FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Yuliya Yamalutdinova

Image Reassembling Algorithms

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the master thesis:	Mgr. Martin Pilát, Ph.D.
Study programme:	Computer Science
Study branch:	Artificial Intelligence

Prague 2022

I would like to thank my supervisor, Mgr. Martin Pilát, Ph.D., for his help and guidance.

Title: Image Reassembling Algorithms

Author: Yuliya Yamalutdinova

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: Mgr. Martin Pilát, Ph.D., Department of Theoretical Computer Science and Mathematical Logic

Abstract: Jigsaw puzzle is a well-known puzzle game that has been around for centuries. However, in addition to entertainment purposes, an ability to reassemble images from pieces has practical applications and can be useful, for example, in restoring torn or cut documents or broken objects in archaeology. Most of the proposed solutions reconstruct the images divided into square pieces. In this work, we propose our solutions for the new types of puzzles with more interesting shapes of pieces, such as rectangles of equal and different sizes, and triangles. The accuracy of our solvers is at the same level as that of the solutions for reconstruction of images from square pieces. Moreover, our solvers, unlike the others, have been tested on images with text as well as on color and black and white photographs.

Keywords: Image reassembling Jigsaw Puzzles Genetic Algorithm

Contents

Introduction

Jigsaw puzzle is a famous puzzle game that has been around for centuries. It is thought to have been invented by John Spilsbury in the 18th century (Williams, 1990). Originally, puzzle pieces were made in the shape of countries on a map, and such puzzles were suggested for use in studying geography. The earliest of them were not more than 50 pieces, but later on puzzles of different shapes and content became a popular entertainment, and today the biggest of them are up to 50,000 pieces[1]. A system that could solve such puzzles automatically would be very interesting by itself, however it has also practical applications, like repairing torn or cut documents or broken archeological objects.

A typical jigsaw puzzle is a picture divided into n pieces of equal or different shapes, and the goal of the puzzle is to put the picture back together. The complexity of puzzles is determined by the image depicted on them, as well as by the shape and number of pieces. The more similar the pieces are to each other and the more there are, the harder the puzzle is to solve. In addition to the well-known classic puzzles, there are also the so-called edge-matching puzzles, the pieces in which have the same, usually square, shape and size. The main feature of such puzzles is that the shape of their pieces does not help to solve them, and the main source of information in this case is the image itself. Erik D. Demaine and Martin L. Demaine examined several types of puzzles, including classic and edge-matching puzzles. They proved that both types of puzzles are NP-complete (Demaine and Demaine, 2007).

Recently, Sholomon et al. proposed a successful solution to square-piece puzzles using a genetic algorithm (Dror Sholomon and Netanyahu, 2014). Their solution is capable of solving puzzles with 22,755 pieces without knowing the dimensions of the puzzle. The goal of our work is to analyze their and other proposed solvers, and to propose our own solutions for puzzles with a more interesting shape of pieces. In this work, we propose solutions for puzzles with rectangular and triangular shaped pieces because in our opinion they have more applications in real life. Next, we propose a solution to a new type of puzzles where the pictures are divided recursively into rectangular pieces, and where the pieces are mostly of different shapes and sizes and are not connected to each other by a whole edge.

Our work is divided into four chapters. In chapter 1, we describe in detail what types of puzzles have been solved by other researchers and what results they have achieved. In chapter 2, we describe in more detail the square-piece puzzles solver proposed by Sholomon et al. because our work is largely based on theirs. In chapter 3, we describe our solutions for each type of puzzle. In chapter 4, we show the results of experiments that were performed on available data as well as on our own data with printed text. In the end, we conclude our work and also suggest how our work can be continued and improved.

[1]`https://web.archive.org/web/20220128074200/https://www.amazon.com/Kodak-Premium-Puzzle-Presents-Largest/dp/B07L8LZRCT`

1. Related work

Many studies have been devoted to the topic of collecting pictures from pieces of various shapes and sizes, but most of them have focused on the reconstruction of color pictures or photographs. No less interesting is the reconstruction of black and white images, more specifically printed texts, which has also been the subject of many studies. In this chapter, we will take a separate look at the studies devoted to these two problems.

1.1 Image reassembling

In their work, Harel et al. point out that all the types of puzzles that have been solved in various studies can be divided into four types based on the geometrical properties of the pieces from which they are composed (Harel and Ben-Shahar, 2021).

1.1.1 Commercial toy puzzles

In the first category are the commercial toy puzzles that everyone is familiar with. The shape of the pieces in such puzzles is often unique, and some researchers use it as the main source of information in their reconstruction (Goldberg et al., 2004). Although some studies also use pictorial data to improve the results. For example, Chung et al. tried to reassemble several toy puzzles of up to 54 pieces using both the shape of the pieces and the pictorial data (Chung et al., 1998). They showed that the pictorial information on the boundaries of the pieces greatly helps in finding a solution. Kosiba, D.A. et al. also used both the shape of the pieces and the pictorial data in their solution (Kosiba et al., 1994). They described each edge of a piece with a set of its features, and used these features to determine whether the pieces would overlap when they are joined. Other studies are similar to these, but all the proposed solutions were only able to reconstruct puzzles with, at best, about 300 pieces (Bunke and Kaufmann, 1993, Nielsen et al., 2008).

1.1.2 Puzzles with square pieces

In the second category are puzzles with square pieces of the same size, which have been studied the most. The peculiarity of such puzzles is that the shape of their pieces does not help to solve them, and the main source of information in solving them is the image depicted on them. Some studies consider a simpler kind of square puzzles when their dimensions and/or orientation are known. A more complex variant is a puzzle with unknown size, double-sided puzzle, or different puzzles mixed into one pile (Dror Sholomon and Netanyahu, 2014, Sholomon et al., 2014).

Most of the proposed solutions are greedy, based on some measure of similarity of the edges of the pieces, but some solutions use other methods as well. Yu et al. proposed a solution based on linear programming, but it relies on knowing the dimensions of the puzzle (Yu et al., 2016). Cho et al. have proposed a probabilistic solver of square puzzles (Cho et al., 2010). In their work, they use a graphical model of a puzzle where each node corresponds to a patch location. Then they maximize a probability function using loopy belief propagation to find the most likely configuration of patches in the graph. They have also proposed a measure of the edges' dissimilarity of the pieces, which we will use in this paper. The difference of edges according to their idea is measured as the root of the sum of the squared pixel differences at the edges of the pieces. The images in this case should be in the normalized L*a*b* color space because the differences between the two colors are easily detected there. Pomeranz et al. have improved the results of the previous work, and proposed their own greedy solver, capable of successfully reassembling large puzzles of up to 3,300 pieces (Pomeranz et al., 2011).

Gallagher et al. proposed an even more successful greedy solver, which looks for minimal spanning tree in a graph whose vertices are pieces and whose edges are the dissimilarities of their edges (Gallagher, 2012). They were able to reconstruct puzzles with 9600 pieces and different puzzles mixed in the same pile, but their solution requires knowledge of the dimensions of the puzzle. They have also proposed another measure of the dissimilarity between the edges of the pieces, which they called *Mahalanobis Gradient Compatibility*. Its goal is to penalize changes in intensity gradients, rather than penalizing changes in intensity, which means that if there is a gradient at the neighbor's edge, then the piece should continue this gradient.

Some researchers also propose successful solutions using evolutionary algorithms. Toyama et al. proposed a solution based on a genetic algorithm, where they used a partial arrangement of pieces as an individual and performed piece exchange and rotation operations on them. Their solver successfully reconstructs small puzzles of 64 pieces (Toyama et al., 2002). Other works propose a solution for puzzles with known dimensions in advance (Sholomon et al., 2017, Guo et al., 2020). They represent the individual as a table of the same size as the puzzle itself, and look for the correct ordering of the pieces in that table. The main operation then is crossover, in which the pieces are gradually connected to each other based on the similarity of their edges, forming clusters. Such solvers are capable of reconstructing puzzles of up to 30,745 pieces.

Senhua Zhao et al. propose a solution using a multi-strategy evolution algorithm (Zhao et al., 2020). They also represent the individual as a table and rely on knowing the size of the puzzle. They then use an elite-based crossover that uses genetic material from both the offspring's parents and the elite, and mutation for both individual pieces and whole lines of pieces, moving them in different directions.

Sholomon et al. proposed a genetic algorithm based solution that can solve puzzles with 22,755 pieces without knowing the size of the puzzle (Dror Sholomon and Netanyahu, 2014). We will discuss their solution in more detail in the next chapter.

1.1.3 Partially constrained modelled puzzles

In the next category are puzzles that have a shape with some limited properties, but not as limited as puzzles with square pieces. We have found 2 studies devoted to this type of puzzles. Shir Gur and Ohad Ben-Shahar focused on solving brick wall puzzles where the rectangular shaped pieces have the same width but different heights, and are connected to each other with a possible shift, which greatly increases the number of possible places where 2 pieces can be connected (Gur and Ben-Shahar, 2017). An example of such a puzzle is shown in Figure 1.1. The authors proposed a greedy algorithm, using previous greedy methods for square pieces, and adding the possibility to shift pieces based on their compatibility.

In the second study, Peleg Harel and Ohad Ben-Shahar proposed a solution for puzzles created by cutting a convex polygon with several straight lines (Harel and Ben-Shahar, 2021). An example of such a puzzle is shown in Figure 1.2. The pieces of such puzzles are also convex polygons, and each edge of a piece has only 1 neighbor of the same length. The complexity of such a puzzle is added by many possible positions of pieces in space. They can be rotated and moved as desired. However, the shape of the pieces and the length of their edges greatly reduces the possible number of connections and helps in solving the problem. To complicate the problem and make it more applicable in real life, the authors added noise to the edges of the pieces by shifting their vertices slightly by a random amount. In their work, the authors proposed a layered reconstruction process based on the pictorial content of the pieces. They divided the problem into two stages. First, they solved the smaller problem of how to arrange the pieces in space while knowing the correct connections. To do this, the puzzle is presented as a system of mass-springs evolving over time. The springs are between the vertices of the edges and if the pieces are far apart, the springs attract them to each other. Then, when the pieces intersect each other, the springs pull them apart. Such a system eventually converges to a minimum of the total potential energy of the spring. In the second stage, the authors propose to connect the pieces to each other in cycles, assessing the correctness of each cycle of connections using the method from the previous stage. The authors considered both pictorial and apictorial puzzles, and evaluated their solutions on puzzles of different sizes. The largest puzzles in their work averaged about 400 pieces.

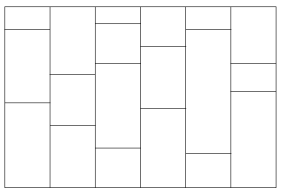


Figure 1.1: Brick wall puzzles

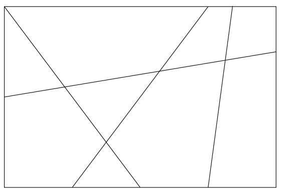


Figure 1.2: Crossing cuts polygonal puzzles

1.1.4 Unrestricted puzzles

In the last category are puzzles with pieces with no shape restrictions. They are usually assembled based on the shape of the pieces and sometimes on the pictorial data. Weixin Kong Benjamin and B. Kimia proposed a solution for 2D and 3D puzzles using curve matching (Kong and Kimia, 2001). Canyu Le and Xin Li proposed a solution for shredded images up to 400 pieces based on convolutional networks (Le and Li, 2018). Michael Makridis and Nikos Papamarkos proposed a solution for even more complex piece shapes based on shape similarity, but tested their solution on smaller puzzles with 173 pieces (Makridis and Papamarkos, 2009). Shengjiao Cao and others proposed a solver for several torn photos, which are mixed together, but they also tested their solution on only a small number of pieces (Cao et al., 2010).

1.2 Reconstruction of shredded documents

Many works have also been devoted to the topic of recovering printed documents. Varad Deshpande et al. have described such works in their review paper (Deshpande et al., 2018). They note that usually in such studies the authors propose to extract some features from the document, such as the size and orientation of the text, the size of the indents, the color of the text and paper. These features are then used in connecting the pieces to each other. Most of the researchers have proposed solutions for recovering manually torn documents or those cross-cut by a shredder machine. Suohai Fan proposed a solver for cross-cut documents using a genetic algorithm, where he also used letter recognition based on a letter database (Fan, 2014). Christian Schauer et al. also proposed a solution for cross-cut documents (Schauer et al., 2010). Their solution is a combination of a genetic algorithm and local enhancement using variable neighborhood search. Ankush Roy and Utpal Garain proposed a probabilistic model for reconstruction of torn forensic documents (Roy and Garain, 2013). In the beginning, their algorithm selects the largest piece and generates probabilities of all combinations between it and other remaining pieces. This step is repeated iteratively, each time using the maximum probability value for the connection. Kantilal P. Rane and S.G. Bhirud proposed a solver for torn documents (Rane and Bhirud, 2011). In their work, they determine the approximate orientation of the pieces based on the text depicted on them, and then they describe new procedures for determining the boundaries and corners of the pieces. Next, they propose to separate the text from its background for easier merging of the pieces.

2. Square-piece puzzles solver

2.1 Problem definition

In this section, we will formally describe the problem we are going to solve. Suppose there is a rectangular image that has been divided into n square pieces of equal size. The pieces do not repeat and do not overlap, their orientation and location in the image are unknown. Our goal is to find the correct neighbors for each piece, such that when we connect them, we get the original picture. We will consider as a successful result a correctly assembled picture, even if it is rotated in the wrong direction. To complicate the task, we will assume that the original dimensions of the picture are unknown.

2.2 Suggested solution

Most of the solutions proposed for square piece puzzles are greedy, which makes them vulnerable to getting stuck in local minima. They are also often dependent on the starting point. For this reason, some studies propose solutions to such puzzles based on evolutionary algorithms. In their works, Sholomon et al. proposed the most successful of such solutions based on a genetic algorithm. They proposed a solver for puzzles with known dimensions in advance (Sholomon et al., 2014, Sholomon et al., 2017), but their most practical solver is the one for puzzles with unknown dimensions or for several puzzles mixed with each other (Dror Sholomon and Netanyahu, 2014).

In their work, the authors proposed to use a classical genetic algorithm augmented with elitism, retaining the 4 best individuals in the next generation, with the following pseudocode:

Algorithm 1 A genetic algorithm with elitism

```
1: $population \leftarrow$ generate 300 individuals
2: for $generation = 1...100$ do
3:     calculate fitness values of all individuals in $population$
4:     $new\_population \leftarrow []$
5:     append the 4 best individuals from $population$ to $new\_population$
6:     while $length(new\_population) < 300$ do
7:         $parent1 \leftarrow$ select parent using roulette selection
8:         $parent2 \leftarrow$ select parent using roulette selection
9:         $child \leftarrow crossover(parent1, parent2)$
10:        append $child$ to $new\_population$
11:    end while
12:    $population \leftarrow new\_population$
13: end for
```

2.2.1 Representation of individuals

The authors proposed a matrix M of size $n \times 4$ as an individual, where an element on the position (i, j) is an edge with which the edge j of a piece i is connected, or *None* if the edge (i, j) has no neighbor. Note that in the correct solution, all pieces except the outermost will have all 4 neighbors, which means that only the pieces on the edges of the puzzle will have the *None* value in the representation. This representation of the individual makes it easier to calculate the value of fitness, and also makes it easier to create new individuals during the crossover.

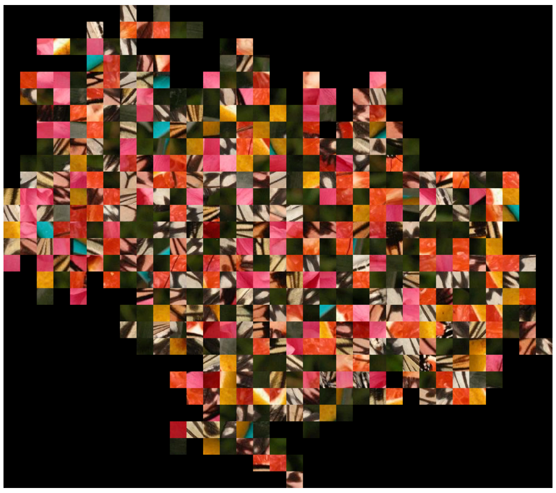


Figure 2.1: Example of an individual in the initial population.

The initial population consists of 300 individuals, each of which is generated by randomly connecting n - 1 edges. All these individuals, however, must be valid, which means that the pieces in them do not intersect each other and each piece is present in the solution exactly once. An example of such an individual is shown in the Figure 2.1.

2.2.2 Selection mechanism and fitness calculation

The authors used roulette selection as a selection mechanism and defined the fitness of an individual M as:

$$\sum_{i=1}^{n} \sum_{j=1}^{4} D(p_{i,j}, M[i, j])$$

where D is a dissimilarity measure, $p_{i,j}$ is the edge j of the piece i and $M[i, j]$ is its neighbor in the individual M. This value should, of course, be minimized. As a measure of edges dissimilarity, the authors used the difference between pixels in the normalized L*a*b* color space proposed by Taeg Sang Cho et al. in their work (Cho et al., 2010), which is defined as:

$$D(x_{i,a}, x_{j,b}) = \sqrt{\sum_{k=1}^{K} \sum_{c=1}^{3} (x_i(k, K, c) - x_j(k, 1, c))^2}$$

where $x_{i,a}$ corresponds to the right edge of the piece i, $x_{j,b}$ corresponds to the left edge of the piece j, k is the index of the pixel in the edge, c corresponds to color channel, K is the size of piece in pixels and $x_i(k, K, c)$ is the pixel of the piece i at the position *(k, K, c)*. Since the fitness function should not only minimize the differences between neighboring edges, but also encourage solutions close in shape to the original image, the authors defined the difference value for an edge without a neighbor as the average difference between all edges multiplied by 2. They found that using a smaller value results in solutions that are more elongated to the sides, while larger values result in solutions where the shape of the rectangle becomes more important than the correct connections.

