A Genetic Algorithm-Based Solver for Small-Scale Jigsaw Puzzles

Wenjing Guo, Wenhong Wei^(✉), Yuhui Zhang, and Anbing Fu

School of Computer, Dongguan University of Technology, Dongguan 523808, China
weiwh@dgut.edu.cn

Abstract. In this paper, we present a genetic algorithm-based puzzle solver, which is mainly used to solve small-scale puzzle problems. We introduce a new measurement function that improves its accuracy by normalizing the Mahalanobis distance and the Euclidean distance between two puzzle pieces. By calculating the difference between edges of two puzzle pieces and using the genetic algorithm to assemble pieces correctly, two “parent” solutions are merged into one improved “child” solution. Using the idea of local search, it avoids the problem of local optimum solutions brought by the genetic algorithm, which greatly improves the accuracy of the puzzle.

Keywords: Computational jigsaw puzzle · Genetic algorithm · Local search

1 Introduction

The Puzzle is a game everyone can play since they are little. A picture is divided into N pieces of different non-overlapping parts and shuffled the order. Players must use the shape and color information of each piece to reconstruct the original image. The Puzzle problem has been proved to be a Non-deterministic Polynomial (NP) complete problem [1]. Although it is often regarded as a game, the idea of solving puzzles usually plays an essential role in many fields like archeology and medicine, the repair of ancient artifact fragments, the restoration of fractures and displaced bones. With the latest development of computer science and engineering, researchers are able to create computer puzzle solvers. The algorithms of the puzzle solver are used to resolve the basic problems in various fields such as machine learning, computer vision, and bioinformatics.

Jigsaw puzzle was first produced in 1760 by a London engraving and mapmaker artist, John Spilsbury. Freeman [2] and Garder were the first people in the scientific community who tried solving the problem theoretically. In 1964, they introduced a square solver which can solve nine pieces of the puzzle. Shortly, the research focus has shifted from shape-based to color-based. Since then, more and more scholars have begun to study the puzzle problem. In 1988 [3], Wolfson H., et al. studied real-world puzzles, which were based on pieces with specific shapes and directions. Soon, Chung MG et al. [4] proposed a method of puzzle based on shape and color. Later, people began to study square puzzles, which pieces cannot distinguish directions. In 2010, Cho et al. proposed a probability-based puzzle solver that can handle up to 432 puzzle pieces and

lead prior knowledge about puzzle problems. One year later, Yang et al. introduced a particle filtering-based puzzle solver, which improved their results. In the same year, Pomerantz [5] et al. first announced a fully automatic square puzzle solver, which can handle up to 3000 pieces of a puzzle. In 2012, Andalo, F. A., et al. [6] suggested a method to solve image problems based on a simple quadratic programming formula. In 2013, Sholomon et al. presented a large-scale puzzle problem based on genetic algorithms. Anne D [7] further explored this problem and considered a more general variant, that is, neither the direction of the pieces nor the size of the puzzle is known. [8] proposed a new method for solving small pieces of the puzzle. In 2019, [9] introduced a new puzzle solver, which performs well in the case of unknown direction and piece size.

In its most basic form, each puzzle solver needs a fitness evaluation function to determine the compatibility between adjacent puzzle pieces and find a puzzle strategy as accurate as possible. Those researchers mainly used (the Sum of Squared Distances (SSD) in LAB color space from Cho et al. [10]) the method of calculating the color distance in establishing the similarity measurement function. Color space distance refers to the difference between the two pieces. In general, the larger the distance, the greater differences between these two pieces, vice versa. When calculating the color distance, similar to calculating the Euclidean distance between two points, the value between the pieces calculated in this way is often not very effective. Therefore Gallagher [11] also proposed a powerful pairwise compatibility measure, Mahalanobis Gradient compatibility (MGC), which penalizes gradient changes when gradient changes exceed expectations. Gallagher mainly adopts the method of calculating Mahalanobis distance to construct compatibility function. Mahalanobis distance is created by P.C. Mahalanobis, an Indian statistician, which represents the distance between a point and a distribution. It is an effective method to calculate the similarity of two unknown sample sets since it takes into account the connections between various features. Based on this, we take the normalization treatment of color distance and Mahalanobis distance to build a new measurement function. Gallagher conducted a comparative study ([11], Table 4) with 432-piece-scale puzzles, and the results showed that the accuracy was only slightly improved compared to Pomerantz et al. (95.1% vs. 95.0%). Supplementary materials provided by Pomerantz et al. [12] showed there was much room for improvement in many puzzle problems. In this paper, we use the powerful technique of genetic algorithm [13] as a puzzle strategy. Toyama et al. tried the design of a ga-based solver, but it could only complete puzzles with a size of 64 pieces. We advanced a new puzzle solver based on the genetic algorithm, using the crossover puzzle strategy proposed in [14]. In addition to this, we improve it by adding a local search method, so that the final puzzle results can avoid falling into the local optimum. The research exhibits that our solver can complete the picture in small-scale puzzle problems.

