Making Puzzles Less Puzzling: An Automatic Jigsaw Puzzle Solver

Cris Zanoci
Stanford University Computer Science
Stanford University Physics
czanoci@stanford.edu

Jim Andress
Stanford University Computer Science
Stanford University Mathematics
jandress@stanford.edu

June 6, 2016

Abstract

We implemented an algorithm to solve jigsaw puzzles in which the scrambled square puzzle pieces have unknown translations and rotations. We assume no input other than the scrambled pieces and the dimensions of the original image. Our algorithm reformulates the puzzle solving problem as a search for a minimum spanning tree in a graph, subject to additional spatial consistency constraints. In order to provide for quick union and find operations while searching for the MST, we implemented a modified Disjoint Set Forest data structure. Our results are on par with those of state of the art puzzle solving algorithms, and we demonstrate successful reconstruction of large, high resolution photos.

1. Introduction

The jigsaw puzzle is one of the most popular puzzle games, known and loved by almost everybody from an early age. The puzzle consists of non-overlapping pieces that have to be assembled into an output image. The game dates back to as early as the 18th century, and the first computational approach to solving this problem was introduced by Freeman in 1964 [4]. Today, with the recent developments in machine learning and computer vision, automatic puzzle solving has drawn the attention of researchers worldwide [2, 5, 10, 11, 12].

In addition to being an interesting task on its own, the solution to this problem has applications to a number of related fields. For instance, automatic reconstruction of fragmented objects is of great importance in archaeology and art restoration [1]. Moreover, puzzle solvers have also been applied to assembling shredded documents [7] and descrambling speech [15].

The jigsaw puzzle come in a variety of forms: traditional interlocking puzzles, 3-dimensional puzzles, monochromatic puzzles, etc. However, for this project, we focus on the more difficult case in which all pieces are square, such that no information about the location or orientation is provided by the pieces. Hence the puzzle has to be solved using image information alone. We use publicly available, high-resolution pictures as our input, which we later divide into squares. We will assume the puzzle to be rectangular and its dimensions, in terms of the number of pieces, are considered known.

Following the definitions in [5] we introduce two types of puzzles: Type 1 puzzles consist of square pieces of known orientation, but unknown position (i.e. two pieces can fit together in 4 different ways), while Type 2 puzzles have both unknown orientation and position (i.e. two pieces can fit together in 16 different ways). Type 2 puzzles are significantly more challenging to assemble because the number of possible reconstructions exceeds that of an equivalent Type 1 puzzle by a factor of 4^K, where K is the number of pieces. In this project we focus on Type 2 puzzles, since our approach is powerful enough to solve them efficiently. Our metric for a successful reconstruction is the number of times two pieces that are adjacent in the original image are adjacent in our final solution.

Our goal is to find an automated procedure for solving the puzzle. Although some earlier approaches like [2] incorporated additional user-supplied information (e.g. a small number of "anchor" patches whose true locations are known), we implement more recent methods which focus on the fully unsupervised version of the problem.

2. Previous Work

2.1. Review

There is a rich history of puzzle solving techniques. The initial approaches only made use of shape information and ignored image data [13]. Later work by Chung et al. [3] ex-

amined RGB values together with hue and saturation along the pieces' edges. It was only in 2008 when Nielsen et al. [9] implemented a solver based on a mixture of image features and shape information that could solve puzzles with about 320 pieces.

More recent assembly methods can be classified into two main categories: greedy methods [5, 10, 12] and global methods [2, 11, 14]. The greedy methods start from initial pairwise matches and successively extend to larger components. Among these, the most important ones are Gallagher's constrained minimum spanning tree approach [5] and Son's loop constraints between pieces [12], which relies on gradually reconstructing larger cycles within the puzzle. Global methods, on the other hand, search directly for a solution by maximizing a global compatibility function. Cho et al. [2] proposed a probabilistic solver which achieves approximated reconstructions by using a Markov Random Field and loopy belief propagation solution. Another recent paper by Sholomon [11] showed that genetic algorithms can be successfully used to reconstruct large jigsaw puzzles.

Although the game was proven to be NP-hard, the successful approaches mentioned above are known to work with puzzles of thousands of pieces. Table 1 summarizes the results of a few papers that we studied. Notice that all of them achieve an accuracy of more than 90% under the neighbor comparison metric, which we will explain in Section 4. We will refer to this table in future sections in order to compare our results

2.2. Our Contribution

For our project, we focused on the minimum spanning tree reconstruction technique described by Gallagher [5]. Our main contribution is an efficient data structure representation of the problem using Disjoint Set Forests which guarantees a fast reassembly. For the other parts of the solver, we closely follow the description in [5], but we implement each component from scratch.