2.2.3 Crossover

The most important part of the algorithm is the crossover, where most of the work takes place. For it to work properly, however, we must perform several calculations. Before we start the evolution, we will determine the most likely pair for each edge, which is the edge that is the most similar to it. This is done using the same edges dissimilarity measure we described above. Then we will call the edges $x_{i,a}$ and $x_{j,b}$, for which the following condition is satisfied, the *best buddies*:

$$\begin{gathered}\forall e \in Edges, D(x_{i,a}, x_{j,b}) \leq D(x_{i,a}, e) \\ and \\ \forall e \in Edges, D(x_{i,a}, x_{j,b}) \leq D(x_{j,b}, e)\end{gathered}$$

where *Edges* is a set of all edges of all pieces. This notion was introduced by Dolev Pomeranz et al. in their work (Pomeranz et al., 2011) and means that both these edges see each other as the most possible neighbor.

The crossover process itself looks like this. At the input of the crossover, we have 2 parents represented using matrices of size $n \times 4$. We start to build a new solution from scratch, and all the offspring pieces at the beginning have no neighbors. The whole crossover process is divided into 4 phases, in which the edges of the pieces are gradually connected to each other. Note that in order to get an individual in which all pieces are connected to each other and present exactly once, it is necessary to make exactly n - 1 successful connection of the edges. The first crossover phase is responsible for transferring information from parent to offspring. It randomly connects those edges that have the same neighbor in both parents. In the second phase, the most likely pairs of edges, which we call best buddies, are connected, however, provided that the pair is connected in at least one of the parents. The third phase contains greedy connections of edges with their most similar neighbor in a random order. The final fourth phase is needed to complete the solution. It includes random connections of edges until n - 1 connections are made as a total. The crossover process also includes a mutation that is performed in all but the last phase. It consists in the fact that each potential connection of a pair of edges is not made with a probability of p. Summarizing the above, having 2 parents M_1 and M_2 we do the following:

1. Connect all edges (i, a) and (j, b) for which the condition $M_1[i, a] = (j, b) = M_2[i, a]$ is fulfilled. Each of the connections will be rejected with probability of p.

2. Connect all pairs of edges that are best buddies, provided that the connection is present in at least one of the parents. Each of the connections will be rejected with probability of p.

3. Try connecting all remaining edges with their most compatible pairs in random order. Each of the connections will be rejected with probability of p.

4. Make random connections until a total of n - 1 connections is made.

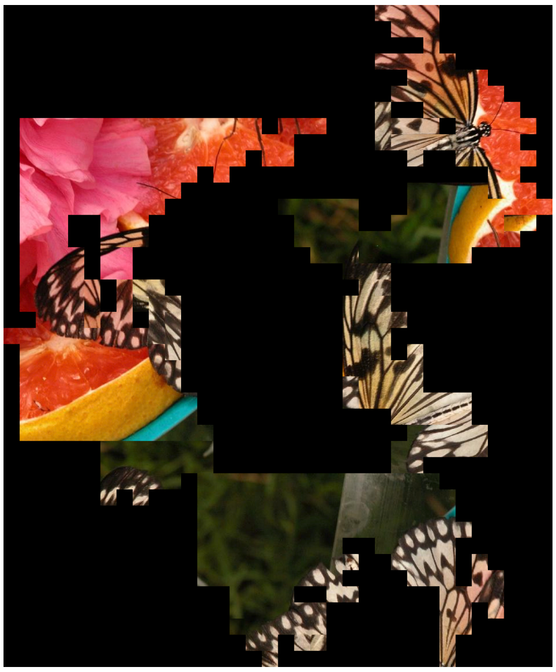


Figure 2.2: The best individual after 2 generations.

An example of the best individual after several generations is shown in the Figure 2.2. It is worth noting that not all the connections we try to make will be successful, since they may create an individual in which some pieces overlap. Only a connection that does not create overlap is allowed. An example of such a failed connection is shown in Figure 2.3.

This solution proved to be successful. The authors tested their algorithm with a population size of 300, 100 epochs, and a mutation probability of 0.001. Their algorithm successfully solved puzzles with 22,755 pieces, as well as several puzzles mixed together. Such successful results inspired us to extend their solution to more interestingly shaped puzzles.

2.3 Our implementation

In this part, we will describe our crossover implementation in more detail, since the authors did not describe in detail some important aspects in their work. We

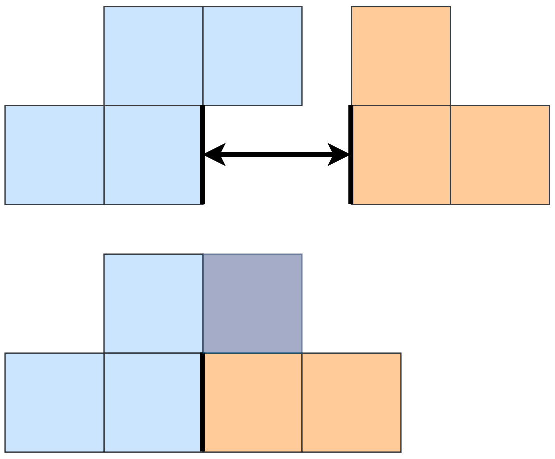


Figure 2.3: Example of a failed connection

propose an implementation in which each piece of the puzzle will be part of a cluster, where a cluster is a set of pieces whose edges are connected to each other. In addition to the indices of the pieces it contains, the cluster also has its own coordinate system in which these pieces are located. We will remember the coordinates of each piece in its cluster, as well as the number of its edge that faces up. At the beginning of the crossover, each piece is in its own cluster of size 1, and then all the pieces are gradually combined into larger clusters. Our goal is to connect all the clusters to each other into one.

Knowing the structure of all clusters, we can easily check whether a potential connection of edges is possible. The authors describe that having a pair of edges that we want to connect, we may encounter 3 situations. The first situation is when the two pieces to which these edges belong are the only ones in their cluster. In this case, we join the pieces into one cluster so that one of them is above the other. This situation is shown in Figure 2.4. When we create a new cluster, we save the coordinates and top edge number for each piece. Since all the pieces are square and have the same size, we do not need to save the coordinates of their vertices in the cluster. It is enough to represent them as 1×1 squares.

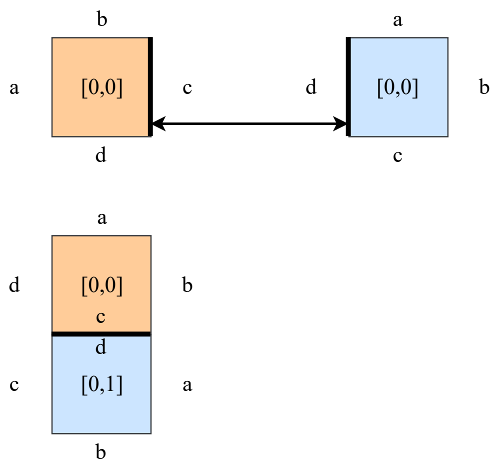


Figure 2.4: Connection of the clusters of size 1

The second situation occurs when the first piece is the only one in its cluster, and the second is not. Then we determine new coordinates of the first piece in its new cluster and thus check whether its place is occupied or not. Then we take all neighboring coordinates and determine new neighbors for the added piece. This situation is shown in Figure 2.5.

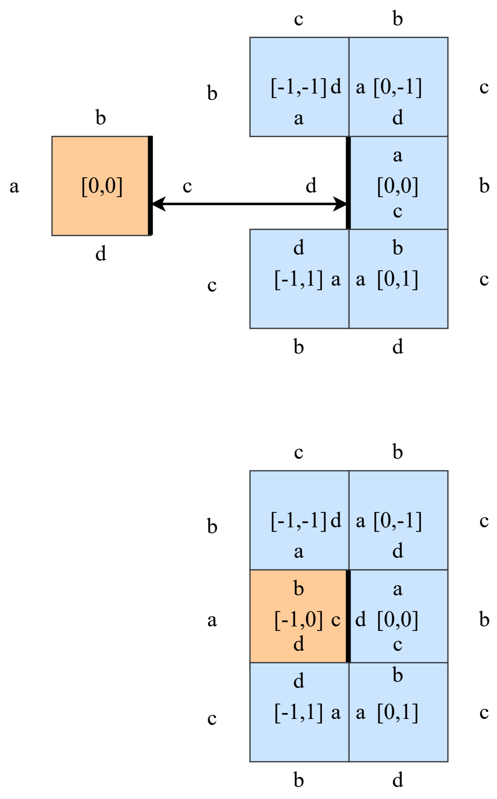


Figure 2.5: Connection of the clusters of size 1 and size 5. First, we determine that the neighbor of the edge d of the piece in the blue cluster at position (0, 0) will have coordinates (-1, 0). Then we check whether the place at position (-1, 0) in the blue cluster is free. Because the place is free, the orange piece becomes part of the blue cluster and gets the new coordinates (-1, 0). This connection is written to the solution. Then we check if the piece has other new neighbors on all 3 sides. In this case there are, and they are pieces at positions (-1,-1) and (-1,1). Therefore, we add 2 more edge connections to the solution.

The third situation is the most interesting one. In this case, both pieces are in clusters with sizes greater than 1. Then we choose the smaller of the two clusters and rotate it around the center of coordinates so that the edges at the place of connection touch correctly. Since the angle of rotation will always be a multiple of 90 degrees, it is not difficult to calculate the new coordinates of the pieces. Next, we determine the coordinates that the piece from the smaller cluster will have when joined to the larger cluster. By doing so, we determine the shift for all the pieces in the smaller cluster and can move them into the coordinate system of the larger cluster. If two different pieces have the same coordinates, it will mean that we have found an intersection, and it is not possible to make a connection between these two edges. An example of a successful connection is shown in Figure 2.6.

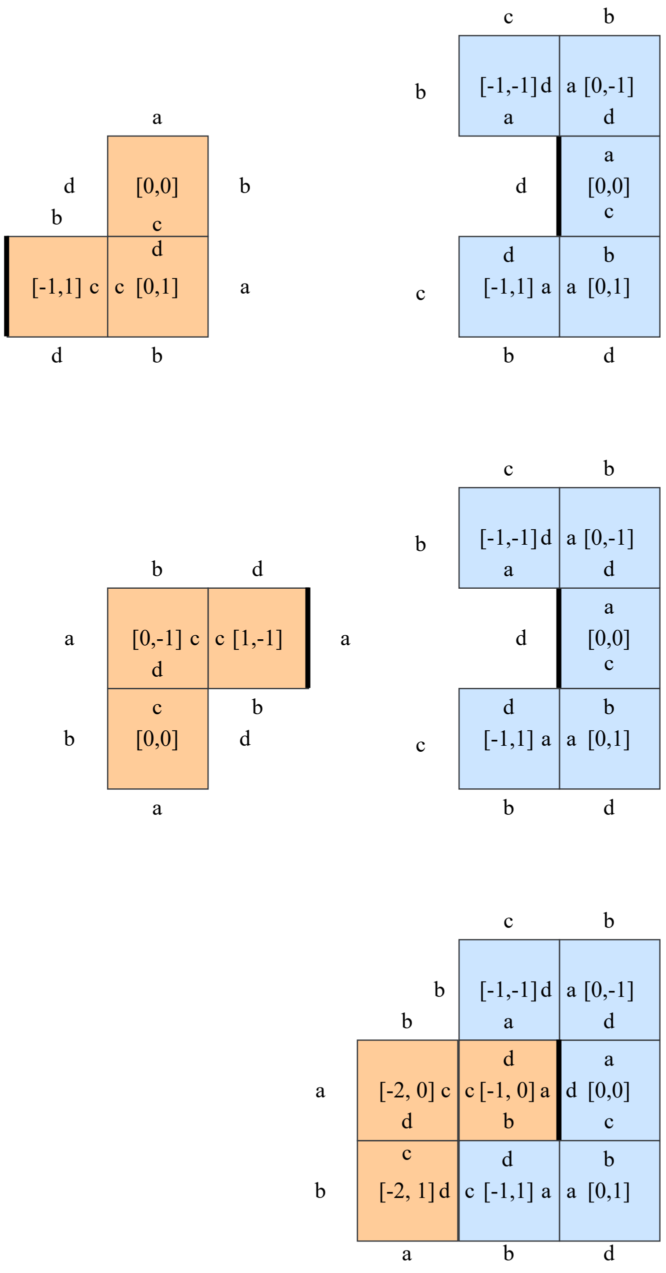


Figure 2.6: Connection of the clusters of size 3 and size 5. First, we identify that the neighbor of edge d of the piece in the blue cluster at position (0, 0) will have coordinates (-1, 0). Then we specify that the edge of this neighbor should be on the right side. The edge a of the piece at position (-1, 1) in the orange cluster, which we will connect, is on the left. To make it right, we rotate the orange cluster 180 degrees counterclockwise. The piece now has coordinates (1,-1). We want the coordinates to be (-1,0) and to achieve that we add (-2,1) to it. We do this for all pieces in the orange cluster and add them to the blue one by one as shown in the previous Figure, checking at the same time whether an intersection of some pieces has been found.

We would also like to clarify how the initial population with random individuals is generated. To speed up the process of creating individuals, we propose to generate them as follows. In the beginning, we choose one of the pieces, and then we will gradually add to its cluster one by one all other pieces, by connecting the random edges that are still available. Since we always connect only one piece to the cluster, this connection will always be successful, and we do not have to check the potential intersection of the pieces.

3. Our solvers

In this chapter, we describe our solutions for rectangular and triangular puzzles, as well as for puzzles recursively divided into rectangles. All of these solutions are based on the solution for square pieces, but some changes were necessary in the representation of individuals, the fitness function, and the crossover.

3.1 Rectangle-piece puzzles solver

3.1.1 Problem definition

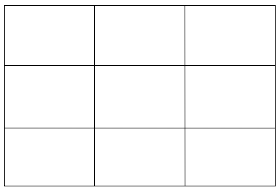


Figure 3.1: Rectangle-piece puzzles 1

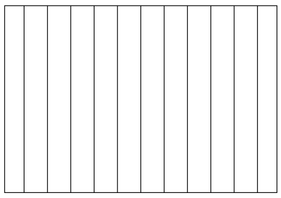


Figure 3.2: Rectangle-piece puzzles 2

Perhaps the simplest extension of the solution for the square pieces is to solve the puzzle divided into rectangular pieces of equal size. Figures 3.1 and 3.2 show 2 variants of such division into pieces. The formal description of the problem in this case remains almost the same as for the square pieces. It is however necessary to add the condition that only edges of the same length can be connected. It is easy to see that in this case the problem becomes easier, since the number of possible candidates for connection decreases by a factor of 2 for each edge. However, at the same time we may encounter a situation where one of the edges of the rectangle is much smaller than the other, as shown in Figure 3.2. Such an edge carries far less information than its larger neighbor, and if we connect the edges in random order, as we do in the crossover for square pieces, we can make many connections based on less accurate data. A generally intuitive rule of thumb is that the more pixels are used to determine edges dissimilarity, the more likely we are to get the correct result.

3.1.2 Crossover variants

In our work, we considered several variants of the crossover. In the first case, we left it almost unchanged, adding the condition that only edges of the same length can be connected. In the second case, we decided to test our theory from the previous paragraph and divided the second and third crossover phases into 2 smaller parts. Instead of connecting the edges with the most compatible neighbor in random order, we propose to give priority to the edges with longer length. To do this, we first connect all pairs of long edges to each other, and only then connect the short ones. Having 2 parents M_1 and M_2, the crossover in this case looks as follows:

1. Connect all edges (i, a) and (j, b) for which the condition $M_1[i, a] = (j, b) = M_2[i, a]$ is fulfilled. Each of the connections will be rejected with probability of p.

2. Connect all pairs of larger edges that are best buddies, provided that the connection is present in at least one of the parents. Each of the connections will be rejected with probability of p.

3. Connect all pairs of smaller edges that are best buddies, provided that the connection is present in at least one of the parents. Each of the connections will be rejected with probability of p.

4. Try connecting all remaining larger edges with their most compatible pairs in random order. Each of the connections will be rejected with probability of p.

5. Try connecting all remaining smaller edges with their most compatible pairs in random order. Each of the connections will be rejected with probability of p.

6. Make random connections until a total of n - 1 connections is made.

Our second idea was to use 2 different mutation values for the larger and smaller edges. We changed the mutation probability values so that with a higher probability, the connection of the smaller edges would not be used. The crossover then looks as follows:

1. Connect all edges (i, a) and (j, b) for which the condition $M_1[i, a] = (j, b) = M_2[i, a]$ is fulfilled. Each of the connections will be rejected with probability of p.

2. Connect all pairs of edges that are best buddies, provided that the connection is present in at least one of the parents. Each of the connections will be rejected with probability of p.

3. Try connecting all remaining edges with their most compatible pairs in random order. If the edge is the larger one, the connection will not be made with a probability of p. Otherwise, the connection will not be made with a probability of $5p$.

4. Make random connections until a total of n - 1 connections is made.

Experiments performed on available datasets showed that the first version of the crossover, which favors large edges, does handle puzzles with very narrow pieces better. However, this solution is not perfect either and has problems with some of the pictures. We will describe the detailed results of the experiments in the next chapter.