2 Genetic Algorithm

A GA is a simulation of natural selection and the calculation of the mechanism of biological evolution, a method of searching through simulation of natural evolution. Because examining all possible solutions to a particular problem is generally considered infeasible, genetic algorithms provide a process of searching within solutions to a problem.

First, the initial population of a candidate solution, also known as a chromosome, is randomly generated. Each chromosome is a complete candidate solution to the puzzle problem, and arrange them into puzzle pieces, where each piece is viewed as a gene on the chromosome. By imitating the crossover, mutation, and evolution of biological heredity, the solution in the population is improved continuously. Thus, the optimal solution (i.e. correct image) is obtained.

In the process of simulating natural selection, the reproduction rate of chromosomes, where the number of times each chromosome is selected and reproduced, is proportional to its fitness value. The fitness value is a score obtained through the fitness measurement function, which measures and evaluates the quality of a given solution. Therefore, a “good” solution will produce more offspring than other solutions. Also, good chromosomes are more likely to reproduce with other good chromosomes, a process known as “crossover”. Operators should allow both parents to pass on their better traits to offspring.

The success of the genetic algorithm mainly depends on the selection of appropriate chromosome representation, crossover operator and fitness function. The chromosome representation and the crossover operator must allow two excellent “parent solutions” to merge into a better “child” solution. The fitness function must correctly detect parts of the chromosome are more promising for transmission to the next generation.

3 The Puzzle Problem Based on Genetic Algorithm

By using the pseudo code of Algorithm 1, the underlying genetic algorithm framework for solving the puzzle problem is determined. As mentioned before, the genetic algorithm first defines and initializes a population, and each chromosome represents a possible solution. Specifically, our genetic algorithm starts from 1000 random placements. In each generation, the fitness function is used to evaluate the whole population (as described below), and a new offspring is generated through the selection and crossover of chromosome pairs. We use the common roulette wheel selection method to choose chromosomes. The probability of a chromosome is selected based on the value we calculated from the fitness function. The larger the value, the greater chances a chromosome will be chosen. Having provided a framework overview, we now describe the critical components of genetic algorithms, such as chromosome representations, fitness functions, and crossover mutation algorithms in detail.

Pseudo code

```

1: population ← generate 1000 random chromosomes
2: for generation number = 1 → 100
3:   evaluate all chromosomes using the fitness function
4:   new population ← NULL
5:   copy 2 best chromosomes to new population
6:   while size(new population) ≤ 1000 do
7:     parent1 ← select chromosome
8:     parent2 ← select chromosome
9:     child ← crossover(parent1, parent2)
10:    add child to new population
11:   end while

```

```

12: population ← new population
13: end for

```

3.1 Fitness Function

In the GA algorithm, fitness describes the relative probability an individual survives, and fitness functions are used to calculate it. Here, each chromosome represents a disorderly picture. In the puzzle, we are not aware of all the initial conditions, so calculating fitness is not a simple task. We accumulate the compatibility of every two adjacent pieces and divide the fitness factor by the result. (to make the chromosome with smaller difference obtain greater fitness).

We define the possibility of two pieces being adjacent pieces as compatibility. The greater the compatibility, the greater the probability of two pieces are adjacent pieces. For every given two pieces, we define a set of spatial direction $\{L, R, T, D\}$ to represent the left, right, top, and down respectively. Then for each direction, we calculate the difference between the two pieces. $C(x_i, x_j, L)$ is used to represent the difference of x_i in the left direction of x_j . We will explain our difference measurement function later.

Based on the compatibility measurement function, many people have suggested suitable methods. Cho et al. [10] proposed to calculate the compatibility value of the two pieces by calculating the sum of the square of each pixel's spatial distance. Pomeranz et al. [12] also proposed some optimizations. It should be noted that using this method, for maximum compatibility, we need to calculate their minimum differences.