While investigating the accuracy of our implementation, we explore two pairwise compatibility metrics for the pieces, the sum of squared distances and Mahalanobis gradient compatibility, and show that the latter performs better. We also implement two measures for evaluating the puzzle reconstruction accuracy and show that our algorithm can reconstruct real images reliably.

3. Methods

Almost all the papers mentioned above start by computing a pairwise compatibility function for the square pieces. This metric tells us how likely two pieces are to be next to each other in the final reconstruction. This is a crucial part of the algorithm, since our solver later uses these scores to assemble the pieces.

Once we know the compatibility of any two pieces, we need an algorithm for assembling the squares. In our case, we treat the pieces as vertices in a graph and compatibility scores as edges. Then we apply a modification of Kruskal's minimum spanning tree algorithm to connect the pieces.

In practice, even after the solver has reassembled most of the pieces, we are not guaranteed that the solution has a rectangular shape. Therefore, we need a post-processing stage as described in [5]. First, since the dimensions of the initial image are known, we trim the solution to make sure that it fits inside the rectangle outlined by the initial image. Next, we fill in the gaps with the leftover squares according to the compatibility scores computed in the beginning.

3.1. Edge Compatibility Metric

The first important compatibility metric we explored is the dissimilarity-based compatibility introduced by Cho *et al.* [2]. It sums the squared distances (SSD) between pixels on either side of the common edge of two potentially neighboring pieces x_i and x_j. If the size of each piece is $P \times P$ pixels, then the dissimilarity between the two when x_i is placed on the left of x_j is:

$$C_{LR}(x_i, x_j) = \sum_{c=1}^{3} \sum_{p=1}^{P} (x_i(p, P, c) - x_j(p, 1, c))^2 \quad (1)$$

where p is one specific pixel in the range 1 to P, and c is one of the three color channels. Intuitively, two adjacent pieces should have similar pixels along their common edge and therefore a small dissimilarity score.

Next, we consider the Mahalanobis gradient compatibility (MGC), proposed by Gallagher [5], which measures the similarity of the two gradient distributions on either side of the common edge between two neighboring squares. We begin by computing the gradient distribution for each puzzle piece x_i. The first step is to approximate the gradient in each color channel of the piece along an edge by taking the difference of the pixel values in the two outermost rows or columns (depending on whether we are considering a vertical or horizontal edge) of the piece. For example, the gradient along the right edge of the piece x_i is roughly

$$G_{iL}(p, c) = x_i(p, P, c) - x_i(p, P-1, c) \quad (2)$$

where the notation is as defined above. We can then use this $P \times 3$ matrix G_{iL} to approximate the gradient distribution on the left edge of piece x_i. For example, we can compute the mean gradient value for a channel c as

$$\mu_{iL}(c) = \frac{1}{P} \sum_{p=1}^{P} G_{iL}(p, c) \quad (3)$$

We similarly compute S_{iL}, which is the 3×3 covariance matrix encapsulating the inter-dependencies of the gradient

Approach	Author	Year	Largest Puzzle	Direct Metric	Neighbor Metric	Perfect
Constrained MST	Gallagher [5]	2012	9600	82.2 %	90.4 %	9/20
Genetic Algorithm	Sholomon [11]	2013	10375	80.6 %	95.2 %	-
Loop Constraints	Son [12]	2014	9801	94.7 %	94.9 %	12/20
Linear programming	Yu [14]	2015	3300	95.3 %	95.6 %	14/20

Table 1: Reconstruction performance of four recent methods on Type 2 puzzles. The last three columns are based on the MIT dataset, with $P = 28$ and $K = 432$.

values in different color channels. These same computations are applied to each edge of the piece, meaning that we are finally left with four mean vectors and four covariance matrices, one per edge.

Once the above information has been computed for each puzzle piece, we can define the dissimilarity between edges on two pieces via an application of the Mahalanobis distance, which is used to measure the distance from a point to a probability distribution. In our specific case, we have estimates of the distribution defining the gradient along each of the edges, and we want to determine whether the gradient between the two edges is consistent with these distributions.