3.1.3 Fitness calculation

The next change we made to the calculation of the fitness value, more specifically, we changed the dissimilarity value for edges without a neighbor. Recall that in the algorithm for square pieces the authors proposed a value equal to the average value of the dissimilarities between all edges multiplied by 2. Suppose we use the similar technique, and use the same value for all edges of all lengths. Then we will encounter a situation where individuals with mostly small edges connected will have a lower fitness value than individuals with more large edges connected. This is because of the edge dissimilarity calculation we have chosen, which summarizes the differences between pixels. The fewer pixels we compare, the less edge dissimilarity there will be on average. Whereas, on the contrary, we prefer solutions in which edges of both lengths are equally connected. It follows that a small edge without a neighbor should not add the same value to the fitness as a large edge without a neighbor. We propose 2 solutions to this problem. The first solution is to try to minimize not only the dissimilarities between the edges, but also the area that the solution occupies. To do this, we propose to add the following value to the fitness function:

$$\frac{(X_{max} - X_{min}) \times (Y_{max} - Y_{min})}{n}$$

where X_{max} and X_{min} are the maximum and minimum x values in the coordinates of the cluster that corresponds to the individual, Y_{max} and Y_{min} are the maximum and minimum y values in the coordinates of the cluster and n is the number of images. We have tested different combinations of the dissimilarities between the edges and this value, and found that the best results were achieved using the following fitness function:

$$\frac{(X_{max} - X_{min}) \times (Y_{max} - Y_{min}) \times \sum_{i=1}^{n} \sum_{j=1}^{4} D(p_{i,j}, M[i,j])}{n}.$$

Using this fitness function we have successfully solved most of the puzzles as in Figure 3.2, but for solving puzzles as in Figure 3.1, the original fitness function without adding the area has been more successful.

Our next idea was to calculate the value for edges without a neighbor separately for short and long edges. That is, when calculating fitness, if a long edge has no neighbor, we add to the fitness value the average dissimilarity value between all long edges multiplied by 2. Similarly, if a short edge has no neighbor, we add to the fitness value the average dissimilarity value between all short edges multiplied by 2. This change improved results for all kinds of rectangular pieces, and we decided to leave it in the algorithm.

Any puzzles with rectangular pieces that fulfill the condition that an edge of their pieces can have only one neighbor of the same length can be solved using our solution. We have also used our solver for puzzles divided like a brick wall. That is the way some shredder machines cut documents. An example of such a puzzle is shown in Figure 3.3. We divided its large pieces into two equal rectangular parts and successfully solved it with our proposed algorithm.

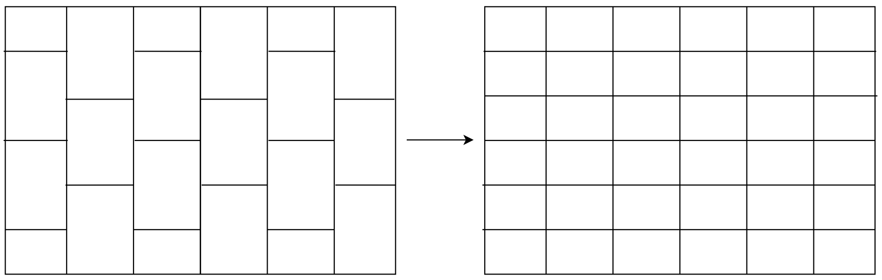


Figure 3.3: Division of brick wall puzzles into rectangles

3.2 Triangle-piece solver

3.2.1 Problem definition

The next shape of pieces for which we have proposed a solution is a right triangle. Examples of such puzzles are shown in Figure 3.4. As in the case of rectangular pieces, we set the condition that only edges of the same length can be connected. A further condition is that a diagonal edge can be only connected with another diagonal edge, and these edges must be parallel to each other. To propose a solution for a given shape of pieces, we had to solve the following two problems:

1. How should the differences between the two diagonal edges be calculated?
2. What coordinates will the pieces have inside the cluster, given that the pieces can be neighbors but, at the same time, be connected not by an edge but by a vertex?

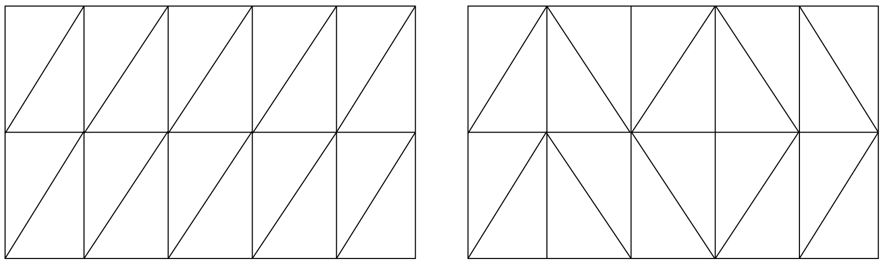


Figure 3.4: Right triangle-piece puzzles

3.2.2 Dissimilarity between the diagonal edges

The triangular pieces our algorithm receives as input are actually rectangles in which half of the pixels are transparent. To calculate the difference between the diagonal edges in such a case, we mapped these diagonal edges to the 2 transparent sides of the rectangle, as shown in Figure 3.5. The resulting edges in this case have the same length as the other non-diagonal edges of the triangle. We propose to calculate the dissimilarity between two diagonal edges as the difference between these mapped edges as follows:

$$D(x_{i,a}, x_{j,b}) = \sqrt{d_{vertical} + d_{horizontal}},$$
$$where$$
$$d_{vertical} = \sum_{h=1}^{H}\sum_{c=1}^{3}(x_{i,H}(h,c) - x_{j,H}(h,c))^2$$
$$and$$
$$d_{horizontal} = \sum_{w=1}^{W}\sum_{c=1}^{3}(x_{i,W}(w,c) - x_{j,W}(w,c))^2$$

where $x_{i,a}$ corresponds to the right diagonal edge of the piece i, $x_{j,b}$ corresponds to the left diagonal edge of the piece j, H is the height of the piece in pixels, W is the width of the piece in pixels, $x_{i,H}$ is the right diagonal edge of the piece i mapped to the right side, $x_{j,H}$ is the left diagonal edge of the piece j mapped to the left side, $x_{i,W}$ is the right diagonal edge of the piece i mapped to the top side and $x_{j,W}$ is the left diagonal edge of the piece j mapped to the bottom side, h and w are indices of pixels on the edges and c corresponds to a color channel.

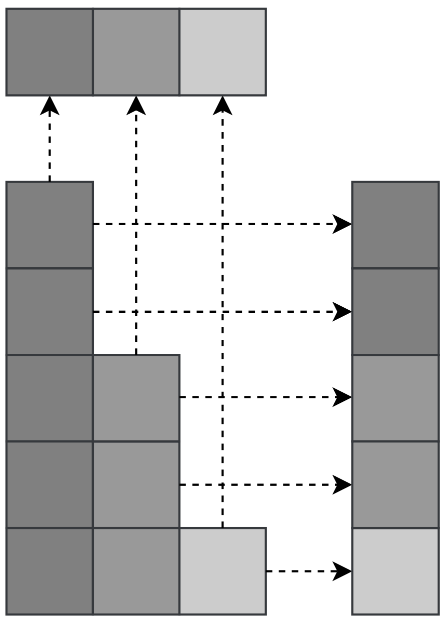


Figure 3.5: Mapping of a diagonal edge to both sides

We have also tested a solution where only one mapped edge, the larger of the two, has been used to calculate the dissimilarity between the edges. However, comparing edges from two sides at once, we got better results. We explain this by the fact that when two diagonal edges are adjacent, some pixels have two neighbors at once, and knowing them, it would be reasonable to calculate the differences with both of them.

When calculating the dissimilarity between diagonal edges, we also encountered the following problem. As we mentioned above, our algorithm receives rectangular pictures where half of the pixels are transparent as input. To support transparency the pictures are saved in RGBA color format, and since we work in the algorithm with pictures in L*a*b* format, we convert them from the

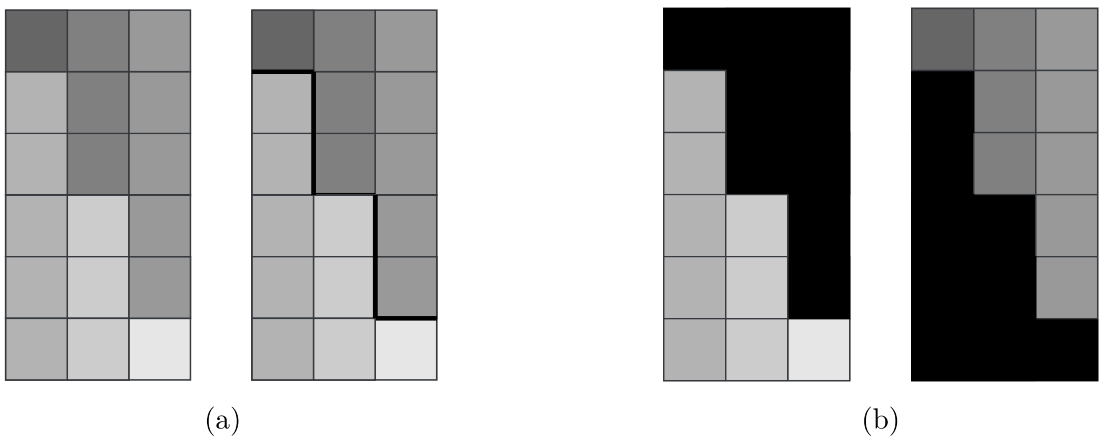


Figure 3.6: The process of splitting a rectangular piece into 2 triangular pieces (a). Transparent pixels when converted from RGBA format to L*a*b* format become black (b).

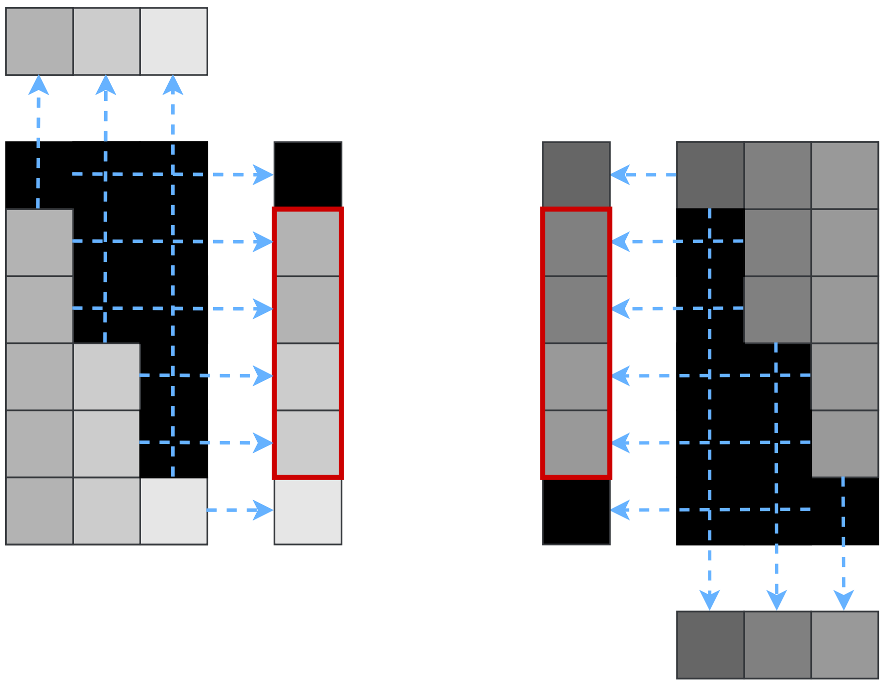


Figure 3.7: After mapping the diagonal edges of both pieces to the left and right sides, there are also black pixels in them, which do not carry any information. So they are cut off at both edges and only the parts of the edges that are highlighted in red are compared.

RGBA format to the L*a*b* format. However, during the conversion process the transparent pixels become black, which creates the problem shown in Figure 3.6. Here we see that when splitting a rectangle in half diagonally, the pixels in the corners of some pieces will be transparent and after conversion they will become black. Thus, the edge which we map to the sides of the rectangle may have black pixels at the corners, which we should not include in the calculation. If such a situation occurs, we cut off unnecessary pixels at the borders of both edges and compare these shortened edges, as shown in the Figure 3.7. If this is not done, images, especially very light ones, will not be reconstructed correctly, due to incorrectly guessed the most compatible neighbors. In general, this problem, if left unsolved, degrades the results of the algorithm by about 30%.

3.2.3 Fitness calculation

It is easy to see that the value of the dissimilarity between diagonal edges will in many cases be higher than the same value for the straight edges, since we are summing up the dissimilarities from both sides at once. In order to prevent our algorithm from starting to prefer solutions in which diagonal edges remain without neighbors due to their large contribution to fitness, we used the same technique as in the case of rectangular pieces. We calculated separately the average value of the difference between all diagonal, vertical, and horizontal edges and took these values multiplied by 2 as the difference value for the edges without a neighbor of the corresponding type. In this case, we expect the algorithm to connect all types of edges equally.

We have also decided to test other techniques used in our solution for rectangular pieces. For example, we compared 2 versions of the fitness function, the original one and the one augmented with the area taken by the solution. In the second case, the algorithm produced worse solutions because the shape of the rectangle in the solution became more important than the correct connection of the edges.

3.2.4 Crossover variants

Since the dissimilarity values for diagonal edges as well as for long-length edges are more reliable, we tested the same approach in the crossover as in the solution for rectangular pieces. In the second and third phases of the crossover, we first connected the diagonal edges and the longer-length edges, and only then the shorter-length edges. By doing this we wanted to prioritize more reliable connections, but this idea was not successful, and we left the original version of the crossover in the solution.

We have also tested a crossover variant, where we gave priority to the edges with the largest difference in dissimilarities between the best neighbor and the second-best neighbor. We assume that the larger the difference, the more likely it is that the best neighbor really is one. The reverse situation, when the difference between the first and the second-best neighbor is small, on the contrary, tells us that the best neighbor may have been chosen incorrectly. To use this trick, we saved the difference between the first and second best neighbors for each edge and sorted them from the largest difference to the smallest. Then in the crossover phases, where we join best buddies or just edges with their best neighbor, we tried to join them in this sorted order. However, this approach also failed and even worsened the results, probably because the individuals became less diverse.

3.2.5 Implementation details

We would also like to mention some details of our implementation that we have found to be important. The first thing we needed to change in our algorithm was the representation of the position of the pieces in the cluster. Recall that a cluster is a set of pieces that are connected together, with their coordinates within this cluster. We find most successful a representation of a cluster, where the position of each piece is determined by the coordinates of the rectangle it is located in, and the side of the rectangle it occupies (left or right). Also note that some triangles are faced up with an edge and others with a vertex. An example of such a cluster is shown in the Figure 3.8.

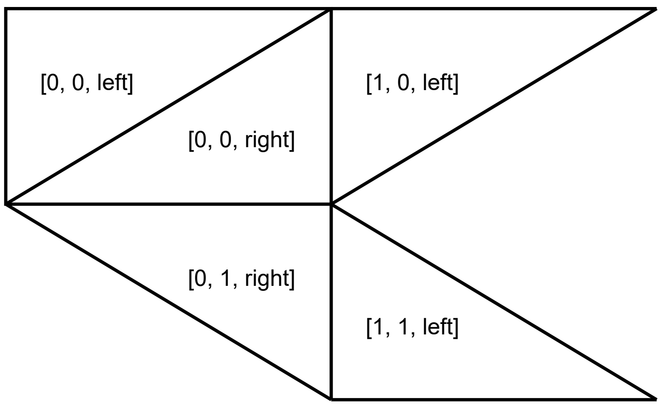


Figure 3.8: An example of a cluster of triangular pieces with their coordinates.

Our next change is made in the last phase of the crossover, where random connections of edges are made until a total of n - 1 connections is made. This is done to ensure that all pieces in the solution are in the same cluster, which also means that all pieces have at least one neighbor. However, we encountered a situation where with a large number of pieces, the last connections of large clusters to each other greatly slow down the whole algorithm. This is because 2 randomly chosen unoccupied edges often cannot be connected to each other. One solution would be to keep and update the edges on the borders of the clusters, since they have the highest probability of being connected. However, we propose a simpler solution, where we allow individuals to be incomplete, so that we set a limit of attempts to connect random pairs of edges. Once this limit is exceeded, we finish the crossover. This technique did not make the results of our algorithm worse, since the fitness of incomplete individuals is higher than that of completed individuals in most cases. Another advantage of this solution is the guarantee that the crossover will always end successfully, because in some rare cases we may not be able to join 2 clusters at all. This will happen when all edges on the borders of one cluster are diagonal and all edges on the borders of the another are straight.

Using our proposed solution, puzzles of more complex shapes can also be solved, provided that their pieces can be divided into right triangles. For example, these can be pentagons and hexagons with edges of equal length, as well as isosceles triangles. We have included an example of such a solution for isosceles triangles in our work.