We propose a new compatibility measurement method. By calculating the sum of the color space distance and the Mahalanobis distance, we get a new set of differences. In order to make the color space distance and Mahalanobis distance comparable, we reduce the weight of the two methods by normalization so that they can get the best results. Suppose there are x_i and x_j represents two pieces of size $K * K * 3$, and K means the height and width of a piece (unit is pixel). So the difference measurement function is:

$$C(x_i, x_j, r) = D(x_i, x_j, r) + DG(x_i, x_j, r) \quad (1)$$

Here D represents x_i, x_j , the color space distance in the R direction:

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

DG stands for Mahalanobis distance, a compatibility measurement algorithm proposed by [11], which calculates compatibility by calculating the gradient value of the color change of two pieces. The formula is as follows:

$$DG(x_i, x_j, r) = \sum_{s=1}^K (\Lambda_{LR}^{ij}(s) - E_{LR}^{ij}(s)) V_{LR} - (\Lambda_{LR}^{ij}(s) - E_{LR}^{ij}(s))^T \quad (3)$$

$$L_{LR}^{ij}(s) = \delta_j(s, 1) - \delta_i(s, K) \quad (4)$$

$$E_{LR}^{ij}(s) = 1/2 (\delta_i(s, K-1) - \delta_i(s, K-1) + \delta_j(s, 2) - \delta_j(s, 1)) \quad (5)$$

It is worth mentioning that $C(x_i, x_j, r) \neq D(x_j, x_i, r)$. Obviously, for greater compatibility, the differences between the two pieces should be as small as possible.

The fitness function also needs to consider the calculation speed. When the program is running, it needs to calculate the fitness of each chromosome in each generation. This consumed time cannot be ignored. Since it's impossible to calculate the degree of difference between pieces for each fitness calculation, so we use a table to cache the degree of difference in the senior of $(n * m)^2 * 2$ and includes the difference degree between all pieces. In this way, we don't need to calculate the difference degree every time, but add the values in the cache.

This is our calculation formula of fitness

$$\sum_{i=1}^N \sum_{j=1}^{M-1} C(x_{i,j}, x_{i,j+1}, r) + \sum_{i=1}^{N-1} \sum_{j=1}^M C(x_{i,j}, x_{i+1,j}, r) \quad (6)$$

Which r and d represent the right and down directions

3.2 Crossover and Mutation

Problem Definition

As mentioned above, we use an $(N * M)$ size chromosome to represent a disordered picture. During initialization, we will give each piece a number based on the original picture (mainly used for the result evaluation after the completion of the puzzle). This method is easy to use and suitable for the assessment. The main problem now is how to define a suitable crossover algorithm, as this method generates a child chromosome from a given two parent chromosomes. It should be able to retain the “good traits” of the parent generation to the child generation (such as complete pieces already been placed), to obtain a better puzzle result. If only a random algorithm is used, it may lead to a completely random selection from the parent's chromosome, causing some pieces to repeat or miss, and the puzzle to fail. Thus, as long as the problem of crossover is solved, the GA algorithm can be improved.

Even if the validity problem is solved, we still need to think carefully about the crossover operator. Crossover is based on the fitness of two chromosomes, which is determined by the compatibility between adjacent pieces. Therefore, crossover cannot precisely locate the correct position but select a piece with the highest similarity according to the fitness. The population is completely random from the beginning, and then gradually improves. We can reasonably assume that after several generations of evolution, more pieces will be placed in the correct position. Considering the limitations of the fitness function, it cannot fully determine whether a piece is in the right position. Hence, we predict some fragments may be the wrong position. So how to find a piece that belongs to the correct location is vital. A properly positioned piece should be considered a good feature and passed to the offspring. The crossover operation must consider how to transfer a correctly spelled fragment so it can move as a whole.

Once we solve the problem of moving a fragment, we need to solve the problem of fragment misplaced. What traits will pass to an offspring? Heuristics methods may be applied and used to distinguish whether prices are correct.

Proposed Methods

Given two parent chromosomes, for example, two completely different chromosomes (with the different distribution of internal pieces), the crossover operation will gradually build a word chromosome from the inside, and take the two parent chromosomes as the prototype to generate a complete offspring (Fig. 1).