We therefore first compute the gradient which would result by placing the two edges next to each other. For example, the resulting gradient from the right edge of piece x_i to the left edge of x_j can be computed as

$$G_{ijLR}(p, c) = x_j(p, 1, c) - x_i(p, P, c) \tag{4}$$

We then take each gradient vector along this edge and compute its squared Mahalanobis distance to the gradient distribution along the right edge of piece x_i, summing the distance across pixels. This gives us a dissimilarity

$$D_{LR}(x_i, x_j) = \sum_{p=1}^{P} (G_{ijLR}(p) - \mu_{iL}) S_{iL}^{-1} (G_{ijLR}(p) - \mu_{iL})^T \tag{5}$$

Again, Equation 5 is only a measure of the distance from the cross-edge gradient to the distribution to the left piece. To compute a symmetric distance function, we modify Equations 4 and 5 to use the edge of the right piece, giving us $D_{RL}(x_j, x_i)$. Our final distance metric is then

$$C_{LR}(x_i x_j) = D_{LR}(x_i, x_j) + D_{RL}(x_j, x_i) \tag{6}$$

3.2. Tree-Based Puzzle Reconstruction

Once an edge compatibility metric has been chosen and edge weights have been computed, we must still find a means of using these weights to reconstruct the puzzle. We again followed the lead of Gallagher [5] and reformulated the puzzle-solving problem as a graph problem, namely that of finding a minimum spanning tree (MST) for a graph representation of the puzzle.

Consider a graph in which each vertex corresponds to a piece in the scrambled puzzle and the edges connecting vertices have weights which correspond to the compatibility of the two connected pieces. Because each piece can be placed in four different orientations, this graph is extremely dense: in fact, there are 16 edges connecting any two distinct vertices. A minimum spanning tree algorithm, such as Kruskal's [6], can then be run on this graph to group the puzzle pieces into a single connected component.

However, in general the minimum spanning tree of the graph will not represent a valid configuration of the puzzle since it does not take into account the additional structure present in this puzzle graph. Specifically, for a given arrangement of the pieces to be valid it must satisfy certain spatial consistency constraints, in particular that each edge of each piece can be connected to at most one neighboring piece.

We therefore implemented a Disjoint Set Forest data structure with modifications to address these additional constraints. The forest begins with each vertex in its own individual tree. The edge e with the smallest weight is then extracted from a priority queue. This edge records not only which pieces are to be connected, but also which edges of those pieces are to be placed next to each other. If the two pieces indicated by e are already contained within the same forest (meaning that they are already within the same cluster of assembled puzzle pieces), then e is thrown out. Otherwise, the no-overlap condition is checked.

To determine whether merging two clusters would result in pieces colliding, each cluster representative in the disjoint set forest stores the coordinates of each piece that it represents. Then, when considering an edge, we translate and rotate the coordinates of one cluster to lie in the same reference frame as the other. If any coordinate pair appears within both clusters, including the edge in the forest would result in a conflict, so it is discarded. If not, the clusters are merged, as can be seen in the various intermediate stages of the reconstruction shown in Figure 1.

At first, the outputs from our reconstruction were far from correct. We determined the cause of these poor results to be the fact that, initially, the algorithm chose to merge uninformative pieces early. For example, in many photos the algorithm would merge all the sky pieces first, even though it is extremely difficult to correctly arrange the sky. This be-

Figure 1: Intermediate stages showing pieces being added to the main puzzle connected component.

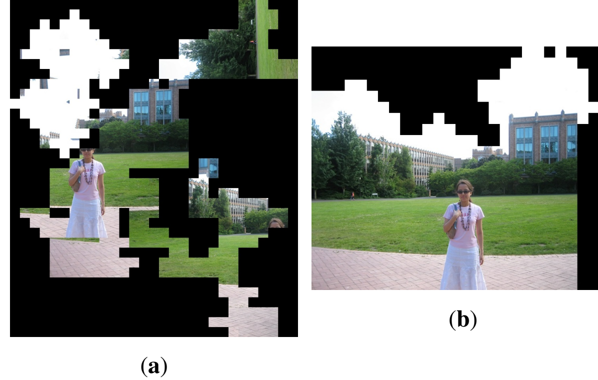


Figure 2: **(a)** The reconstruction before dividing by the second lowest edge weights **(b)** The reconstruction after dividing by the second lowest edge weights

havior has further negative effects since the merged sky imposes additional spatial constraints on the rest of the pieces, thereby preventing the rest of the puzzle from being solved correctly.