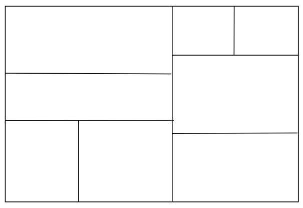


Figure 3.9: Puzzle with random rectangular pieces that is easier to solve

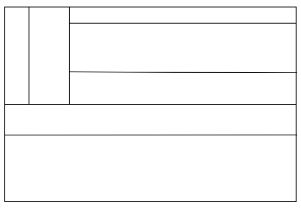


Figure 3.10: Puzzle with random rectangular pieces that is harder to solve

3.3 Random rectangle-piece solver

3.3.1 Problem definition

Our two previous solutions rely on the restriction that the edges of the pieces can only have one neighbor and are connected to each other by a whole edge. A much more interesting task would be to reassemble pictures from pieces without such constraints. Therefore, we propose a solution for a new type of puzzle, which as far as we know has not been solved by anyone yet. Suppose we have a rectangular picture. We will choose a random edge out of four and a random point on it. From that point, we draw a line connecting the edge to the opposite edge and dividing the rectangle into two smaller rectangles. Then we perform the same operation recursively on these two rectangles. To summarize the above, in order to divide a rectangular picture into n pieces, where n is a number which is a power of two, we use the following function:

```
function DIVIDE_IMAGE_INTO_RECTANGLES($image$, $n$)
    if $n == 1$ then
        return [$image$]
    end if
    $result \leftarrow []$
    $rectangle1$, $rectangle2$ ← make a split at random point
    $pieces1$ ← DIVIDE_IMAGE_INTO_RECTANGLES($rectangle1$, $n/2$)
    $pieces2$ ← DIVIDE_IMAGE_INTO_RECTANGLES($rectangle2$, $n/2$)
    extend result with $pieces1$ and $pieces2$
    return $result$
end function
```

Examples of such puzzles are shown in Figures 3.9 and 3.10. Note that an edge in this type of puzzle can now have several neighbors, and the pieces can be of any size. Furthermore, we assume that we do not know the correct orientation of the pieces and the dimensions of the original picture. Our task is to reassemble the picture from the given pieces.

As in our previous solutions, we will keep the group of connected pieces together with their coordinates as a cluster. It is easy to see that puzzles with the proposed type of pieces can be reassembled iteratively if we connect not edges of individual pieces, but edges of entire clusters, provided that these edges have the same length and are the most similar to each other. To do this, at the beginning of the algorithm, each piece will be in its own cluster, and these clusters will be connected sequentially until all of them are connected. In this case, the correct connection sequence would be the sequence in reversed order of how the picture was divided into pieces. However, this solution is not perfect and will not be able to handle a situation where some of the pieces are missing, for example. In this case, one of the cluster edges may not be complete and therefore cannot be connected to its neighbor from another cluster. We propose a more universal solution to such puzzles, which can also handle puzzles in which some of the pieces are missing.

We propose to take the same solver for puzzles with square pieces, based on the genetic algorithm as the basis. However, in this case, we should make a few changes to it. First of all, we have to relax the condition that only edges of the same length can be connected. We propose to connect edges of any length with each other instead, but provided that at least one of the two vertices of an edge coincides with a vertex of another edge. This situation is illustrated in the Figure 3.11. Connecting the edges in this way, we can reconstruct the picture even if some of the pieces are missing. Note, however, that in some cases there may be so many missing pieces that the correct solution would be several clusters that are not connected to each other, which further complicates the problem. Since we do not know in advance whether the pieces of the original puzzle are missing or not, it seems right to try to connect as many pieces as possible together. Therefore, our solution always produces a result where all the pieces are in the same cluster.

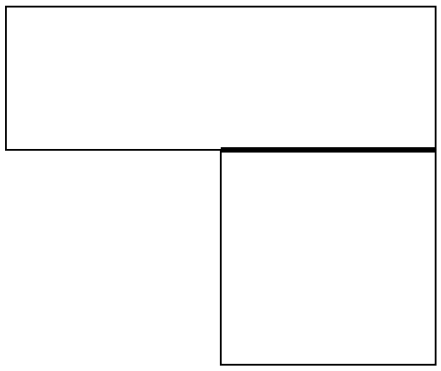


Figure 3.11: Connection of rectangles. The vertex of the first rectangle coincides with the vertex of the second one

3.3.2 Representation of individuals

Since we relaxed the conditions for connecting edges, the number of potential places where edges can be connected has also changed. Note that now any two edges can connect in two ways at one of the two vertices. So in our solution we propose to work not with edges, but with the places near the vertices on the edges. Each edge has 2 such places, which means that we can connect a piece to another piece in 8 places. Based on this, we have changed the size of the individual from $n \times 4$ to $n \times 8$ to save a neighbor for each of the 8 places.

3.3.3 Dissimilarity between the edges

In order to use a genetic algorithm based solution, we should also determine how dissimilarities between edges connected in one of the two places will be calculated. Here we have two problems. The first is the different lengths of the edges we are comparing. The second is the tendency of the algorithm to consider shorter edges as the most similar because they do not carry much information and may be similar to many parts of other edges. Further complicating things for us is the fact that not all edges should, in fact, be connected by contiguous vertices. Some of them will be connected in random places as a consequence of connecting their neighbors to each other.

In general, we would like the function measuring dissimilarity of edges to have the following properties. First, when comparing edges of the same length, we would like them to be marked as more similar to each other, regardless of their length, since they are likely to be correct neighbors. However, the function should not be based only on the common length of the edges, since not all edges of the same length will be actually the best neighbors. Second, when comparing edges of different lengths, we should base it more on the part where they touch and less on the difference in their lengths. In our work, we have tested several variants of this function.

In the first case, we compared only the total contiguous part of the edges and calculated the dissimilarity using the same formula that we used for rectangular pieces:

$$D(x_{i,a}, x_{j,b}, top) = \sqrt{\sum_{k=1}^{K}\sum_{c=1}^{3}(x_i(k, W, c) - x_j(k, 1, c))^2}$$

where $x_{i,a}$ corresponds to the right edge of the piece i, $x_{j,b}$ corresponds to the left edge of the piece j, *top* means that the edges are connected near the top vertex, W is the width of the piece i, k is the index of the pixel in the edge, c corresponds to a color channel and

$$K = min(length(x_{i,a}), length(x_{j,b})).$$

This measurement option, however, proved to be unsuccessful because of the small-sized edges, which the long edges considered to be the most similar. In this case, even the edges of the same size, which should be connected in the solution by a whole edge, were not identified as most similar to each other. Their place was taken by all the same edges of small size.

An obvious solution for comparing edges with large size differences might seem to be to calculate the average value of the differences between pixels instead of the sum of the differences. That is, as follows:

$$D(x_{i,a}, x_{j,b}, top) = \frac{\sum_{k=1}^{K}\sum_{c=1}^{3}(x_i(k, W, c) - x_j(k, 1, c))^2}{K}$$

where $x_{i,a}$ corresponds to the right edge of the piece i, $x_{j,b}$ corresponds to the left edge of the piece j, *top* means that the edges are connected near the top vertex, W is the width of the piece i, k is the index of the pixel in the edge, c corresponds to a color channel, and

$$K = min(length(x_{i,a}), length(x_{j,b})).$$

We have tested this method, but it has been even less successful than the previous one. That is why we returned to the variant with sum of differences, but supplemented it with penalization of great difference of lengths of edges. As a result, the function looked as follows:

$$D(x_{i,a}, x_{j,b}, top) = \sqrt{\left(\sum_{k=1}^{K}\sum_{c=1}^{3}(x_i(k,W,c) - x_j(k,1,c))^2\right) + (L-K) * 2 * D_{mean}}$$

where $x_{i,a}$ corresponds to the right edge of the piece i, $x_{j,b}$ corresponds to the left edge of the piece j, top means that the edges are connected near the top vertex, W is the width of the piece i,

$$K = min(length(x_{i,a}), length(x_{j,b})),$$
$$L = max(length(x_{i,a}), length(x_{j,b}))$$
$$and$$
$$D_{mean} = \frac{\sum_{k=1}^{K}\sum_{c=1}^{3}(x_i(k,W,c) - x_j(k,1,c))^2}{K}.$$

This approach, though not ideal, does not give such an advantage to edges of small length. The proposed function is also effective when comparing edges of the same length. We have chosen the value $2 * D_{mean}$, with which we have complemented the difference for pixels without neighbor, based on experiments on different pictures. This function gave us the best results, so we have chosen it as the main one in our algorithm.

3.3.4 Crossover

The next change we propose to make to the crossover. Recall that one of its parts is the connection of edges which we call best buddies, which means that the edges consider each other as best neighbors. In this solution, we propose to distinguish between two types of best buddies. The first of them will include pairs of edges that are most similar to each other and have the same length. The second type would include pairs of edges that are most similar to each other and may have different lengths. In the crossover we propose to connect pairs of edges of the first type in the beginning, and only then the pairs of the other type, so that we do not lose connections that are very likely to be correct. We compared this variant of the crossover with the unchanged variant, and it gave us slightly better results, so we decided to use it as the main one in our solution.

3.3.5 Fitness calculation

The next important part of the algorithm is the calculation of the fitness values of individuals. As in the previous solutions, we propose to calculate it as the sum of the dissimilarity of each edge with its neighbor. To do this, however, we must decide what dissimilarity value should be used for edges without a neighbor. Recall that this value should gently encourage the solution to be in rectangular form, but it cannot be too large because then the rectangular form takes precedence over the correct connections. We have already noted in the previous two solutions for rectangular and triangular pieces that using the same value for all edge lengths is not efficient. In these solutions, we calculated this value separately for all edge lengths as the average dissimilarity value multiplied by 2, but in the case of recursively divided rectangles, this method does not seem appropriate. To complicate things further, we need to know this value not only for whole edges, but also for individual pixels. This is needed to evaluate the quality of the solution when some parts of the edges are adjacent to other pieces and some are unoccupied, as shown in the Figure 3.12, for example. Therefore, it seemed more appropriate to calculate the dissimilarity not between the edges, but between the pairs of pixels on the borders of the pieces.

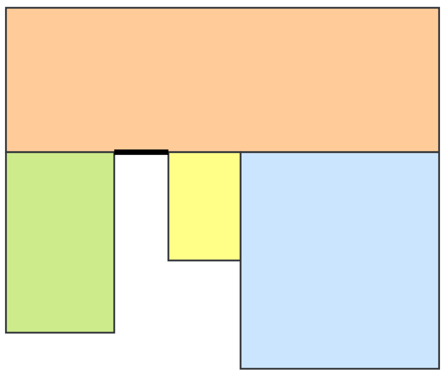


Figure 3.12: The bottom edge of the red rectangle has 3 neighbors (the edges of the blue yellow and green rectangles), but its thick part is not connected to anything.

We have compared two ways of calculating the dissimilarity between pieces. In the first one, we calculated the average value of the difference between all pixels of all pairs of edges and multiplied it by 2. We used this value as the dissimilarity value for pixels without a neighbor. Then for each edge in the solution, we took all its neighboring pixels and summed the differences between all pairs of pixels. The sum of all these values for all edges in the solution was the value of its fitness. This method, however, was not successful because the best individuals were then very stretched out to the sides. This led us to the idea that, unlike previous solutions for rectangular and triangular pieces, in this case we needed to push the solution harder to the rectangular shape. To achieve this, we have decided to calculate the fitness differently.

In the second case, having a solution, we have depicted it on a white or black background, depending on the average lightness of the colors of the pixels on the borders of the pieces. If the pixels had on average a lighter color, we depicted the individual on a black background and vice versa. This gave us more contrast between the borders of the pieces and the background. Then, having this image of an individual, we could easily calculate the fitness value for any edge. If we take all its neighboring pixels from this image, then we obtain as a result its ”neighbor edge”, with which we can compare it by the same formula as for rectangular pieces. Note that in this case we do not need to check separately which pieces are adjacent to each other and how, so the calculation is greatly simplified. Because the dissimilarities between pixels and white or black color were large, the best solutions began to acquire a more rectangular shape. This greatly improved our results.

We also decided to use the method we had previously used for solving puzzles with narrow rectangular pieces, and combined this value of the dissimilarities between edges with the relative area occupied by the individual by using the following formula:

$$\frac{(X_{max} - X_{min}) \times (Y_{max} - Y_{min})}{total\_area} * 0.9 + \left(\sum_{i=1}^{n} \sum_{j=1}^{4} D(p_{i,j}, M[i,j]) \right) * 0.1$$

where X_{max} and X_{min} are the maximum and minimum x values in the coordinates of the cluster that corresponds to the individual, Y_{max} and Y_{min} are the maximum and minimum y values in the coordinates of the cluster, M is a matrix that represents an individual, n is the number of images and

$$total\_area = \sum_{i=1}^{n} piece_i\_area.$$

This approach has given us better results in some cases than fitness based only on the dissimilarities between edges, but not always. We will make a detailed comparison in the next chapter.

3.3.6 Second mutation

While testing our solution, we encountered another problem. When we connect edges that are adjacent only at one vertex and these edges have almost the same length, we can connect edges near the wrong vertex mistakenly. This happens because the neighboring edge pixels are very similar in both cases, and it is not always clear which connection is correct. When we looked through the solutions proposed by our algorithm, this was one of its frequent mistakes. To solve it, we propose to use a second mutation inside the crossover, which, when connecting two edges at a vertex, with some probability will connect them at the other vertex. This situation is shown in Figure 3.13. In our solution, we have chosen the probability of not using an edge connection equal to 0.05, and the probability of using another vertex when connecting the edges equal to 0.001. The final version of the crossover then looks as follows:

1. Given two parents M_1 and M_2, connect all edges (i, a) and (j, b) for which the condition $M_1[i, a] = (j, b) = M_2[i, a]$ is fulfilled. With probability 0.05 the connection will not be made, and with probability 0.001 it will be made, but another pair of vertices will be joined.

2. Try to connect all best buddies with the same length in random order, provided that the connection is present in at least one of the parents. With probability 0.05 the connection will not be made.

3. Try to connect all best buddies with different length in random order, provided that the connection is present in at least one of the parents. With probability 0.05 the connection will not be made, and with probability 0.001 it will be made, but another pair of vertices will be joined.

4. Try to connect the edges with their most compatible pairs in random order. With probability 0.05 the connection will not be made, and with probability 0.001 it will be made, but another pair of vertices will be joined.

5. Make random connections until a total of n - 1 connections is made.

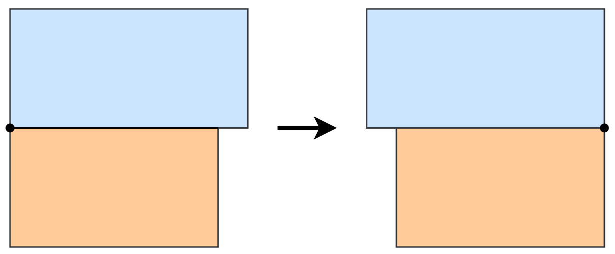


Figure 3.13: An example of the proposed second mutation. The bottom edge of the blue rectangle should be connected with the top edge of the red rectangle by touching the left vertex. After the mutation, the edges are connected by touching the right vertex.

3.3.7 Implementation details

In this section, we would also like to describe our implementation of a cluster with random rectangular pieces. Recall that in our previous solutions, the pieces were of the same size. Therefore, it was sufficient for us to represent them as objects of size 1 x 1 to check for potential overlap. In the case of rectangles of different sizes, we have to remember their real sizes and the coordinates of their vertices in the cluster. Note that all pieces, as well as any of their allowed connections, are polygons. Having 2 clusters represented as polygons, when connecting them we only need to merge the corresponding polygons. If we want to check whether two clusters intersect or not, we perform an operation similar to what we did for square pieces. First, we determine the coordinates of the connection point in the larger cluster. Then we rotate the smaller cluster, if necessary, and move it so that its coordinates at the connection point are correct. Next we need to calculate the intersection area of the two polygons and if it is greater than 0, this means that some of the pieces intersect and the connection cannot be made. We have used one of the available Python libraries for working with geometric objects[1] to perform the operation of merging and to calculate the area of intersection of polygons.

[1]https://shapely.readthedocs.io/en/stable/index.html

4. Experiments

In this chapter, we describe the results of our experiments on several datasets. First, we have tested our solutions on a dataset with color photographs, which has also been used in other works. Then we converted them to black and white images and tested our solutions on them. The third group of experiments we have performed on our dataset with printed texts.

We would like to emphasize that the types of puzzles for which we have proposed solutions have not yet been solved by anyone. Therefore, in this chapter we compare our results with the results of the solutions for the square pieces since this type of puzzles is the most similar to all our puzzles. In all cases where we had the opportunity, we divided the images into approximately the same number of pieces as the other authors, so that the comparison is more representative.

As a measure of the accuracy of our solutions, we, like the other authors, have used a measure proposed by Cho et al. in their work (Cho et al., 2010). They call it the *neighbor comparison metric*, in which the correctness of the solution is evaluated as the percentage of correctly guessed connections between edges. In our case, the solution is represented as a matrix of size $n \times 4$. We compare the elements of this matrix and the matrix of the correct solution, and the score of the success of this solution will be the number of identical elements in the matrices divided by the number of all elements.

4.1 Pomeranz et al. dataset with photos

4.1.1 Color images

In this section, we describe experiments performed on the dataset from the work of Pomeranz et al. with color photographs of 3 sizes (Pomeranz et al., 2011). The first group consists of 20 images of size 644×980. The second group contains 3 images with the size of 1120×1652. In the third group there are also 3 images, but with a size of 1400×1848. All images are mostly nature photos, where there are few objects and large areas are occupied by a single color. This makes such images hard to reconstruct, because the best neighbors for the edges can be guessed inaccurately. In general, we as well as other authors have noticed that the content of the image greatly affects the success of the solution.

# pieces	Pomeranz et al.	Zhao et al.	Sholomon et al.
805	89.70%	96.07%	**96.26%**
2360	84.67%	**96.52%**	88.86%
3300	85.00%	**93.39%**	92.76%

Table 4.1: Solvers for puzzles with square pieces.