Fig. 1. Shows the process of crossover. Randomly select two parents, the parents have the same fragment, 2, 3, 6. We start with piece 2, and the algorithm will first select the same fragments around 2 piece in the parent generation (the number of adjacent fragments is the same in a certain direction). Based on this principle, we get ① ② ③ pictures. Next, we choose the best-buddy piece. When there is no best-buddy piece, we traverse all the boundaries to find the piece with the highest matching compatibility.

This crossover operation starts with a randomly selected piece. Then gradually acquire the available pieces (the highest possible unused adjacent pieces). Merging with neighboring pieces, a picture obtained by this method must be continuous and complete. This operation continuously seeks the most compatible piece from the candidate pool, until there are no more candidates, which the puzzle is completed. Because each piece can only appear once, which will not cause duplicate piece, and considering the size of the image is known, this operation will not exceed the original boundary. It also ensures this operation can generate an effective picture (no repeating or missing pieces). Depending on the kernel growth vector, the position of each piece may be moved before completing the picture. For example, if a lower-left corner is randomly selected as the

first piece, then the kernel will tend to grow to the upper-right. The first piece may be in any four corners or any other location. With this feature, the position of a piece is independent, and the whole fragment can be shifted.

Based on previous thoughts, we still need to solve the following issues: (1) How to select a suitable piece from the candidate pieces, (2) and where to place it in the child generation. Given a starting point, an image with only one piece, we can mark all reachable boundaries. The boundary of this piece is composed of itself and directions, which is represented by a pair (x_i, R) . The crossover process consists of three stages. The method proposed in [14] is used here. The first stage: For a known boundary, if the program detects a candidate pieces, this piece and the fragment composed this candidate pieces both exist in the parents (The parent has the same piece in the same direction). Then we believe that these two pieces are correct and should directly be retained in the offspring. If more than one of these conditions exists in the parent at the same time, choose one randomly (so the order is not the most important, because children generation will keep all such pieces). A used piece cannot be used again. If a match occurs with a used piece, then the piece is ignored. The second stage: If there is no piece matching the parents at the same time, then enter the second stage.[14] is quoted here as the best-buddy matching piece. Assuming there are two pieces here, if each piece is the best match to the other piece (the direction is opposite), then we call it the best-buddy match piece. If pieces x_i, x_j are best-buddy matching pieces, then:

$$\begin{aligned} \forall x_k \in \text{Pieces}, C(x_i, x_j, R_1) &\geq C(x_i, x_k, R_1) \\ &\text{and} \\ \forall x_p \in \text{Pieces}, C(x_j, x_i, R_2) &\geq C(x_j, x_p, R_2) \end{aligned}$$

Pieces represent all pieces, R_1 and R_2 represent a set of opposite directions. In the second stage, the program checks whether a piece in the parents so there is a best-buddy matching piece in the optional boundary. If it exists, the piece will be selected and retained in the offspring. At the same time, if the program detects multiple best-buddy matching pieces, it will choose one randomly to perform the puzzle problem first. If this candidate piece already used in the child generation, the process will skip this match and look for new matches or enter the third stage. The third stage: if no best-buddy matching piece is selected, the process will randomly select a piece and select the piece with the highest degree of matching in the current direction as its child generation.

We propose two new mutation methods. After the crossover stage is the mutation stage. For every child generation, we will select the best offspring for mutation operation according to the fitness. For each piece in the child chromosome, there is a chance to mutate. The mutation will select a piece and a direction. We will find the best matching piece, and exchange the best matching piece with the adjacent pieces in the same direction. If the fitness of the offspring increases after this exchange, it means the mutation succeeds, and the result will be retained. If the fitness after the exchange decreases, it means the mutation failed, and we will restore this mutation.

Ranking adjacent piece achieves another mutation method, and we call it the best boundary ranking. In the mutation stage, for each generation, we will select the best offspring for mutation operation according to the fitness. For each piece in the child chromosome, there is a chance of mutation. We will randomly select a piece, and calculate

Fig. 2. Shows the process of mutation. There are two cases of mutation, successful mutation and mutation failure. When the mutation is successful, as shown in the left picture, the correct mutation will increase the compatibility of the picture, so that the mutation is retained. If the compatibility of the picture is reduced, as shown in the right figure, the mutation will be restored.

the sum of the similarity ranking of the two adjacent pieces (If these two pieces are the best-buddy matching pieces for all directions, the result is $4 + 4 = 8$). Then calculate the sum of the similarity ranking after this exchange. If the sum of the similarity rankings after this exchange is less than the sum of the previous similarity rankings, it means this mutation was successful, and the result is retained. Otherwise, it means that the mutation failed, and we will restore this mutation (Fig. 2).

