however, the algorithm also calculates the similarity of this piece with any of the adjacent pieces to add it to the total similarity of the proposed puzzle.



Borders of a piece

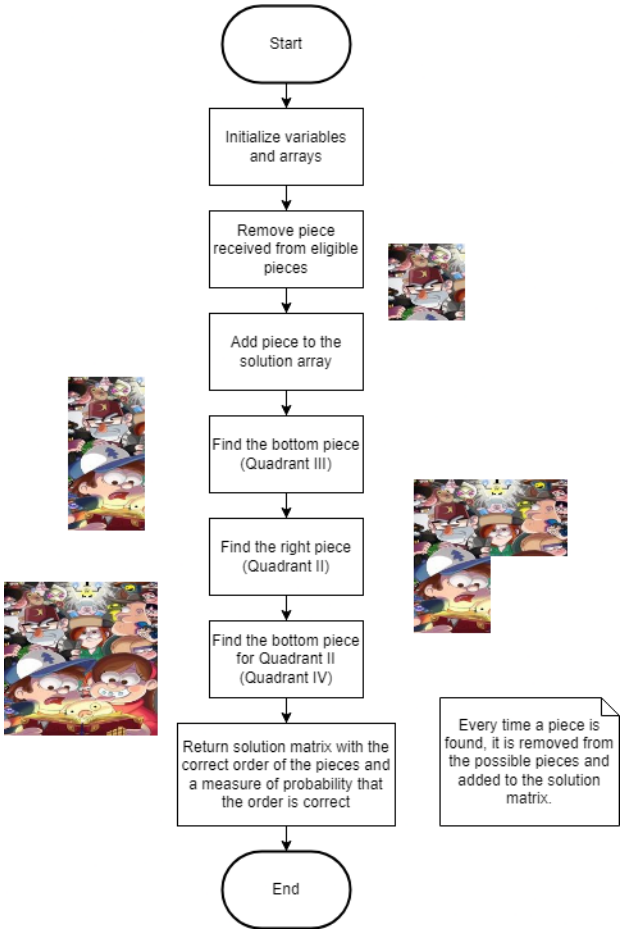
The following algorithm represents how a quadrant found the correct order of the pieces

Algorithm 1. quadrant_I

1. **Require:** Piece, borders
2. **Result:** A list with a order of the pieces and the probability of being the correct order of that list
3. elegible_pieces = [0,1,2,3], correct_order = [0,1,2,3], probablity = 0; accumulated_probability = 0
4. remove the piece received from the eligible pieces list
5. add the piece received in the correct order list in the position 0
6. found_inf ← Compare each piece contained in the eligible pieces and obtain the piece with the best probability
7. found_inf returns a piece and his probability of simility, accumulated_probability sums the probability
8. remove the best_piece obtained from the eligible pieces list
9. add the best_piece obtained in the correct order list in the position 2

10. found_right ← Compare each piece contained in the eligible pieces and obtain the piece with the best probability
11. found_right returns a piece and his probability of simility, accumulated_probability sums the probability
12. remove the best_piece obtained from the eligible pieces list
13. add the best_piece obtained in the correct order list in the position 1
14. found_inf ← At this point we only have only one piece remaining, we use the function to calculate the probability
15. found_inf returns a piece and his probability of simility, accumulated_probability sums the probability
16. add the best_piece obtained in the correct order list in the position 3
17. return the accumulated probability and the correct order

There is a function for each quadrant (I, II, III, IV), each function assumes that the piece it receives as a parameter is in the correct quadrant. The following flow chart shows the steps a quadrant follows to find the correct order of the parts. There is a function per quadrant, each with a similar function, where the objective is to find the adjacent pieces correctly.



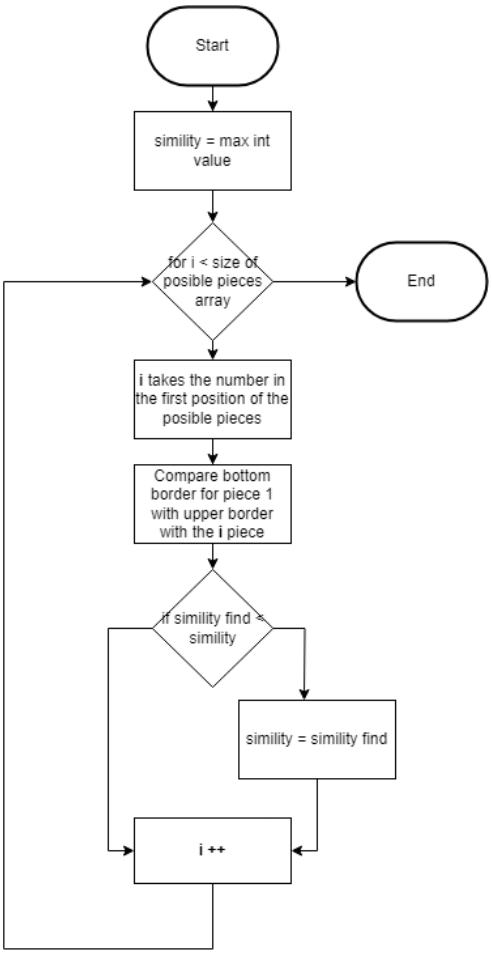
The following algorithm shows how a function compare the edges of the pieces