The authors of the proposed solutions for square pieces have tested them on pieces of size 28×28. To adequately compare our solutions with their solutions, we tried to divide the images into pieces so that their area was approximately the same. To begin with, we divided all the images from these three groups into rectangular and triangular pieces. As a result, the rectangular pieces had

# pieces	average of 3 runs	best of 3 runs
784	96.75%	96.99%
3135	87.19%	87.72%
4703	88.42%	88.88%

Table 4.2: Our solver for puzzles with rectangular pieces.

# pieces	same, average	same, best	random, average	random, best
920	95.32%	95.46%	95.26%	95.44%
2240	96.62%	96.72%	96.69%	96.86%
3360	96.89%	96.93%	97.29%	97.38%

Table 4.3: Our solver for puzzles with triangular pieces.

dimensions of 23×35 in the first group, 10×59 in the second group, and 25×22 in the third group. The straight edges of the triangular pieces had dimensions of 28×49 in the first group, 28×59 in the second group, and 35×44 in the third group. We have also considered 2 cases of right triangle pieces. In the first one, all the triangles are identical, which means that the rectangles from which they are formed have been equally separated by a diagonal line. In the second case, the rectangles have been divided at random by one of the two possible diagonal lines. Next, we have compared our solutions with the best solutions proposed for the square pieces, which are shown in Table 4.1 and taken from the works of the authors (Pomeranz et al., 2011, Zhao et al., 2020, Dror Sholomon and Netanyahu, 2014). We do not specify the results of our solution for square pieces separately, since it is only an exact realization of the work of Sholomon et al., and we assume that ours and their results do not differ. Our results for rectangular and triangular pieces are shown in Tables 4.2 and 4.3. Note that the solutions proposed by Pomeranz et al. and Zhao et al. use knowledge of image dimensions in their solutions. Our solutions, on the other hand, are more universal. In general, as expected, our proposed solutions are not worse than the others, since the reconstruction of images from rectangular and triangular pieces is easier than from square pieces. However, we also do not see a significant improvement in the results, since some of the images with any shape of pieces remain hard to reconstruct. We ran our algorithms 3 times on each image. Tables 4.2 and 4.3 show the average and the best results among all images.

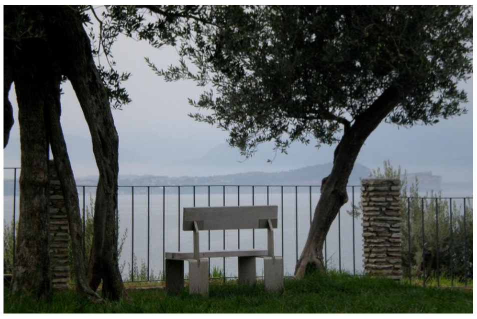
Let us consider the results for each picture from the first group separately in Tables 4.4 and 4.5. Here we see that although the average results for rectangular pieces (96.99%) are slightly better than for triangular pieces (95.46% and 95.44%), the number of perfectly solved images is higher in the case of triangular pieces (7 and 9 versus 4). However, we see that the solution for the triangular pieces is less stable and shows, for example, much worse results for pictures 6 and 12. These pictures are shown in Figures 4.1a and 4.1b, and the solutions proposed by our algorithm for the triangular pieces in Figures 4.2a and 4.2b. We see that on both images there is a dark-colored green tree, which creates a big problem for our solution. We also notice that the solution in Figure 4.2a has mostly correctly connected diagonal edges and has problems with straight edges. However, the solution in Figure 4.2b has incorrectly connected edges of both types equally and

image	NCM, average	NCM, best
1	97.83%	97.85%
2	92.24%	92.55%
3	98.36%	99.87%
4	99.84%	**100.0%**
5	97.60%	97.65%
6	99.59%	99.59%
7	98.64%	98.82%
8	95.43%	95.43%
9	99.48%	99.56%
10	98.52%	98.54%
11	93.49%	93.91%
12	78.06%	78.62%
13	99.33%	99.33%
14	99.61%	**100.0%**
15	**100.0%**	**100.0%**
16	99.23%	99.23%
17	98.82%	99.59%
18	89.61%	89.79%
19	**100.0%**	**100.0%**
20	99.36%	99.46%

Table 4.4: Detailed results for the first group of images, rectangular pieces.

Same triangles			Random triangles		
image	NCM, average	NCM, best	image	NCM, average	NCM, best
1	94.51%	94.70%	1	96.08%	96.16%
2	89.07%	89.89%	2	92.12%	93.25%
3	99.78%	99.78%	3	99.75%	99.75%
4	99.83%	**100.0%**	4	**100.0%**	**100.0%**
5	99.81%	99.81%	5	98.69%	99.29%
6	74.51%	74.56%	6	69.56%	69.79%
7	**100.0%**	**100.0%**	7	**100.0%**	**100.0%**
8	**100.0%**	**100.0%**	8	99.89%	**100.0%**
9	98.66%	98.75%	9	99.37%	99.37%
10	99.78%	99.78%	10	99.78%	99.78%
11	97.43%	98.07%	11	98.01%	98.65%
12	63.23%	63.58%	12	61.79%	62.17%
13	**100.0%**	**100.0%**	13	**100.0%**	**100.0%**
14	**100.0%**	**100.0%**	14	**100.0%**	**100.0%**
15	**100.0%**	**100.0%**	15	**100.0%**	**100.0%**
16	98.99%	99.13%	16	99.02%	99.10%
17	99.64%	99.64%	17	**100.0%**	**100.0%**
18	91.90%	91.98%	18	91.52%	91.54%
19	**100.0%**	**100.0%**	19	**100.0%**	**100.0%**
20	99.45%	99.59%	20	99.78%	**100.0%**

Table 4.5: Detailed results for the first group of images, same and random triangular pieces.


(a) The sixth image from the first group.

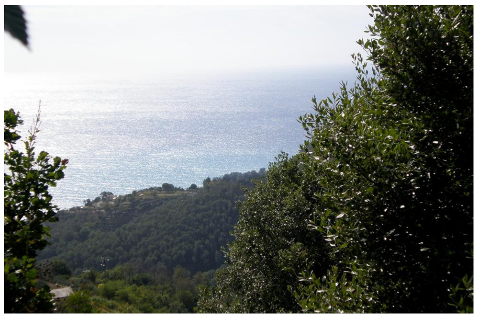
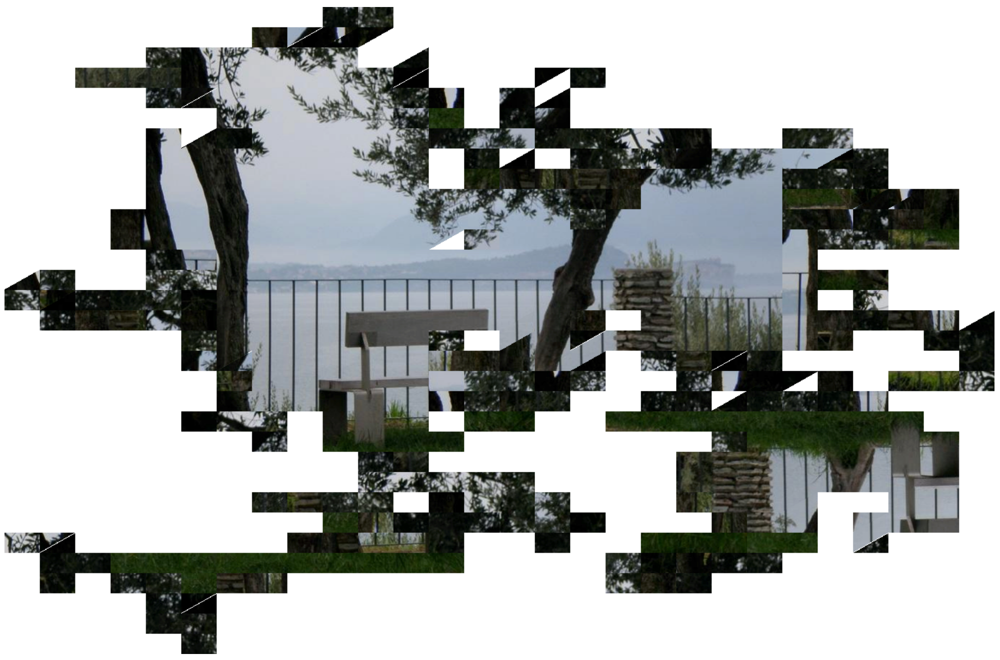

(b) The twelfth image from the first group.

Figure 4.1: Images from the first group which are difficult to reconstruct.


(a) A solution proposed to the sixth image from the first group. NCM = 74.56%

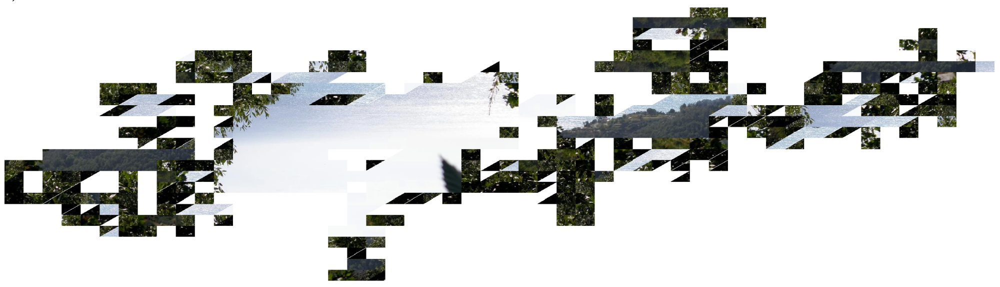

(b) A solution proposed to the twelfth image from the first group. NCM = 63.58%

Figure 4.2: The solutions proposed to the images 6 and 12 from the first group. Pieces have a shape of triangle.

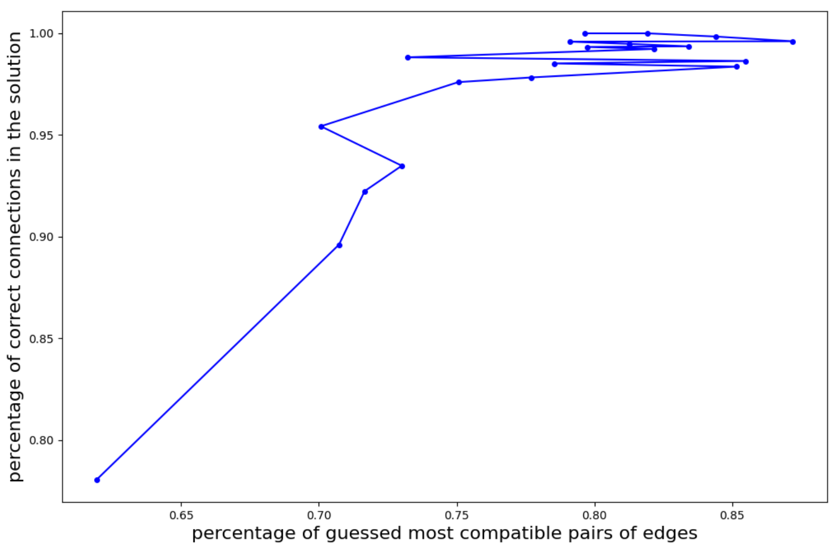


Figure 4.3: The dependence of the success rate of solution for rectangular pieces on the percentage of initially correct guesses of the most similar edges.

has the worst success rate of all of them.

Recall that before we start all our solutions for each edge, we find its most similar neighbor. The assumptions thus obtained are used within the crossover and affect how successful the resulting solution will be. The large difference in results for triangular pieces motivated us to analyze how well the best neighbors are guessed for each image and how the number of correctly guessed neighbors affects the success of the whole solution in the output of the algorithm. We calculated the average percentage of correctly guessed neighbors for all edges except the edges on the borders of the image. The results for the rectangular and triangular pieces are shown in Table 4.6, and in Figures 4.3 and 4.4. Here we see that for solutions of puzzles made of rectangular pieces with success rate of 95% or higher, it is necessary to correctly guess at least 70% of the correct neighbors. In the case of puzzles consisting of triangular pieces, to solve them with the same success rate it is necessary to guess about 85% of correct neighbors correctly, which makes this type of puzzles more dependent on the similarity measure of edges. Table 4.6 also shows that in the case of triangular pieces, it is worth relying more on the guessed neighbors for diagonal edges, since in all cases more correct neighbors have been guessed for them than for the straight edges.

The next type of pieces we have considered are narrow rectangles, where the sides are very different in length. This case requires a different approach, and the same algorithm does not work for it as for the previous type of rectangular pieces. This is where the second type of crossover described in the previous chapter, in which the longer edges have priority when connected, comes in handy. Next, we have used the second version of the fitness function, augmented by the relative area occupied by the individual, also described in the previous chapter. We divided the images from the first group into pieces of size 644×10 and 322×10, images from the second group into pieces of size 10×1652 and 10×826, and images from the third group into pieces of size 1400×12 and 700×12. The results

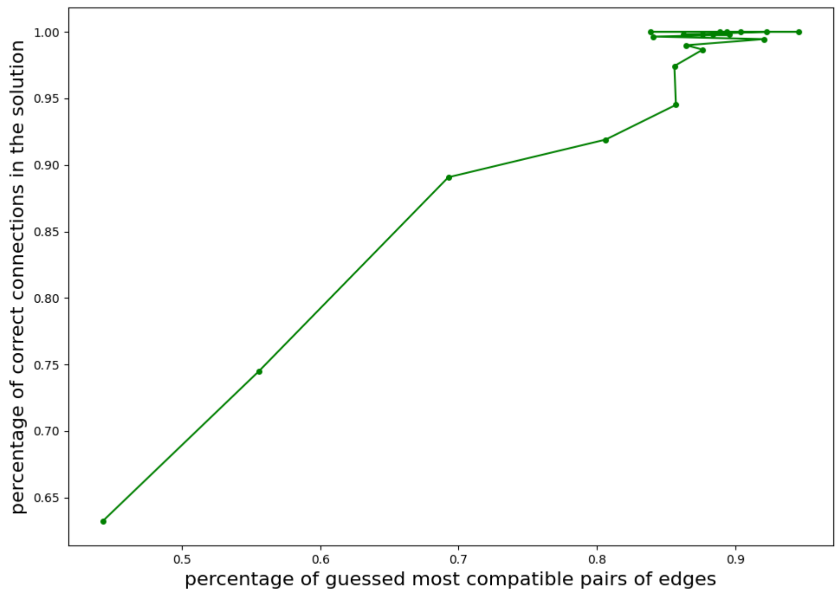


Figure 4.4: The dependence of the success rate of solution for triangular pieces on the percentage of initially correct guesses of the most similar edges.

Percentage of guessed most compatible neighbours				
	rectangle	triangle		
image	straight	diagonal	straight	both
1	77.71%	96.85%	79.87%	85.71%
2	71.66%	89.13%	58.84%	69.26%
3	85.15%	98.91%	84.66%	89.56%
4	84.42%	99.78%	81.24%	87.62%
5	75.06%	97.06%	80.62%	86.28%
6	79.13%	88.54%	38.19%	55.58%
7	85.48%	99.13%	88.71%	92.29%
8	70.07%	97.93%	76.51%	83.88%
9	81.28%	95.86%	83.35%	87.65%
10	78.54%	97.82%	83.41%	88.36%
11	73.01%	93.69%	81.36%	85.60%
12	61.93%	70.76%	29.99%	44.28%
13	79.76%	98.26%	84.72%	89.37%
14	87.20%	99.67%	91.91%	94.57%
15	81.94%	99.89%	85.40%	90.38%
16	82.17%	95.65%	81.64%	86.46%
17	73.21%	95.86%	77.88%	84.07%
18	70.73%	87.71%	76.91%	80.63%
19	79.66%	98.36%	83.92%	88.89%
20	83.43%	98.58%	88.65%	92.07%
average	78.08%	94.97%	76.89%	83.13%

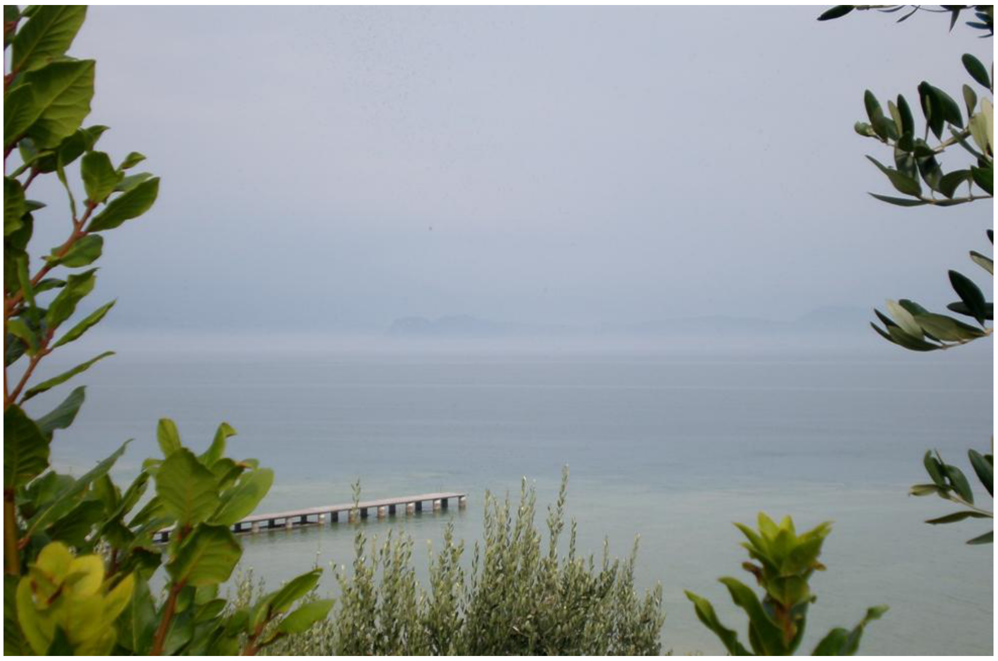
Table 4.6: Percentage of guessed most compatible neighbors for straight and diagonal edges.

image	644×10	322×10
1	**100.0%**	**100.0%**
2	99.39%	**100.0%**
3	99.39%	**100.0%**
4	82.04%	92.65%
5	**100.0%**	**100.0%**
6	99.39%	97.96%
7	**100.0%**	**100.0%**
8	**100.0%**	**100.0%**
9	**100.0%**	**100.0%**
10	**100.0%**	**100.0%**
11	**100.0%**	**100.0%**
12	**100.0%**	**100.0%**
13	**100.0%**	**100.0%**
14	**100.0%**	**100.0%**
15	**100.0%**	**100.0%**
16	**100.0%**	**100.0%**
17	99.39%	**100.0%**
18	**100.0%**	**100.0%**
19	**100.0%**	**100.0%**
20	82.45%	**100.0%**
average	98.10%	99.53%

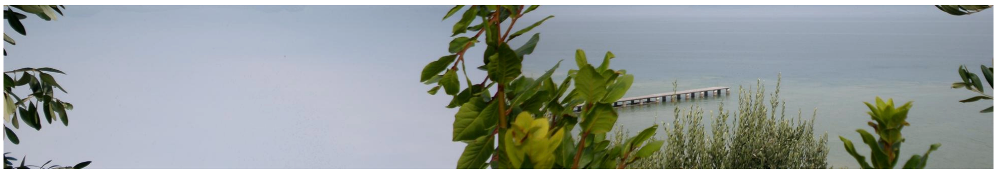
Table 4.7: Results for images from the first group divided into narrow rectangles.

image	10×1652	10×826	1400×12	700×12
1	**100.0%**	**100.0%**	**100.0%**	**100.0%**
2	81.25%	98.21%	83.1%	94.55%
3	**100.0%**	**100.0%**	**100.0%**	**100.0%**
average	93.8%	99.40%	94.4%	98.18%

Table 4.8: Results for images from the second and third groups divided into narrow rectangles.