Fig. 3. Shows the process of solving a 646 pieces puzzle. The pictures almost completed in the fourth generation.

In summary, the program repeats these three stages until there are no candidate pieces available, and the program ends. Crossover always take turns in the first, second, and third stage. We only consider starting the puzzle from the boundary to maintain the

continuity of kernel growth, and after each successful match, we will return to the first stage. The code description for this stage as follows:

Code description:

The first stage: Within the assembled boundary, find whether two pieces are in the parent's chromosome at the same time.

If yes, add it to the child chromosome, update the boundary. Otherwise, enter the second stage.

The second stage: Within the assembled boundary, find the best-buddy matching piece in the parent chromosome

If yes, add it to the child chromosome, update the boundary. Otherwise, enter the second stage.

The third stage: Within the assembled boundary, randomly searching for the best matching piece to add to the child chromosome (Fig. 3).

Principle

In the GA framework, good traits should pass to the child generation. In this case, the good fragment of the parent generation can preserve well because this algorithm does not rely on the position. Each right fragment corresponds to the correct position of each puzzle piece. It should be noted that the relative spatial position of x_i and x_j is the key to solving the puzzle problem. However, every chromosome has all the pieces of the puzzle. Due to the randomness of the first generation, the program must actively seek the best features between pieces. We did some research. Assuming parents share a good characteristic, which is the reason for their survival and choice. In other words, if both parents have the same characteristic, we treat it as highly accurate and keep it to the offspring. Not all frequent features will pass to the next generation. Because of the kernel growth algorithm, some of the same pieces may match other pieces prematurely and occupied, so the correct pieces cannot match them. As for the second stage, without the same characteristics, people tend to randomly select a parent and start with it. Another option is to choose a piece with the highest compatibility or the best-buddy matching piece. Since the position of a piece in the parents is random, even the best-buddy matching piece may not be the correct match. Two pieces are the best-buddy matching pieces and have the same characteristics in the parents, which is a sign for a valid match. Another point of view is that each chromosome contains some correct fragments. Passing the right gene from parent to offspring is the core of the genetic algorithm. In addition, if each parent contains a correct fragment, and those fragments partially overlapped. The overlapping portion will pass to the child generation in the first stage, the parents complete the inheritance of the correct fragment in the second stage, and to form larger and more correct fragments. As for the third stage, we can conclude this: the genetic algorithm tries multiple puzzles at different locations simultaneously. Only fragments seem to be right can be passed on to the next generation, which embodies the principle of the fine traits of natural selection theory.

4 Experimental Result

Cho et al. [10] introduced three measures to evaluate the complete puzzle, two of which repeatedly used in the work. One is the direct comparison method, which measures the

proportion of debris in the correct position. The other is the comparison of adjacent pieces, which measures the right proportion of adjacent pieces in the puzzle. In this experiment, we adopted the direct comparison method, since it's a more intuitive way to judge the completion of a puzzle.

In our experiments, we used the same genetic algorithm parameters as pseudocode 1. The population has 1000 chromosomes. In each generation, we kept the best 2 chromosomes (an elitist measure). The remaining population will be generated by the crossover operator with a mutation rate of 5%. Parent chromosomes are produced by roulette wheel selection, each time generating an offspring. GA runs 100 generations at a time.

We selected a set of image data, which contains 25 pictures, and divided each picture into different size pieces. We run the genetic algorithm 10 times on each image, each time using a different random seed, and record the best, worst, and average (and standard deviation). Table 1 lists the results of our genetic algorithm on each set. Interestingly, despite the random nature of GA, the results of different runs are close, which proves the robustness of genetic algorithms.

Table 1. Best, worst, and average results of variety pieces number

Number of pieces	Best	Worst	Average
256	100	99.21875	99.53125
646	99.69040	97.52311	98.45201
1196	87.95986	79.93311	84.94983

We compared the effectiveness of three compatibility measurement functions on different picture puzzles. The data collected from the 646 pieces puzzle showed in Table 2. We can conclude that the efficacy of using the color-space distance measurement function alone is not as good as using the Mahalanobis distance measurement function. The best result is achieved by combining the two functions with normalization. We can see that through the normalized function, the degree of restoration reached 98%, while using the Mahalanobis distance and color-space distance methods is 96% and 97%, respectively. Compared with the previous method, our method has greatly improved.