Algorithm 2. Found inf

1. **Require:** Number of piece in the list of pieces; Piece; list with the eligible_pieces
2. **Result:** The piece with the best similarity and its probability of be the correct piece
3. result = 0; best_piece = 0; umbral = max int size ← umbral is used to actualize the best result in each comparison
4. **for** i in eligible_pieces
5. **if** i different than piece **then**
    a. metric ← metric used to determine the probability than two pieces are similar.

   b. Metric returns a float number that represents the similarity between the compared pieces
   c. **if** result best than umbral **then**
      i. umbral ← result
      ii. best_piece ← i
6. result ← umbral
7. return best_piece, result

The following flowchart shows the steps followed by the function found_inf to find the edge with the highest similarity, the functions found_sup, found_right and found_left work practically the same, only varying in the edges they compare.



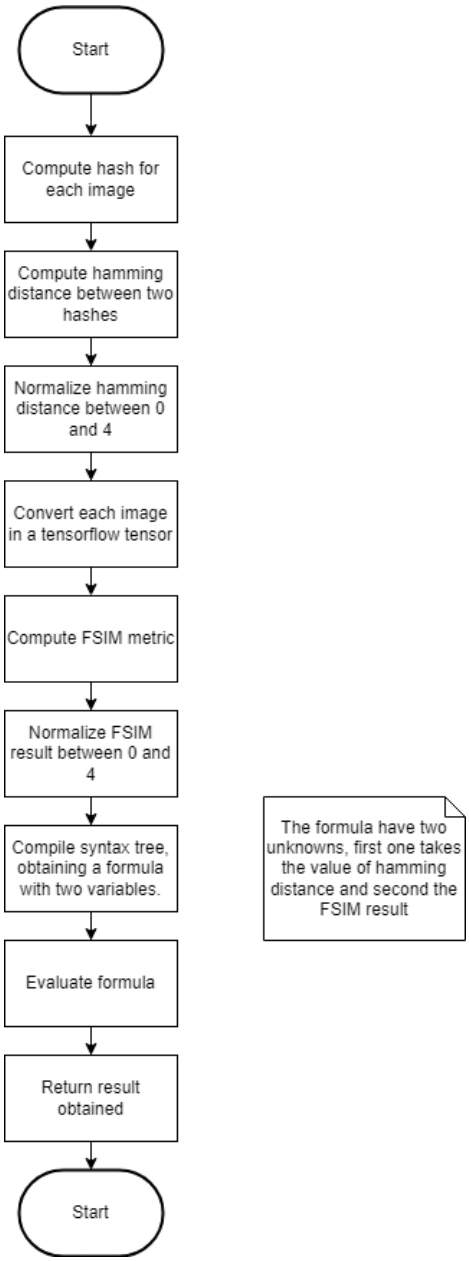
The edge comparison functions make use of a metric to determine which edges are most similar.
The metric is the same for all edge comparison functions, and receives as parameter two images to compare, and makes use of the syntax tree generated by the genetic algorithm to generate the formula with two variables (FSIM and DHash), and with which the similarity between the two images will be evaluated. The following algorithm shows how the metric works

Algorithm 3. Simility metric

1. **Require:** Two images to compare
2. **Result:** Float number that represents the similarity between the images
3. hash_img_1 ← Obtain the hash of the first image
4. hash_img_1 ← Obtain the hash of the second image
5. hamming_distance ← Obtain the difference between the hashes
6. normalized_distance = hamming_distance / hash_size ← normalize the hash in a range of 0 to 4
7. fsim_result ← A function which calculates the FSIM metric between the two images
8. fsim_result = fsim_result * 4 ← Normalize the result in a range of 0 to 4

9. compile syntax tree ← Obtain the formula represented in the syntax tree
10. evluate formula ← Receives the two metrics results and realize the formula, returns a float number which represents the simility
11. return evaluation of the formula

The following flowchart shows the steps of the similarity function to determine the similarity between two edges or images



IV.II GENETIC PROGRAMMING ALGORITHM

As already mentioned, the proposed solution makes use of genetic programming algorithm, where some arithmetic functions were defined as primitives, such as addition, multiplication and the generation of a random floating number between 0 and 1. And for the terminals, two variables were defined: FSIM and DHash. These variables change their value in each evaluation, since they take the value that each function returns when comparing the edges.