(a) The third image from the first group.



(b) A solution proposed to the third image from the first group. The pieces have a shape of rectangle and size 644×10.

Figure 4.5: The solution proposed by our algorithm for the third image of the first group. Despite the fact that it is wrong, the error made by the algorithm is not immediately obvious even to the human eye.

of our experiments are shown in Tables 4.7 and 4.8. In this case, we ran our algorithm only 1 time for all images, because in most cases they were completely reconstructed almost immediately. We see that only a few images could not be reconstructed successfully. However, if we look at the solutions proposed by our algorithm, we see that even for the human eye the errors committed in the solution are not so obvious. One of these solutions is shown in Figure 4.5b. We will compare it with the correct solution shown in Figure 4.5a, and we will see that it is practically the same, but two halves of the image are connected horizontally, as a consequence of the connection of short edges, whereas long edges should have been connected. However, we still consider the proposed solution to be successful.

We have performed the next part of experiments on images from all 3 groups, divided recursively into random rectangles. Since this problem is more complex, and because the speed of our solution is not fast, we have decided to focus our attention on a smaller number of pieces. In the previous chapter, we have proposed 2 versions of the fitness function for our algorithm, where the first is based on the difference between all the pixels on the edges, and the second is augmented by the area occupied by the individual. We have also proposed a second kind of mutation, where the vertex near which the edges are connected changes with a

image	fit 1, mut 0	fit 1, mut 0.001	fit 2, mut 0	fit 2, mut 0.001
1	100.0%	100.0%	100.0%	100.0%
2	96.52%	95.13%	96.35%	**96.53%**
3	97.91%	96.87%	97.22%	**97.92%**
4	97.91%	**100.0%**	97.56%	98.26%
5	100.0%	100.0%	100.0%	100.0%
6	100.0%	100.0%	100.0%	100.0%
7	100.0%	100.0%	100.0%	100.0%
8	100.0%	100.0%	100.0%	100.0%
9	100.0%	100.0%	100.0%	100.0%
10	100.0%	100.0%	100.0%	100.0%
11	100.0%	100.0%	100.0%	100.0%
12	**88.19%**	85.93%	**88.19%**	86.97%
13	100.0%	100.0%	100.0%	100.0%
14	100.0%	100.0%	100.0%	97.91%
15	100.0%	100.0%	100.0%	100.0%
16	100.0%	100.0%	100.0%	100.0%
17	100.0%	100.0%	100.0%	100.0%
18	97.22%	97.22%	**100.0%**	97.22%
19	100.0%	100.0%	100.0%	100.0%
20	100.0%	100.0%	100.0%	100.0%
21	100.0%	100.0%	100.0%	100.0%
22	100.0%	100.0%	100.0%	100.0%
23	100.0%	100.0%	100.0%	100.0%
24	100.0%	100.0%	100.0%	100.0%
25	100.0%	100.0%	100.0%	100.0%
26	100.0%	100.0%	100.0%	100.0%
	99.14%	99.04%	**99.20%**	99.03%

Table 4.9: Results of the reconstruction of images divided into 64 random rectangles. The effectiveness of the solutions with 2 different fitness functions, as well as with and without the second mutation, is shown.

image	fit 1, mut 0	fit 1, mut 0.001	fit 2, mut 0	fit 2, mut 0.001
1	**96.78%**	96.09%	91.14%	96.09%
2	**94.27%**	94.18%	**94.27%**	94.09%
3	98.00%	98.00%	98.00%	98.00%
4	100.0%	100.0%	100.0%	100.0%
5	**98.87%**	99.04%	98.52%	98.52%
6	97.74%	97.74%	97.74%	96.44%
7	99.30%	99.30%	99.30%	99.30%
8	96.70%	**100.0%**	96.35%	**100.0%**
9	96.00%	96.00%	96.18%	**96.52%**
10	99.04%	99.04%	99.04%	**100.0%**
11	96.35%	96.18%	96.18%	**96.44%**
12	**93.22%**	92.96%	92.96%	93.05%
13	98.87%	98.87%	98.87%	98.87%
14	100.0%	100.0%	100.0%	100.0%
15	98.26%	98.26%	**98.61%**	96.96%
16	97.13%	95.66%	97.13%	97.13%
17	93.92%	93.57%	93.92%	**94.18%**
18	91.31%	90.89%	**91.32%**	90.71%
19	93.14%	**95.05%**	93.14%	92.96%
20	100.0%	100.0%	100.0%	100.0%
21	100.0%	100.0%	100.0%	100.0%
22	100.0%	100.0%	100.0%	100.0%
23	100.0%	100.0%	100.0%	100.0%
24	97.65%	**99.05%**	**99.05%**	99.04%
25	**98.43%**	98.09%	97.74%	98.09%
26	98.35%	**98.44%**	98.35%	98.43%
	97.44%	**97.55%**	97.22%	97.49%

Table 4.10: Results of the reconstruction of images divided into 128 random rectangles. The effectiveness of the solutions with 2 different fitness functions, as well as with and without the second mutation, is shown.

image	fit 1, mut 0	fit 1, mut 0.001	fit 2, mut 0	fit 2, mut 0.001
1	85.15%	85.37%	85.18%	**85.80%**
2	81.59%	81.33%	**81.68%**	81.20%
3	90.21%	90.19%	90.21%	**90.40%**
4	**89.64%**	89.23%	89.60%	89.53%
5	81.99%	81.70%	82.03%	**82.50%**
6	82.73%	**82.92%**	82.57%	82.55%
7	89.91%	89.40%	**89.97%**	89.36%
8	**82.34%**	82.11%	**82.34%**	82.14%
9	**85.72%**	84.63%	84.96%	84.85%
10	**81.81%**	81.12%	81.44%	80.98%
11	78.39%	78.79%	78.49%	**78.97%**
12	75.22%	75.46%	**75.82%**	75.34%
13	**85.81%**	84.85%	85.46%	84.83%
14	**90.06%**	89.30%	89.90%	89.34%
15	86.91%	86.80%	**87.15%**	86.65%
16	88.38%	88.76%	**89.21%**	88.38%
17	87.41%	87.35%	87.28%	**87.73%**
18	**81.79%**	80.85%	81.68%	81.57%
19	**85.62%**	85.22%	85.48%	85.43%
20	91.19%	91.03%	**91.51%**	90.69%
21	**97.80%**	97.46%	97.72%	97.46%
22	**92.14%**	91.75%	91.94%	91.21%
23	84.38%	84.29%	**85.54%**	84.41%
24	96.28%	**96.37%**	96.33%	96.07%
25	90.86%	90.73%	**90.88%**	90.21%
26	94.90%	94.81%	**95.03%**	94.50%
	86.86%	86.61%	**86.90%**	86.62%

Table 4.11: Results of the reconstruction of images divided into 512 random rectangles. The effectiveness of the solutions with 2 different fitness functions, as well as with and without the second mutation, is shown.

small probability. However, which of all the proposed variants will be effective is not obvious. Also recall that we assume the possible absence of some pieces, which further complicates our task. The two fitness function options, as well as the possibility of applying and not applying the second mutation, give us 4 different solutions. We have decided to compare and choose the best one out of them. To do this, we have divided the images from all 3 groups into 64, 128 and 512 pieces and tested all 4 variants of the solution on them. Each of the solutions has been run 3 times on each of the images. The results for the 64-, 128-, and 512-piece puzzles are shown in Tables 4.9, 4.10, and 4.11, respectively. The best result for each image is in bold. From the results, however, we do not see a clear winner, since there is little difference in the results. In some cases, for example, the new mutation has not been successful, in others, on the contrary, it has helped to find a better solution. The same situation is with the second fitness function. However, we still have chosen our main solution for the following experiments on the basis of the number of perfectly reconstructed images, as well as the average success. It is the variant with the second fitness function and without the second mutation.

Note that we have obtained the worst results by reconstructing images 12 and 18 from the first group. The solutions for them are depicted in the Figures 4.7 and 4.6. Note that image 12 created problems for all our proposed solutions, and if we look closely at the solution proposed for it, we see that the problem is created, as in the previous cases, by the pieces with the green leaves. In the case of image 18, we see that the proposed solution is quite good, and that its two halves are reassembled almost perfectly. However, they are not connected together, probably because of the incorrectly guessed best neighbors for some of the edges.

Our next step has been to check the effectiveness of our chosen solution in the case where some of the pieces are missing. For this purpose we have conducted experiments on all the same images with the same number of pieces, but at the input of the algorithm we have first removed 10% of random pieces, and then repeated the experiments, but with 20% of missing pieces. In general, in this case, we expect some decrease in the efficiency of the solution. This is at least because the correct solution can be several clusters of pieces that are not connected to each other, while our solution always connects all pieces into one cluster. The results of the experiments are shown in Tables 4.12 and 4.13. Here we do see some decrease in the success rate of the solutions, but only by a little bit. Moreover, in some cases, the success of the solution even increased, probably due to the absence of problem pieces that are hard to find a pair.

Analyzing the results for all three numbers of pieces, we see that despite the random division of images into pieces, what is depicted in the picture strongly influences the success of the solution in all cases. This can be seen in the example of picture 12 from the first group, which has some of the worst results in all cases, regardless of the number of pieces and the percentage of missing pieces.

Summarizing all our results, we believe that they are quite successful and comparable to existing solutions for square pieces. However, it is worth noting that the speed of our solutions is not as good as the solution proposed by Sholomon et al. in their paper (Dror Sholomon and Netanyahu, 2014). There they note that their solution for puzzles with 22,755 square pieces takes 3.5 hours, whereas

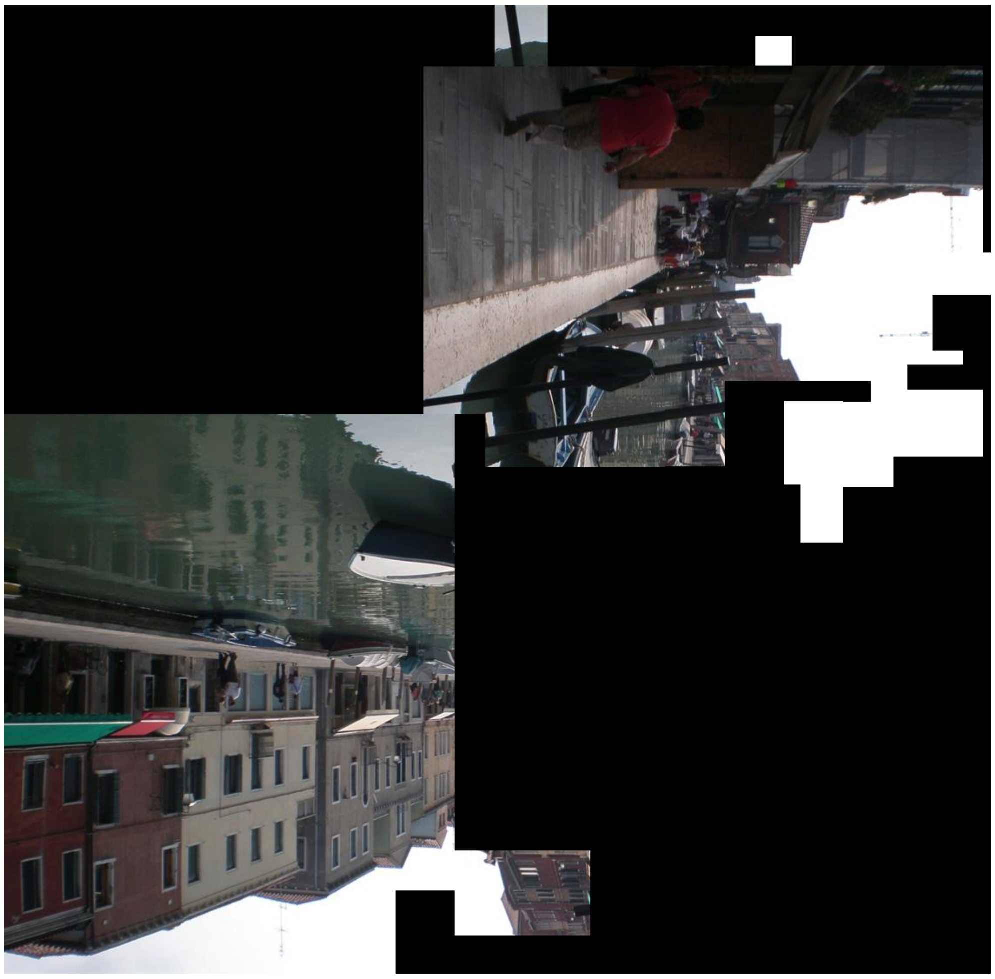


Figure 4.6: The solution proposed for the 18th image divided into 128 pieces. NCM = 91.32%.

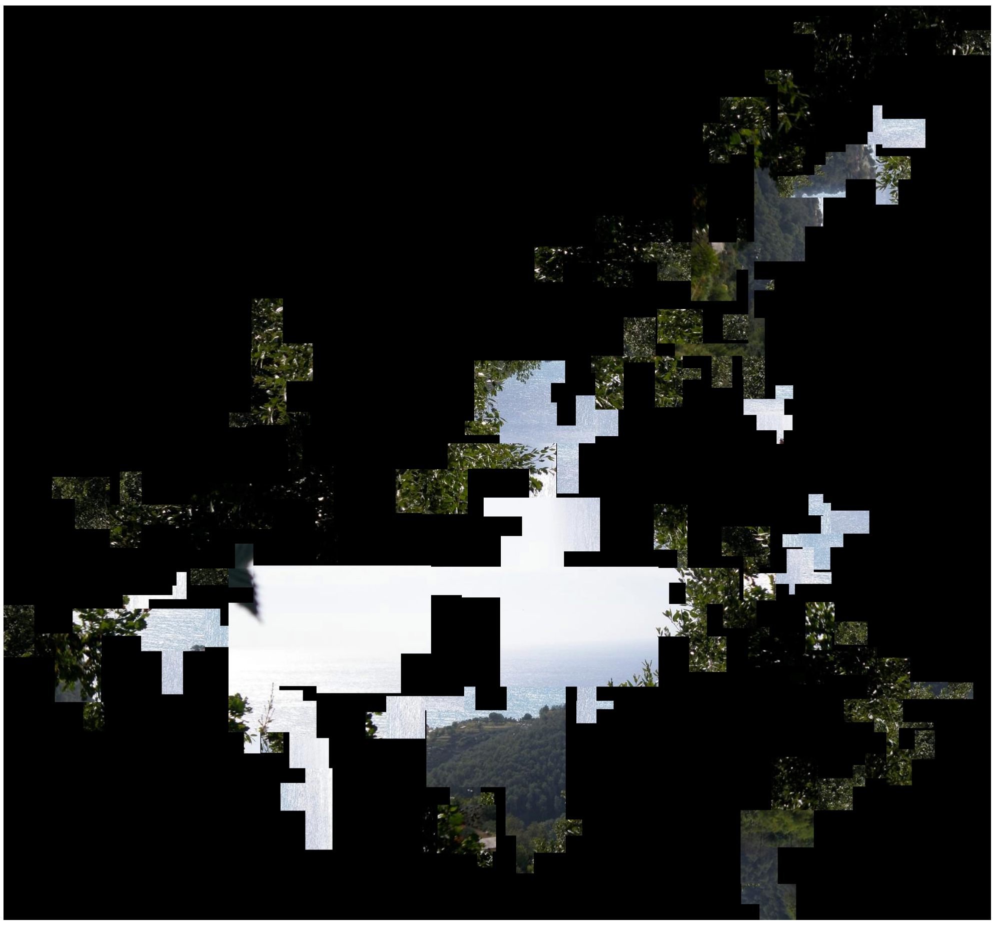


Figure 4.7: The solution proposed for the 12th image divided into 512 pieces. NCM = 75.82%.