We therefore sought a means of re-prioritizing the edges such that those chosen early in the merge process were more likely to be correct. To solve this problem, we used a trick popularized by the SIFT feature descriptor [8]. The key insight is that if a piece has a single match which is significantly better than all others, that match is much more likely to be correct than if the piece had multiple matches which all had a similar score. We therefore divide each set of pairwise compatibility scores corresponding to a particular edge by the second smallest score. This ensures that any dominant matches have a score much less than 1, while any non-dominant matches are brought back up to an edge weight of 1.

The effect of this change can be seen by comparing the two sides of Figure 2. The photo labeled (a) shows a reconstruction generated using the raw edge compatibility scores, and the one labeled (b) shows the same puzzle reconstructed with edge weights which have been divided by the second best. Clearly, (b) is significantly closer to the correct reconstruction.

3.3. Trimming

A sample output of the Tree-Based Reconstruction algorithm is shown in Figure 3 (a). As can be seen, the Tree-Based Reconstruction output is often not a perfect rectangle since it has no knowledge of the true puzzle dimensions. In order to correct the minor errors present in the MST output, we trim the puzzle to ensure that it has the same dimensions as the original image. This is accomplished by moving a frame the size of the original image over the puzzle and determining the location which includes the largest number of pieces. Because we are dealing with Type 2 puzzles, which include rotations, the MST can be rotated by 90°, meaning that we have to try both possible orientations of the sliding frame.

Once the frame location has been determined, any pieces outside of the frame are trimmed from the main puzzle. The output of this stage is shown in Figure 3 (b).

3.4. Filling

The final step is to use any pieces trimmed in the previous stage to fill the holes remaining in the constructed puzzle. First, we determine all of the holes which have the highest number of filled neighbors. We then loop over each remaining piece and each hole, computing the total compatibility score of placing that piece next to each of the occupied neighbors of that hole. The piece with the minimum score is then added to the puzzle in the proper position and orientation, and the process repeats, with the newly added piece included in the neighbor computations. A sample of the results from this stage is shown in Figure 3 (c).

4. Results

In order to create the best puzzle reconstructions possible, we must first determine which of our two edge compatibility metrics is better. In this case, the accuracy of an edge metric can be measured by determining the percentage of puzzle piece edges in which the best match as reported by the metric is, in fact, the true matching edge in the puzzle.

Table 2 shows the percentage of puzzle edges whose nearest neighbor with respect to a specific metric is their true neighbor in the image. We used the MIT dataset compiled by Cho [2], which consists of 20 pictures that are commonly used as a benchmark in all the references. We also used a larger picture to see the changes when both P (piece size) and K (total number of pieces) increase. The result were consistent with our expected patterns: the MGC

Figure 3: **(a)** The puzzle before trimming **(b)** The trimmed puzzle **(c)** The filled puzzle

	MIT dataset ($P = 28$, $K = 432$)	
Type	Type 1	Type 2
RGB SSD	0.357	0.293
MGC	0.842	0.815
	Elephant picture ($P = 64$, $K = 540$)	
Type	Type 1	Type 2
RGB SSD	0.764	0.663
MGC	0.953	0.951

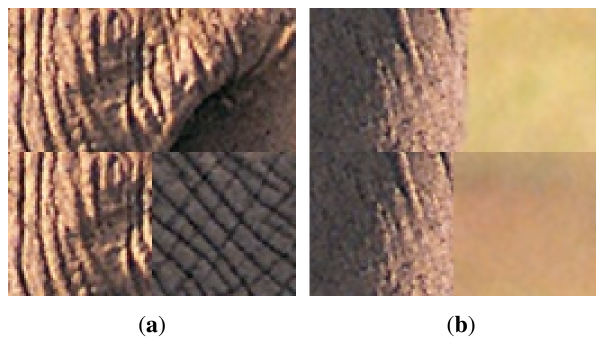
Table 2: The different metrics which were applied to the MIT dataset and a larger elephant picture, and the resulting percentage of edges which were matched with their true neighbor.

metric consistently gets a higher percentage of true matches, and Type 1 puzzles are generally easier to solve than Type 2. Moreover, both metrics do significantly better when there is more information at each edge (i.e. larger P).

Figure 4 shows the nearest neighbor for various pieces found by both the MGC and SSD metric. The two images, especially (b), demonstrate why consideration of gradient distributions is useful, as similar colors across an edge are often less important than similar gradient values. Although the edge pixels do share a common color scheme, the smooth texture in the bottom right of (b) clearly doesn't match the rough elephant skin, a feature which is detected by the MGC metric.