We also compared whether to use the local search for the puzzle. Using the local search as the mutation function, the data are shown in Table 3. When testing the data in this section, we use the 646 pieces puzzle, and the normalized measurement function of two distances as the benchmark.

Table 2. Best, worst, and average results of different measure function

	Best	Worst	Average
Color space distance based measure function	97.5232	95.04643	96.037142
Mahalanobis distance based measure function	98.7616	95.97523	97.647052
Proposed	99.69040	97.52311	98.45201

Each group of test pictures divided into pieces of different sizes, and Table 1 shows the results. Here are the best, worst, and average results.

Table 3. Best, worst, and average results of different mutation function

	Best	Worst	Average
No mutation	91.02167	83.90092	88.885448
Fitness based mutation function	99.69040	97.52311	98.45201
best boundary ran-king based mutation function	99.38080	97.21362	97.86377

Records the results of the puzzle using three different measurement functions. Using the 646 pieces puzzle, there are the best, worst, and average results under those three measurement functions.

Records the results obtained without mutation and utilizing different mutation algorithms. Using the 646 pieces puzzle, there are the best, worst, and average results under different mutation algorithms.

5 Discussion and Future Work

This article introduces a puzzle solver based on the genetic algorithm, which can accurately solve small-scale puzzle problems while significantly reduced the time. It improved the fitness function and the accuracy of the puzzle pieces. Based on our work, we can further study on solving the puzzle problem in more complex situations like unknown direction pieces, various missing pieces, and the accuracy of large-scale puzzles. To improve the efficiency of existing solver, solve the problem of unknown size and 3D Puzzle. It can also provide a reference for solving problems in other fields.

Acknowledgement. This work was supported by the Key Project of Science and Technology Innovation 2030 supported by the Ministry of Science and Technology of China (Grant No. 2018AAA0101301), the Key Projects of Artificial Intelligence of High School in Guangdong Province (No. 2019KZDZX1011) and Guangdong Provincial Science and Technology Plan Projects (No. 2016A010101034).

References

1. Demaine, E.D., Hearn, R.A.: Playing games with algorithms: algorithmic combinatorial game theory (2001)
2. Freeman, H., Garder, L.: Apictorial jigsaw puzzles: the computer solution of a problem in pattern recognition. *IEEE Trans. Electron. Comput.* **13**(2), 118–127 (1964)
3. Wolfson, H., Schonberg, E., Kalvin, A., Lamdan, Y.: Solving jigsaw puzzles by computer. *Ann. Oper. Res.* **12**(1), 51–64 (1988). <https://doi.org/10.1007/BF02186360>

4. Chung, M.G., Fleck, M.M., Forsyth, D.A.: Jigsaw puzzle solver using shape and color. In: Proceedings of the 1998 Fourth International Conference on Signal Processing. ICSP 1998. IEEE (1998)
5. Yang, X., Adluru, N., Latecki, L.J.: Particle filter with state permutations for solving image jigsaw puzzles (2011)
6. Andalo, F.A., Taubin, G., Goldenstein, S.: Solving image puzzles with a simple quadratic programming formulation. In: Graphics, Patterns & Images. IEEE Computer Society (2012)
7. Williams, A.D.: The Jigsaw Puzzle: Piecing Together a History. Berkley Publishing Group, New York (2004)
8. Son, K., Moreno, D., Hays, J., Cooper, D.B.: Solving small-piece jigsaw puzzles by growing consensus. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1193–1201. IEEE (2016)
9. Son, K., Hays, J., Cooper, D.B.: Solving square jigsaw puzzle by hierarchical loop constraints. IEEE Trans. Pattern Anal. Mach. Intell., 2222–2235 (2019)
10. Cho, T.S., Avidan, S., Freeman, W.T.: A probabilistic image jigsaw puzzle solver. In: The Twenty-Third IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2010, San Francisco, CA, USA, 13–18 June 2010. IEEE (2010)
11. Gallagher, A.C.: Jigsaw puzzles with pieces of unknown orientation. In: 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE (2012)
12. Pomeranz, D., Shemesh, M., Ben-Shahar, O.: A fully automated greedy square jigsaw puzzle solver (2011)
13. Holland, J.H.: Adaptation in Natural and Artificial System. MIT Press, Cambridge (1992)
14. Sholomon, D., David, O., Netanyahu, N.S.: A genetic algorithm-based solver for very large jigsaw puzzles. In: IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2013. IEEE (2013)