64 pieces			
image	0%	10%	20%
1	100.0%	100.0%	96.07%
2	96.35%	96.55%	91.93%
3	97.22%	94.63%	93.02%
4	97.56%	96.55%	94.99%
5	100.0%	97.70%	96.51%
6	100.0%	94.63%	95.21%
7	100.0%	99.23%	96.94%
8	100.0%	97.13%	89.54%
9	100.0%	98.27%	95.64%
10	100.0%	100.0%	99.95%
11	100.0%	100.0%	98.26%
12	88.19%	83.90%	92.16%
13	100.0%	97.31%	94.12%
14	100.0%	99.23%	94.34%
15	100.0%	100.0%	98.26%
16	100.0%	99.62%	96.08%
17	100.0%	100.0%	95.42%
18	100.0%	95.01%	94.98%
19	100.0%	98.85%	94.99%
20	100.0%	97.70%	96.95%
21	100.0%	100.0%	96.08%
22	100.0%	98.46%	97.82%
23	100.0%	100.0%	94.77%
24	100.0%	100.0%	95.86%
25	100.0%	99.61%	94.34%
26	100.0%	98.65%	96.95%
	99.20%	97.81%	95.43%

128 pieces			
image	0%	10%	20%
1	91.14%	93.62%	92.37%
2	94.27%	92.36%	92.91%
3	98.00%	96.81%	94.23%
4	100.0%	99.61%	95.96%
5	98.52%	94.49%	92.70%
6	97.74%	95.75%	94.00%
7	99.30%	96.42%	96.08%
8	96.35%	96.71%	94.34%
9	96.18%	93.33%	89.76%
10	99.04%	97.48%	93.57%
11	96.18%	94.00%	91.39%
12	92.96%	90.82%	84.96%
13	98.87%	94.88%	92.81%
14	100.0%	98.45%	95.42%
15	98.61%	94.49%	95.42%
16	97.13%	94.97%	89.32%
17	93.92%	94.87%	90.85%
18	91.32%	88.50%	84.31%
19	93.14%	92.27%	90.85%
20	100.0%	98.26%	96.29%
21	100.0%	99.80%	95.53%
22	100.0%	97.10%	94.66%
23	100.0%	99.03%	91.07%
24	99.05%	98.65%	95.31%
25	97.74%	96.32%	92.27%
26	98.35%	98.26%	93.68%
	97.22%	95.66%	92.69%

Table 4.12: Results of the reconstruction of images divided into 64 and 128 random rectangles. The first column shows the results when all pieces are present. The second and third columns show the results in the situation when 10 and 20 % of the pieces are missing, respectively.

image	0%	10%	20%
1	85.18%	85.10%	82.57%
2	81.68%	80.74%	79.83%
3	90.21%	89.66%	86.55%
4	89.60%	89.75%	86.85%
5	82.03%	80.24%	82.62%
6	82.57%	79.85%	80.24%
7	89.97%	88.65%	86.42%
8	82.34%	81.93%	81.35%
9	84.96%	83.87%	83.22%
10	81.44%	81.99%	81.62%
11	78.49%	77.94%	77.83%
12	75.82%	76.19%	74.36%
13	85.46%	85.15%	82.73%
14	89.90%	89.39%	87.34%
15	87.15%	86.81%	83.41%
16	89.21%	88.62%	86.42%
17	87.28%	84.72%	82.57%
18	81.68%	81.75%	79.78%
19	85.48%	85.66%	82.33%
20	91.51%	88.64%	88.86%
21	97.72%	96.38%	94.42%
22	91.94%	90.02%	88.11%
23	85.54%	83.13%	81.97%
24	96.33%	95.28%	93.44%
25	90.88%	88.79%	87.61%
26	95.03%	93.18%	92.33%
	86.90%	85.90%	84.41%

Table 4.13: Results of the reconstruction of images divided into 512 random rectangles. The first column shows the results when all pieces are present. The second and third columns show the results in the situation when 10 and 20 % of the pieces are missing, respectively.

our solutions reconstruct images with 8 times fewer pieces on average in the same time. We believe that our implementation is not the most efficient, and that the solution time is also affected by the chosen programming language (Python). However, we have noticed that the number of epochs chosen by the authors of the algorithm equal to 100 is too large in most cases, and can be reduced. This makes the solution faster. Moreover, we noticed that with a large number of epochs and a number of pieces less than 1000, the algorithm finds the best solution before the 50th epoch, and then the success of the proposed solutions sometimes even worsens, even though the fitness value only improves. This shows that sometimes lower fitness values do not correspond to more successful solutions. Therefore, we propose to use 100 epochs in the algorithm only in the case of a large (more than 1000) number of pieces, because this is the case where this value is most successful. In experiments with fewer pieces, we have used 30 or 50 epochs, and this has given us good results more quickly.

4.1.2 Black and white images

The authors of most of the proposed solutions for puzzles with differently shaped pieces tested their solutions on color images. We found it interesting to study the effect on the results of converting images from color to black and white. This type of images should be more difficult to reconstruct, as there is no information about the color of the pieces, and we can only rely on their lightness. We experimented on the same dataset with three groups of images, but this time we made them black and white. We left the number of pieces the same for a better comparison. The results of the experiments on the puzzles with rectangular and triangular pieces are shown in Tables 4.14, 4.15 and 4.16. We see that the success of the experiments performed on the first group of images decreased by about 3%, which is not critical. However, with more pieces, we see more degradation of results. This can be seen in the results of experiments performed on the second and third groups of images with rectangular pieces, where we see a worsening of the results by 10% and 24% respectively. However, it is interesting that the results for triangular pieces worsened by only a couple of percents. This may be due to the lower number of neighborhood candidates for the edges. We also believe that the neighborhood candidates are more accurate for diagonal edges rather than for straight edges, so the solution for triangular pieces is not as vulnerable to missing colors in the image.

Next, we have tested our solution for narrow rectangular pieces. The results are shown in Tables 4.17 and 4.18. Interestingly, in this case the results did not worsen much, and in some cases are even better, as for example in the case of image 2 in the second group. This provides an interesting observation that in the case of some images it may be better to rely on pixel lightness rather than color when comparing edges.

The results of experiments performed on images divided into 512 random rectangles also show some worsening of the results, from 7 to 9%. They are shown in Table 4.19. It is interesting that on average the results do not change regardless of the number of missing pieces.

Note also that, as in the case of color images, the most difficult to reconstruct in most cases is image 12. We have looked more closely at the solutions of our

Rectangular pieces, b&w		
image	average	best
1	95.45%	96.63%
2	88.87%	89.74%
3	94.61%	95.43%
4	97.09%	98.51%
5	84.54%	85.61%
6	98.11%	99.59%
7	94.10%	95.10%
8	89.15%	89.43%
9	85.28%	85.76%
10	98.52%	98.52%
11	85.10%	85.25%
12	72.55%	73.03%
13	97.32%	97.34%
14	99.77%	99.77%
15	99.51%	99.51%
16	97.88%	98.06%
17	98.46%	98.67%
18	90.10%	90.20%
19	99.41%	99.59%
20	97.52%	97.52%
	93.17%	93.66%

Triangular pieces, b&w		
image	average	best
1	92.90%	93.34%
2	83.23%	83.77%
3	97.50%	97.50%
4	89.80%	90.02%
5	95.48%	95.73%
6	72.06%	72.33%
7	98.64%	98.64%
8	94.51%	94.89%
9	88.36%	88.36%
10	99.78%	99.78%
11	94.91%	95.27%
12	62.17%	62.33%
13	**100.0%**	**100.0%**
14	**100.0%**	**100.0%**
15	**100.0%**	**100.0%**
16	93.72%	93.80%
17	99.45%	99.48%
18	91.52%	91.63%
19	**100.0%**	**100.0%**
20	97.77%	97.79%
	92.59%	92.73%

Table 4.14: Results of experiments performed on black and white images from the first group.

rectangular pieces, b&w, second group			rectangular pieces, b&w, third group		
image	average	best	image	average	best
1	91.52%	91.91%	1	71.91%	72.20%
2	72.13%	72.16%	2	52.67%	52.67%
3	66.63%	67.17%	3	68.30%	68.83%
	76.76%	77.08%		64.29%	64.57%

Table 4.15: Results of experiments performed on black and white images from the second and third groups.

triangular pieces, b&w, second group			triangular pieces, b&w, third group		
image	average	best	image	average	best
1	99.78%	99.78%	1	99.92%	99.95%
2	88.27%	88.37%	2	78.65%	78.80%
3	90.94%	91.79%	3	95.38%	95.52%
	93.00%	93.31%		91.32%	91.42%

Table 4.16: Results of experiments performed on black and white images from the second and third groups.

Narrow rectangular pieces, b&w		
image	644 × 10	322 × 10
1	**100.0%**	**100.0%**
2	99.39%	**100.0%**
3	**100.0%**	83.16%
4	84.08%	92.65%
5	**100.0%**	**100.0%**
6	99.39%	93.16%
7	**100.0%**	**100.0%**
8	**100.0%**	**100.0%**
9	**100.0%**	**100.0%**
10	**100.0%**	**100.0%**
11	**100.0%**	**100.0%**
12	**100.0%**	**100.0%**
13	**100.0%**	**100.0%**
14	**100.0%**	**100.0%**
15	**100.0%**	**100.0%**
16	**100.0%**	**100.0%**
17	99.39%	**100.0%**
18	**100.0%**	**100.0%**
19	99.39%	**100.0%**
20	79.79%	**100.0%**
	98.07%	98.45%

Table 4.17: Results of experiments performed on black and white images from the first group. Pieces have a shape of narrow rectangle.

Narrow rectangular pieces, b&w				
image	10 × 1652	10 × 826	1400 × 12	700 × 12
1	**100.0%**	**100.0%**	**100.0%**	**100.0%**
2	91.79%	84.64%	82.08%	78.96%
3	**100.0%**	**100.0%**	**100.0%**	**100.0%**
	97.26%	94.88%	94.03%	92.99%

Table 4.18: Results of experiments performed on black and white images from the second and third groups. Pieces have a shape of narrow rectangle.

Random rectangular pieces, b&w			
image	0%	10%	20%
1	76.49%	75.70%	**77.37%**
2	77.13%	**78.21%**	77.85%
3	78.10%	78.45%	**78.59%**
4	**83.02%**	81.85%	82.49%
5	**70.83%**	70.40%	70.13%
6	**77.43%**	75.07%	76.58%
7	76.49%	75.68%	**77.31%**
8	76.76%	**78.55%**	77.51%
9	76.12%	**76.59%**	74.90%
10	77.23%	77.41%	**77.91%**
11	71.94%	73.65%	**74.28%**
12	70.18%	**72.23%**	70.32%
13	**79.42%**	77.58%	78.42%
14	81.57%	**82.57%**	81.62%
15	78.10%	**78.28%**	78.16%
16	80.03%	**80.45%**	79.75%
17	78.34%	**77.89%**	77.83%
18	**77.38%**	76.30%	76.99%
19	**83.96%**	83.41%	81.00%
20	**79.86%**	79.15%	79.13%
21	**95.87%**	93.42%	90.65%
22	**86.26%**	85.32%	84.93%
23	**81.27%**	80.93%	79.89%
24	90.75%	**91.03%**	88.78%
25	**81.72%**	80.88%	78.88%
26	**90.38%**	87.87%	85.88%
	79.87%	79.57%	79.12%

Table 4.19: Results of experiments performed on black and white images from the second and third groups. Pieces have a shape of random rectangle.

algorithms proposed for it, and we see that the greatest number of errors has been again committed in the places where the leaves of the trees are depicted.

In general, we see that our solutions can also be used to reconstruct black and white images, although not with such good results as in the case of color images. Our experiments show the importance of pixel lightness when comparing edge similarity.

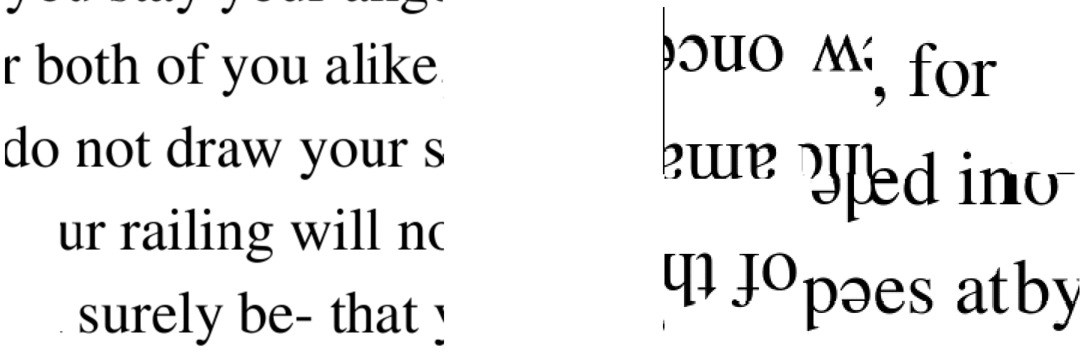
4.2 Our dataset with printed text

Our last idea has been to use our proposed solutions for the reconstruction of text documents. Usually all solutions proposed to solve this problem use letter recognition in one way or another. Furthermore, such solutions use information from the whole piece, not just the edges. We wondered how well text documents can be reconstructed based only on the pixels on the edges of the pieces.

For the experiments, we have created our own dataset of 20 images with printed text, where we have used 5 font sizes and 4 types of text alignment on the page. The text has been taken from the work The Iliad by Homer[1] and from the play Titus Andronicus by Shakespeare[2]. We have tried to divide these images into roughly the same number of pieces as in previous experiments to make the results clear. As a result, the rectangular pieces were 117×46, the triangular pieces were 117×92, and the narrow rectangles were 2340×12 and 1170×12. We have performed the experiment on each image 1 time, because according to our observations, the solutions that the algorithms produce did not practically change. The results are shown in Table 4.20. We see that texts divided into rectangles and triangles are reconstructed by a third on average, and if we look at the proposed solutions, we see that they strongly depend on where the images have been divided into pieces. For example, if the splitting occurred between letters, we have no information at the edges of the pieces. On the contrary, if the letters have been divided into parts, the chance of finding the correct neighbor for the edge increases. Figures 4.8a and 4.8b show parts of the text from rectangular pieces, which have been reconstructed well and those which have failed to be reconstructed. Figures 4.9a and 4.9b show the successful and unsuccessful parts, but for the triangular pieces. More successful has been the reconstruction of texts divided into narrow strips. This is how some shredding machines cut text documents, so this example of reconstruction is very realistic. Here we see a wide variation of success rate, from 21% to 75%. If we look at the most successful solution, we see that the text in the middle of the image is reconstructed almost perfectly, and mistakes have been made in the connection of the white borders of the image, which is not critical.

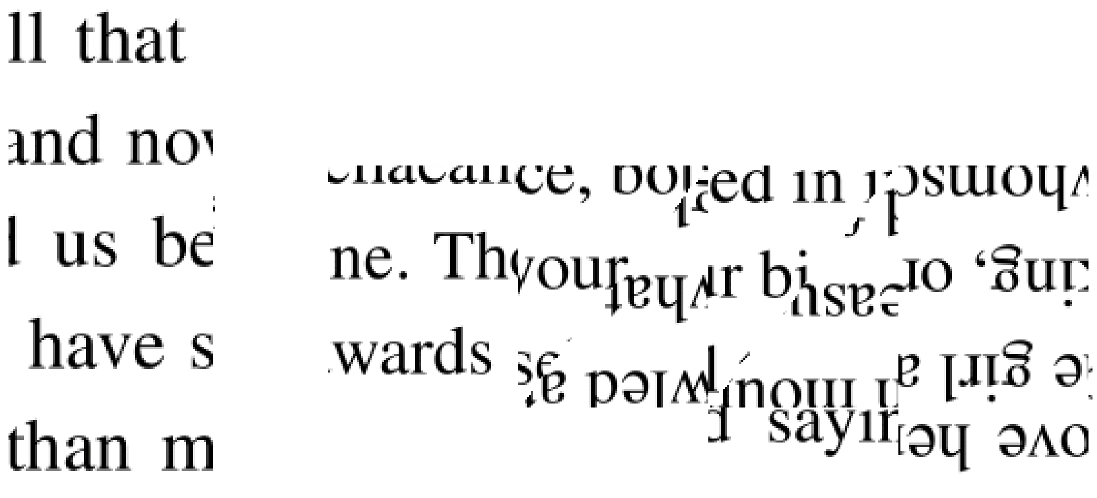
Next, we have divided the images with text into 512 random rectangular pieces and have tried to reconstruct them. We have also removed 10% of the input pieces and then 20% again, and have compared the success of the proposed solutions. The results of the experiments are shown in Table 4.21. They are comparable to the results obtained for rectangular pieces of the same size, but here we see an interesting feature. We see that the more pieces are missing, the more successful

[1]`http://classics.mit.edu/Homer/iliad.1.i.html`
[2]`http://shakespeare.mit.edu/titus/full.html`



(a) (b)

Figure 4.8: Examples of well (a) and poorly (b) reconstructed text from rectangular pieces.