This is complemented with the implementation of a genetic programming algorithm where each individual in the population is assigned a different syntax tree, and the fitness function is applied to each individual, applying the

function generated by the tree. The following flowchart shows the steps followed by the proposed algorithm to find the correct puzzle.

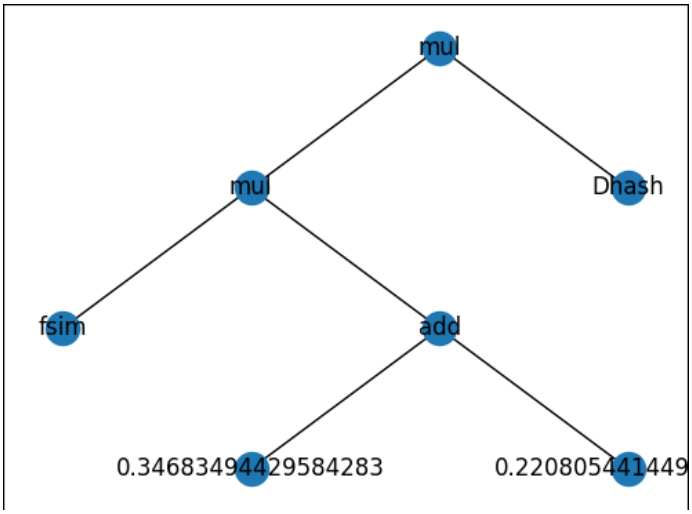
IV.III PARAMETERS FOR DEAP AND GENETIC PROGRAMMING ALGORITHM

For the genetic programming algorithm, a population of 100 individuals and 15 generations was used, with a mutation probability of 0.4 and a crossover probability of 0.6. Each individual evaluates 20 puzzles with a different syntax tree and subsequently compares the puzzles obtained with the true solution, obtaining an integer that represents how many of these puzzles are solved correctly. From where it was defined that the best individuals were chosen according to the result of the fitness function to generate the next population. The choice of these parameters was made through trial and error.

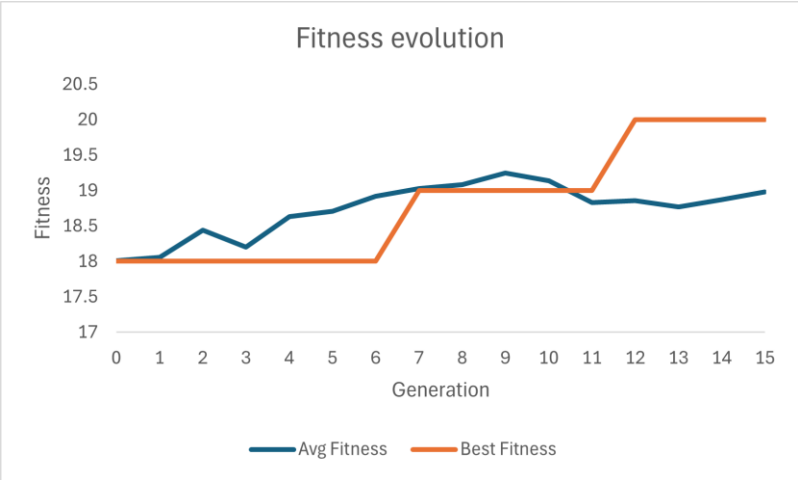
On the other hand, to configure DEAP, among the most important configurations are the selection of primitives, which are operations such as addition, multiplication, and the generation of a random number between 0 and 1. Initially, more operations were considered, such as subtraction, division, exponentiation, however, after various tests, the chosen set showed better results. For the terminals, it was chosen to use DHash and FSIM, where at first it was also intended to use SSIM, but due to compatibility problems it was no longer considered. And for the generation of the trees, a maximum height of 5 was established for the generation of the trees.

V. RESULTS

At the end of the 15 generations, the following formula **fsim * (0.34683494429584283+0.22080544144935366) * Dhash** was obtained, resulting in the following syntax tree which corresponds to the generated formula.



The following graph shows the evolution of the genetic programming algorithm over the generations until reaching the proposed solution. And where you can see that starting from generation 12, the 20 solved puzzles that each individual evaluates in the fitness function are reached.



This formula allows to correctly solve 98 of the 110 puzzles in the dataset. Which represents 89.09% of accuracy

2x2 Puzzle contained in the dataset

Solved puzzle

The following table presents the results obtained with genetic programming and the results obtained using only the quadrant technique with a single metric for edge comparison.