Once we had established the superiority of the MGC metric over SSD, we decided to only use the MGC metric in reconstructing our final results. Next, we had to determine a means of measuring the accuracy of a puzzle reconstruction. We focused on two metrics widely used in the puzzle-solving literature:

Direct Comparison Metric: The Direct metric measures the percent of pieces in the reconstructed puzzle which are in the same position and have the same orientation as the pieces in the initial, unscrambled picture. Note that this metric can report back a score of zero, even for a relatively good reconstruction, if the pieces have all been shifted over from their true location. Therefore, this metric doesn't always accurately reflect the quality of our reconstruction. This is why we also introduce a second metric.



(a) **(b)**

Figure 4: Results showing the nearest neighbors found by the MGC and SSD metrics for pieces of the elephant photograph. In each of the two images, the upper and lower left corners represent the query piece. The upper right corner is the nearest neighbor as measured by MGC, and the lower right corner is the nearest neighbor as measured by SSD.

Neighbor Comparison Metric: The Neighbor metric measures the percent of edges in the reconstruction which are adjacent to their true neighbor in the original image. This metric is more robust to translations than the Direct metric, since only those edges along the wrap-around edge will be counted as incorrect.

Table 3 shows our algorithm's results on the MIT dataset, using several different values for P and K. Comparing these numbers with those in Table 1, we can see that our result for $P = 28$ and $K = 432$ is on the same order of magnitude as those generated by state of the art algorithms.

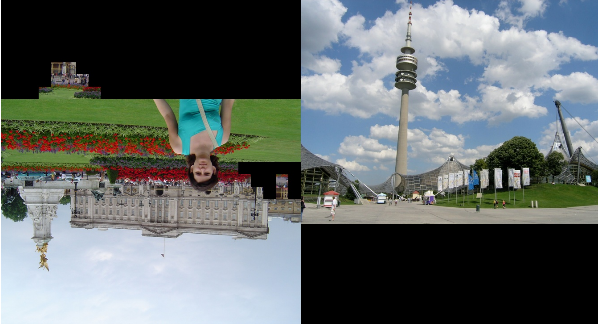


Figure 5: Reconstruction from a mixture of two different images, each consisting of 432 pieces.

Also notice that the size of the squares is a very important parameter. Just by decreasing the size of the pieces by 25%, from 28 to 20 pixels, we see that our results decrease dramatically. Therefore, we the piece size imposes significant limitations on our algorithm. A solution for this problem is to keep P constant but switch instead to higher resolution pictures in order to increase K. Additional tests showed that our algorithm is less sensitive to increasing the number of pieces K, as is still performs well even with thousands of pieces (see our best result). Also notice that a significant fraction of the dataset is reconstructed perfectly, and that it runs in reasonable time.

Our tree-based method only uses the dimensions of the initial image in the trimming and filling stages. Therefore, it is still possible to assemble a puzzle with extra pieces that don't necessarily belong to our initial image. To illustrate this point, we run our algorithm on a mixture of pieces from two different images (see Fig. 5). We are still able to reconstruct the pictures almost entirely, even though our algorithm didn't know anything (dimensions, number of pieces belonging to each image, etc.) about the two mixed pictures. Notice that we cannot run the post-processing steps on these images, since we don't each picture's separate dimension.

Finally, Figure 6 represents our largest puzzle reconstruction thus far. This puzzle has 2856 pieces, each of size 28×28 pixels, and took one hour on a Stanford Corn Machine. It is a perfect reconstruction of the original image.

5. Conclusions

In summary, this paper implements a recent algorithm for assembling jigsaw puzzles of unknown orientation and positions. First, we compared two compatibility metrics and determined that the Mahalanobis Gradient Compatibility performs better because it incorporates information about the continuity of pixel values along the common edge. Next, we used a novel Disjoint Set Forest implementation for a modified version of Kruskal's algorithm. This modified version takes into account spacial consistency, which is crucial for the puzzle reassembly problem. We then postprocess the reconstructed image to guarantee it has the same dimensions as the original. We found our algorithm to perform with high accuracy, matching the reconstructions of state of the art algorithms when run on common datasets. Finally, we demonstrate the power of our algorithm by perfectly reconstructing a 2856-piece puzzle.