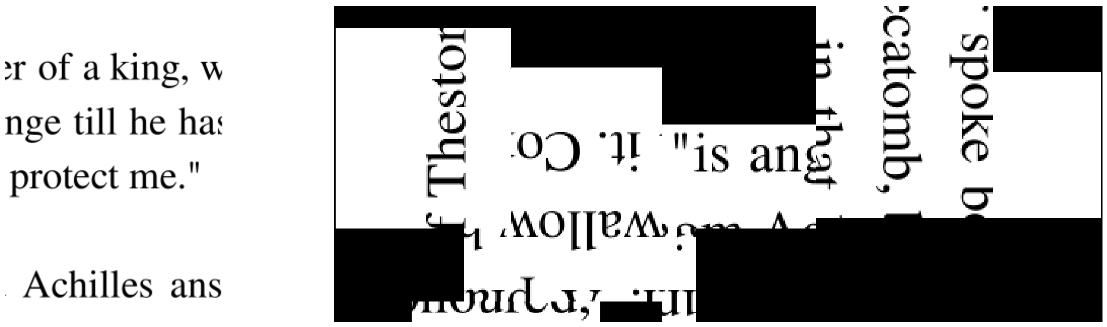
(a) (b)

Figure 4.9: Examples of well (a) and poorly (b) reconstructed text from triangular pieces.

Rectangular and triangular pieces, text		
image	rectangles	triangles
1	39.83%	33.96%
2	39.36%	34.93%
3	35.97%	33.05%
4	36.44%	33.72%
5	30.80%	32.15%
6	38.77%	33.44%
7	36.13%	33.40%
8	42.00%	33.85%
9	34.31%	33.58%
10	32.13%	32.67%
11	34.58%	33.12%
12	40.41%	34.69%
13	41.50%	36.08%
14	36.55%	35.79%
15	38.50%	37.99%
16	41.58%	35.48%
17	43.55%	34.55%
18	39.61%	35.93%
19	36.38%	36.25%
20	39.11%	37.01%
	37.88%	34.58%

Narrow rectangular pieces, text		
image	2340×12	1170×12
1	72.61%	67.32%
2	55.65%	52.75%
3	24.64%	22.03%
4	56.09%	22.68%
5	47.97%	22.39%
6	74.06%	68.77%
7	75.36%	63.98%
8	59.71%	63.26%
9	61.59%	22.17%
10	25.79%	21.59%
11	73.48%	54.28%
12	74.20%	58.12%
13	73.48%	53.12%
14	74.06%	55.00%
15	73.33%	21.96%
16	61.01%	69.13%
17	57.10%	55.87%
18	62.32%	52.10%
19	73.19%	56.59%
20	72.75%	21.88%
	62.42%	46.25%

Table 4.20: Results of experiments performed on images with text. Pieces have a shape of rectangles, narrow rectangles and triangles.



(a) (b)

Figure 4.10: Examples of well (a) and poorly (b) reconstructed text from random rectangular pieces.

solution, on average, our algorithm offers. Figure 4.10a shows an example of the part of the text that has been restructured better, and Figure 4.10b shows the one that has been restructured worse.

In general, this experiment has been interesting, and the results show that our solution can be used as an assistant to other algorithms for the reconstruction of text documents.

Random rectangular pieces, text			
image	0%	10%	20%
1	50.59%	53.26%	**56.15%**
2	48.76%	51.14%	**54.01%**
3	44.08%	47.12%	**49.16%**
4	46.66%	49.62%	**50.79%**
5	48.33%	51.05%	**54.74%**
6	49.20%	51.96%	**53.84%**
7	48.26%	50.63%	**54.15%**
8	48.63%	50.63%	**54.49%**
9	46.29%	46.59%	**52.44%**
10	47.46%	50.66%	**53.25%**
11	48.72%	51.24%	**52.66%**
12	42.62%	46.93%	**49.86%**
13	47.24%	50.52%	**52.87%**
14	51.43%	53.09%	**57.07%**
15	46.12%	47.38%	**50.54%**
16	43.68%	47.17%	**49.97%**
17	47.78%	50.22%	**53.20%**
18	47.50%	47.05%	**53.36%**
19	48.54%	50.61%	**54.39%**
20	49.86%	52.28%	**55.31%**
	47.59%	49.96%	**53.11%**

Table 4.21: Results of experiments performed on images with text. Pieces have a shape of random rectangles.

Conclusion

The goal of this work has been to research existing algorithms for reconstruction of images divided into pieces, as well as to propose our own solution for this problem. In the first chapter, we have classified the existing works on this topic into several types and have separately described the solutions proposed by other authors for each of them. We have also mentioned in the first chapter a similar problem, which however has a useful practical application, that is the reconstruction of textual documents. In our work, we have decided to propose solutions for the more interesting, in our view, shapes of pieces, these being rectangles, right-angle triangles, and rectangles of random size. We have also complicated the problem we have been solving by not relying on knowing the dimensions of the picture we are reconstructing.

The most similar problem to the one we have solved is the reconstruction of images divided into square pieces. One of the most successful and universal solutions for it that have been proposed to date is that of Sholomon et al. based on a genetic algorithm (Dror Sholomon and Netanyahu, 2014). We have described their algorithm as well as our implementation in more detail in the second chapter.

In the third chapter, we have separately described our solutions for all the proposed shapes of pieces. We have explained the peculiarities of each type of piece, as well as what changes in each case we need to make to the algorithm for square pieces. We have tested all our ideas on an available dataset with pictures, which has been also used by other authors in their works. For each type of piece, we have also shared what we consider to be interesting implementation details that may be useful to other researchers of this problem. We also provide a short documentation for developers and users in Attachment A.1.

In the last chapter, we have evaluated the performance of our solutions on the same dataset with photos of 3 different sizes. The largest number of pieces we have tried to reassemble was 4703 for rectangles, 3360 for triangles, and 512 for random-sized rectangles. However, we believe that our solutions can handle more pieces without a significant decrease in efficiency. In the fourth chapter, we show that the results of our solutions are comparable to the best results of other authors. It is worth mentioning, however, that our implementation is not as fast as other authors, and is a place of potential improvement in our solution.

Unlike other authors, we have also tested our solutions on black and white images as well as images with text. We have found that in many cases, we do not particularly need color information, but only lightness information to successfully reconstruct images. In the case of images with text, as expected, our solutions are not sufficient for their successful reconstruction. However, we believe that they can be used as an assistance for other, more advanced algorithms.

We would also like to describe ideas for further research on this topic. We have already mentioned that pixel lightness has been very useful in comparing edge dissimilarity of black and white pieces. It would be interesting to propose and test another edge dissimilarity function in which color and lightness would be considered separately, or where lightness would have more weight. In general, the reconstruction of black and white images is not much studied and is an interesting research topic in its own right.

We also want to note that our solution for random rectangles has not been as successful as our other solvers, because this problem is more complicated in general. We have not tested it on a large number of pieces, due to time constraints. Note that the biggest problem of our algorithm is posed by pieces with very short edges, for which it is very difficult to find correct neighbors. We have proposed various solutions to this problem in our work, but they are not perfect. Therefore, this part of the algorithm can also be the subject of further research.

Bibliography

H. Bunke and G. Kaufmann. Jigsaw puzzle solving using approximate string matching and best-first search. In Dmitry Chetverikov and Walter G. Kropatsch, editors, *Computer Analysis of Images and Patterns*, pages 299–308, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg. ISBN 978-3-540-47980-2.

Shengjiao Cao, Hairong Liu, and Shuicheng Yan. Automated assembly of shredded pieces from multiple photos. In *2010 IEEE International Conference on Multimedia and Expo*, pages 358–363, 2010. doi: 10.1109/ICME.2010.5582544.

Taeg Sang Cho, Shai Avidan, and William T. Freeman. A probabilistic image jigsaw puzzle solver. *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 183–190, 2010.

Min Gyo Chung, M.M. Fleck, and D.A. Forsyth. Jigsaw puzzle solver using shape and color. *ICSP '98. 1998 Fourth International Conference on Signal Processing (Cat. No.98TH8344)*, 2:877–880, 1998. doi: 10.1109/ICOSP.1998.770751.

Erik D. Demaine and Martin L. Demaine. Jigsaw puzzles, edge matching, and polyomino packing: Connections and complexity. *Graphs and Combinatorics*, 23:195–208, 2007.

Varad Deshpande, Devansh Gosalia, Tanya Gupta, and Neha Katre. Review paper on stitching of torn documents. *International Journal of Engineering Development and Research*, 6:361–366, 2018.

Eli (Omid) David Dror Sholomon and Nathan S. Netanyahu. Genetic algorithm-based solver for very large multiple jigsaw puzzles of unknown dimensions and piece orientation. *ACM Genetic and Evolutionary Computation Conference (GECCO)*, page 1191–1198, 2014.

Suohai Fan. A solution to reconstruct cross cut shredded text documents based on character recognition and genetic algorithm. *Abstract and Applied Analysis*, 2014:11, 06 2014. doi: 10.1155/2014/829602.

Andrew C. Gallagher. Jigsaw puzzles with pieces of unknown orientation. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 382–389, 2012. doi: 10.1109/CVPR.2012.6247699.

David Goldberg, Christopher Malon, and Marshall Bern. A global approach to automatic solution of jigsaw puzzles. *Computational Geometry*, 28(2):165–174, 2004. ISSN 0925-7721. doi: https://doi.org/10.1016/j.comgeo.2004.03.007. Special Issue on the 18th Annual Symposium on Computational Geometry - SoCG2002.

Wenjing Guo, Wenhong Wei, Yuhui Zhang, and Anbing Fu. A genetic algorithm-based solver for small-scale jigsaw puzzles. In Ying Tan, Yuhui Shi, and Milan Tuba, editors, *Advances in Swarm Intelligence*, pages 362–373, Cham, 2020. Springer International Publishing.

Shir Gur and Ohad Ben-Shahar. From square pieces to brick walls: The next challenge in solving jigsaw puzzles. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 4049–4057, 2017. doi: 10.1109/ICCV.2017.434.

Peleg Harel and Ohad Ben-Shahar. Crossing cuts polygonal puzzles: Models and solvers. *The Conference on Computer Vision and Pattern Recognition*, pages 3083–3092, 2021.

Weixin Kong and B.B. Kimia. On solving 2d and 3d puzzles using curve matching. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 2, pages II–II, 2001. doi: 10.1109/CVPR.2001.991015.

D.A. Kosiba, P.M. Devaux, S. Balasubramanian, T.L. Gandhi, and K. Kasturi. An automatic jigsaw puzzle solver. *Proceedings of 12th International Conference on Pattern Recognition*, 1:616–618 vol.1, 1994. doi: 10.1109/ICPR.1994.576377.

Canyu Le and Xin Li. Jigsawnet: Shredded image reassembly using convolutional neural network and loop-based composition. *CoRR*, abs/1809.04137, 2018. URL `http://arxiv.org/abs/1809.04137`.

Michail Makridis and Nikos Papamarkos. A new technique for solving puzzles. *IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society*, 40:789–97, 10 2009. doi: 10.1109/TSMCB.2009.2029868.

Ture R. Nielsen, Peter Drewsen, and Klaus Hansen. Solving jigsaw puzzles using image features. *Pattern Recognition Letters*, 29(14):1924–1939, 2008. ISSN 0167-8655. doi: 10.1016/j.patrec.2008.05.027.

Dolev Pomeranz, Michal Shemesh, and Ohad Ben-Shahar. A fully automated greedy square jigsaw puzzle solver. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 9 – 16, 07 2011. doi: 10.1109/CVPR.2011.5995331.

Kantilal Rane and S. Bhirud. Text reconstruction using torn document mosaicing. *International Journal of Computer Applications*, 30:21–27, 09 2011. doi: 10.5120/3669-5170.

Ankush Roy and Utpal Garain. A probabilistic model for reconstruction of torn forensic documents. In *2013 12th International Conference on Document Analysis and Recognition*, pages 494–498, 2013. doi: 10.1109/ICDAR.2013.105.

Christian Schauer, Matthias Prandtstetter, and Günther R. Raidl. A memetic algorithm for reconstructing cross-cut shredded text documents. In María J. Blesa, Christian Blum, Günther Raidl, Andrea Roli, and Michael Sampels, editors, *Hybrid Metaheuristics*, pages 103–117, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-16054-7.

Dror Sholomon, Eli David, and Nathan Netanyahu. A generalized genetic algorithm-based solver for very large jigsaw puzzles of complex types. *Proceedings of the National Conference on Artificial Intelligence*, 4:2839–2845, 01 2014.

Dror Sholomon, Eli David, and Nathan S. Netanyahu. An automatic solver for very large jigsaw puzzles using genetic algorithms. *CoRR*, abs/1711.06767, 2017. URL `http://arxiv.org/abs/1711.06767`.

F. Toyama, Y. Fujiki, K. Shoji, and J. Miyamichi. Assembly of puzzles using a genetic algorithm. In *2002 International Conference on Pattern Recognition*, volume 4, pages 389–392 vol.4, 2002. doi: 10.1109/ICPR.2002.1047477.

Anne D. Williams. *Jigsaw puzzles: an illustrated history and price guide*. 1. Wallace-Homestead, Radnor, Pennsylvania, 1990. ISBN 0-87069-537-1.

Rui Yu, Chris Russell, and Lourdes Agapito. Solving jigsaw puzzles with linear programming. *CoRR*, abs/1511.04472:139.1–139.12, 01 2016. doi: 10.5244/C.30.139.

Senhua Zhao, Yue-Jiao Gong, and Xiaolin Xiao. Multi-strategy evolutionary computation for automated jigsaw puzzles. In Haiqin Yang, Kitsuchart Pasupa, Andrew Chi-Sing Leung, James T. Kwok, Jonathan H. Chan, and Irwin King, editors, *Neural Information Processing*, pages 50–62, Cham, 2020. Springer International Publishing. ISBN 978-3-030-63833-7.

List of Figures

List of Tables

A. Attachments

A.1 Project documentation

In this section, we will describe the structure of the implementation of our solution and the necessary requirements for its work. Our project has been written and tested in Python version 3.10. We have also used several additional libraries, such as Numpy[1], Deap[2], Shapely[3], and others. All required libraries and their versions are listed in file `requirements.txt` and can be installed with the following command:

```
python -m pip install --user -r requirements.txt
```

The structure of our solutions is as follows. The solver for each type of piece shape is in its own file. We have implemented a total of 6 of them:

1. `square_pieces_solver.py`
2. `rectangular_pieces_solver.py`
3. `cross_cut_shredder_solver.py`
4. `right_triangle_pieces_solver.py`
5. `triangle_pieces_solver.py`
6. `random_rectangular_pieces_solver.py`

The basis of all solvers is the solution proposed for square pieces, which is implemented in file `square_pieces_solver.py`. It is represented as a class in which all operators of the genetic algorithm are implemented. In order to use it, it is necessary to create and initialize an instance of this class, and then call the `solve` method on it. In each iteration of the algorithm, the best solution is saved in the separate file. All other solutions are inherited from this and, in some cases, rewrite the implementation of the operators. This structure allows us to extend our solution, since we only need to inherit from one of the classes and rewrite the necessary methods as needed.

Since the calculation of fitness values and the crossover process can run in parallel, we use a module in Python to work with multiple processes. In our experiments, we ran our solutions in 8 processes, but we allow changing this value.

We have also tested the complex parts of our implementation where it is easy to make mistakes, such as checking for cluster intersection and calculating the fitness value. To run the tests, you can use the following command:

```
python -m pytest -vv tests/
```

[1] `https://numpy.org/doc/stable/`
[2] `https://deap.readthedocs.io/en/master/`
[3] `https://shapely.readthedocs.io/en/stable/index.html`

We would like to clarify that all methods and classes in all files are documented with comments.

We attach to our work the functions we have implemented to work with images. They can be found in the file `image_utils.py`. We also attach to the solution a script `create_puzzles.py` for splitting the pictures into pieces of all the forms mentioned in this work. The resulting pieces are stored in a certain order, necessary for the subsequent identification of the correct solution. Note that this correct solution is in no way used in the work of the algorithm, but is only needed for its subsequent comparison with the proposed solution. Let us specify that at the beginning of each algorithm we randomly mix the pieces and give their edges random indices, so that the correct solution is not always the same and does not depend on the order of the pieces. When we divide images into random rectangles, the correct solution with already shuffled pieces is saved in a separate file. In other cases, we only save the pieces in the correct order, and the correct solution is always determined by itself before the algorithm starts.

In the following, we describe how to use our proposed solutions. Recall that each solver is in its own file. All of these scripts work in two modes. The first is the experiment mode, in which we assume a comparison of the correct solution with the proposed solution. To do this, this correct solution must be provided to the algorithm as input in the case of pieces in the form of random rectangles, or as pieces sorted in the correct order in the case of pieces of all other shapes. To work in this mode, we propose to divide images into pieces using our script as follows, for example:

```
./create_puzzles.py -p imgs_from_papers/pomeranz_db1/1.jpg \
-f square -s 28 28
```

In this mode, the algorithm will output the value of the *neighbor comparison metric* after each iteration, and it will also save the best individual in a special directory.

In the second mode, we do not assume knowledge of the correct solution, and we give each script an input directory of pieces in any order. In this case, after each iteration, the algorithm will only output the fitness value of the best individual, and will save it in a special directory. We have given examples of usage of our solvers for all types of pieces in the file `examples.txt`.

We also attach a CD to our solution, which contains the best individuals for each of the experiments performed in this work, in order to make the results of our work more illustrative. Part of the attachment to the work are also the datasets on which the experiments have been performed.