Metric	Number of puzzles solved	Success percentage
Without genetic programming algorithm, using a single metric to solve puzzles		
Coeficent of correlation	87	79.82 %
RMSE	62	56.88 %
DHash (With hash size of 16 bits)	93	85.32 %
SSIM (borders)	44	40.36 %
FSIM (borders)	71	65.13 %

With genetic programming algorithm, using the proposed metric to solve puzzles		
Proposed metric (DHash with hash size of 16 bits + FSIM)	98	89.09 %

As can be seen in the table, using a single DHash metric alone already provided good results with 85% success and using the proposed strategy better results are obtained with 89% success.

VI. CONCLUSION AND FUTURE WORK

In conclusion, puzzle solving is a field of interest for other researchers, as it has implications in photo editing, image processing, archaeology, among others. We propose the use of genetic programming to perform the combination of image similarity metrics to perform a better detection of similarity between the edges of the pieces and find the solution to the puzzle. The obtained results show a 4% improvement over the best result obtained using a single metric to solve the puzzle, which corresponds to the DHash metric. And a considerable improvement of 68% compared to the algorithm proposed by Serhii Biruk, author of the dataset. Using genetic programming in comparison to the predictive neural network proposed by Serhii Biruk

However, to achieve this performance improvement, more computational time was needed, as the genetic algorithm took about 5 hours to evaluate all the puzzles in the dataset and using the DHash metric without the genetic algorithm it takes about 6 minutes to evaluate all the puzzles.

Future work includes the evaluation of other image comparison metrics within the genetic algorithm, the optimization of the genetic algorithm in order to maintain or improve the results obtained in a shorter time, as well as the design of a solution that allows the evaluation of larger puzzles and the performance of the genetic algorithm.

References

[1] Wolfson, H., Schonberg, E., Kalvin, A. et al. Solving jigsaw puzzles by computer. Ann Oper Res 12, 51–64 (1988). https://doi.org/10.1007/BF02186360

[2] Nielsen, T. R., Drewsen, P., & Hansen, K. (2008). Solving jigsaw puzzles using image features. Pattern Recognition Letters, 29(14), 1924–1933. https://doi.org/10.1016/j.patrec.2008.05.027

[3] Gallagher, A. C. (2012). Jigsaw puzzles with pieces of unknown orientation. 2012 IEEE Conference on Computer Vision and Pattern Recognition, 382–389.

[4] Cho, T., Avidan, S., & Freeman, W. (2010). A probabilistic image jigsaw puzzle solver. 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 183–190. https://doi.org/10.1109/CVPR.2010.5540212

[5] DEAP documentation — DEAP 1.4.1 documentation. (n.d.). Readthedocs.Io. Retrieved March 4, 2024, from https://deap.readthedocs.io/en/master/

[6] Cho, T. S., Butman, M., Avidan, S., & Freeman, W. T. (2008). The patch transform and its applications to image editing. 2008 IEEE Conference on Computer Vision and Pattern Recognition.

[7] Abdullah. (2015). Jigsopu: Square jigsaw puzzle solver with pieces of unknown orientation. International Journal of Advanced Computer Science and Applications : IJACSA, 6(3). https://doi.org/10.14569/ijacsa.2015.060312

[8] Biruk, S. (2023, March 16). Jigsaw puzzle. Kaggle. https://www.kaggle.com/datasets/serhiibiruk/jigsaw-puzzle/code

[9] Genetic Programming. (2023, August 14). Saturncloud.Io. https://saturncloud.io/glossary/genetic-programming/

[10] Genetic programming — DEAP 1.4.1 documentation. (n.d.). Readthedocs.Io. Retrieved March 5, 2024, from https://deap.readthedocs.io/en/master/tutorials/advanced/gp.html

[11] What is the genetic algorithm? - MATLAB & Simulink. (n.d.). Mathworks.com. Retrieved December 1, 2023, from https://www.mathworks.com/help/gads/what-is-the-genetic-algorithm.html

[12] Biruk, S. (2023). Gravity Falls Jigsaw Puzzle Dataset [Data set]. https://www.kaggle.com/datasets/serhiibiruk/jigsaw-puzzle

[13] Buchner, J. (n.d.). Imagehash: A Python Perceptual Image Hashing Module. Retrieved March 4, 2024, from https://github.com/JohannesBuchner/imagehash

[14] Zhang, L., Zhang, L., Mou, X., & Zhang, D. (2011). FSIM: A feature similarity index for image quality assessment. IEEE Transactions on Image Processing: A Publication of the IEEE Signal Processing Society, 20(8), 2378–2386. https://doi.org/10.1109/tip.2011.2109730

[15] Image-similarity-measures. (n.d.). PyPI. Retrieved November 3, 2023, from https://pypi.org/project/image-similarity-measures/