One caveat of this approach is that we have no way of backtracking (except for trimming / filling). That is, the algorithm has no way of correcting a mistake which occurs early in the reconstruction process. Therefore, we found while our results typically yielded an almost perfect reconstruction, it occasionally made a mistake early on while solving a puzzle, leading to extremely poor results in isolated cases.

This problem is especially prevalent in pictures where we have a lot homogeneous textures, for example the sky. In these cases, it is easy for the algorithm to incorrectly match the nearly identical sky pieces. Unfortunately, the MIT dataset contains many pictures in which the sky has been almost completely washed out. Therefore, it is quite likely that on other datasets, our reconstruction accuracies would be even higher than they were on the MIT data.

The number of possible edge pairs in a K piece puzzle is approximately $16K^2$, meaning that our largest puzzle computed over 130 million edge weights. This edge weight computation accounts for a huge majority of the algorithm's runtime (the actual reconstruction takes around a minute for large puzzles and seconds for small). Furthermore, we found that less than the top one percent of edges were considered by Kruskal's before the pieces formed a single connected component. Therefore, if we were able to bucket the edges and only compute distances between edges likely to be neighbors, it would have a profound effect on our algorithm's runtime. Thus, in the future we hope to implement some form of Locality Sensitive Hashing in order to decrease the number of edge compatibility computations run by the algorithm.

Our code can be found at: `https://github.com/czanoci/cs231a-cris-jim`

References

[1] A. G. Castañeda, B. J. Brown, S. Rusinkiewicz, T. A. Funkhouser, and T. Weyrich. Global consistency in the automatic assembly of fragmented artefacts. In *VAST*, volume 11, pages 73–80, 2011.

[2] T. S. Cho, S. Avidan, and W. T. Freeman. A probabilistic image jigsaw puzzle solver. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 183–190. IEEE, 2010.

[3] M. G. Chung, M. M. Fleck, and D. A. Forsyth. Jigsaw puzzle solver using shape and color. In *Signal Processing Proceed-*

Square side P, number of pieces K	Direct Metric	Neighbor Metric	Perfect	Runtime per 20 images
$P = 20$, $K = 825$	52.4 %	76.2 %	5/20	221 min
$P = 28$, $K = 432$	88.5 %	92.9 %	12/20	37 min
$P = 36$, $K = 432$	95.8 %	95.5 %	15/20	12 min

Table 3: Reconstruction performance of our solver on Type 2 puzzles. The average is taken over five independent runs on the MIT dataset.

(**a**) (**b**)

Figure 6: Our largest reconstructed puzzle (2856 pieces) **(a)** The scrambled puzzle **(b)** The solved puzzle

ings, 1998. ICSP'98. 1998 Fourth International Conference on, volume 2, pages 877–880. IEEE, 1998.

[4] H. Freeman and L. Garder. Apictorial jigsaw puzzles: The computer solution of a problem in pattern recognition. *Electronic Computers, IEEE Transactions on*, (2):118–127, 1964.

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

[6] J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956.

[7] H. Liu, S. Cao, and S. Yan. Automated assembly of shredded pieces from multiple photos. *Multimedia, IEEE Transactions on*, 13(5):1154–1162, 2011.

[8] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[9] T. R. Nielsen, P. Drewsen, and K. Hansen. Solving jigsaw puzzles using image features. *Pattern Recognition Letters*, 29(14):1924–1933, 2008.

[10] D. Pomeranz, M. Shemesh, and O. Ben-Shahar. A fully automated greedy square jigsaw puzzle solver. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 9–16. IEEE, 2011.

[11] D. Sholomon, O. David, and N. Netanyahu. A genetic algorithm-based solver for very large jigsaw puzzles. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1767–1774, 2013.

[12] K. Son, J. Hays, and D. B. Cooper. Solving square jigsaw puzzles with loop constraints. In *Computer Vision–ECCV 2014*, pages 32–46. Springer, 2014.

[13] H. Wolfson, E. Schonberg, A. Kalvin, and Y. Lamdan. Solving jigsaw puzzles by computer. *Annals of Operations Research*, 12(1):51–64, 1988.

[14] R. Yu, C. Russell, and L. Agapito. Solving jigsaw puzzles with linear programming. *arXiv preprint arXiv:1511.04472*, 2015.

[15] Y.-X. Zhao, M.-C. Su, Z.-L. Chou, and J. Lee. A puzzle solver and its application in speech descrambling. In *Proc. 2007 WSEAS Int. Conf. Computer Engineering and Applications*, pages 171–176, 2007.