Solving Jigsaw Puzzles Using Image Processing

Victor Artois
vartois@ulb.ac.be

Cédric Hannotier
channoti@ulb.ac.be

Mathieu Petitjean
mpetitje@ulb.ac.be

***Abstract*—An automatic jigsaw puzzle reconstructs an image starting from scrambled, non-overlapping puzzle parts. Here, an algorithm is developed to solve puzzles composed of coloured, square pieces with known orientation. Several dissimilarity metrics from the literature are introduced, exploiting the color intensities or the gradient intensities at the boundaries of the pieces. Based on these, an iterative algorithm is used to reconstruct the puzzle. First, the pieces are placed based on the search for the lowest dissimilarity, then symmetric matches are extracted as a start for the next iteration. The parameters of the metrics are tuned on a image set, and the performances of the metrics for various piece sizes are compared. For larger pieces (84 by 84 pixels), the gradient information allows for the best accuracy (93%). For 56 by 56 pieces and smaller, the combination of pixel intensities and gradient variation is the best performing metric.**

***Keywords*—jigsaw puzzle, pixel intensity, color gradient**

I. Introduction

Automatic solving of puzzles has become more and more popular because it has applications in various fields ranging from the reassembly of archaeological relics to DNA modelling [**robust**]. It has been the subject of various studies aiming at the automatic reconstruction via image processing. While most algorithms are designed to solve puzzles composed of square, digitally scrambled pieces, some can even start from pictures of actual puzzles (for example in [**shape**]).

In this work, puzzles of coloured square pieces are considered. No prior information from the original image is used, but the pieces are assumed to be correctly oriented. For theses assumptions, solvers with more than 90% accuracy exist. Different methods of the measure of the dissimilarity between two pieces from the literature are first detailed, then the placement algorithm on the basis of this compatibility is introduced. Finally, the performance of the different methods are compared.

II. Algorithm

A. Dissimilarity metrics

The reconstruction of the puzzle is based on a dissimilarity metric between all the pieces. The higher the dissimilarity between two given pieces, the lower priority is given to the fact that these pieces are placed next to each other. Several dissimilarity metrics are introduced.

Figure 1 depicts the working principle of the dissimilarity computation. The intensity of the red channel for the right side of a piece of the puzzle is plotted along with the left side of two others pieces. One can see that the matching piece has nearly the same profile as the starting piece (nearly superimposed), while the third one is very different. Of course, the difference between matching and non matching pieces are not always that obvious.

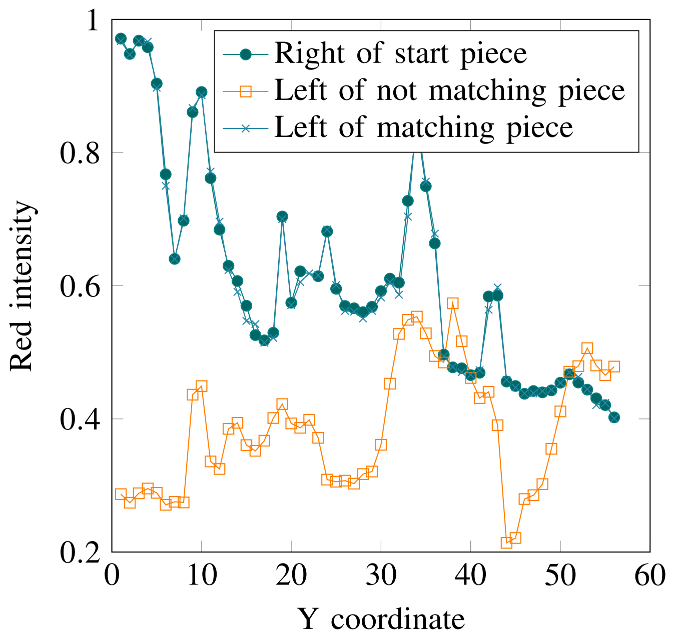


Figure 1. Illustration of the principle of dissimilarity computation by showing the red channel intensities for three pieces.

a) Sum of Squared Distances (SSD):

The SSD between two pieces x_i and x_j is obtained by summing up the squared differences of the pixels along the pieces' juxtaposing boundaries, for each color channel (red, green and blue) [**robust**]. For a piece x of size $M \times M$, $x(m, n, c)$ denotes the value of the pixel at the coordinates (m, n) in the color channel c. The SSD between x_i and x_j, x_i being placed at the left of x_j, can be expressed as

$$D_{LR,SSD}(x_i, x_j) = \sum_{c=1}^{3}\sum_{k=1}^{M}\left(x_i(k, M, c) - x_j(k, M, c)\right)^2 \tag{1}$$

This can be extended to the cases where x_i is placed on the right, on the top or on the bottom of x_j.

b) $(L_p)^q$ norm:

The SSD is actually the L_2 norm of the two vectors that form the boundaries of the two pieces. The $(L_p)^q$ can also be used, and it is defind as

$$D_{LR,pq}(x_i, x_j) = \left(\sum_{c=1}^{3}\sum_{k=1}^{M}\left(\left|x_i(k, M, c) - x_j(k, M, c)\right|\right)^p\right)^{\frac{p}{q}} \tag{2}$$

Hence, the SSD is a particular case of $(L_p)^q$ norm with $p = q = 2$. It was experimentally found in [**greedy**] that the good results were obtained for $p = 3/10$ and $q = 1/16$, they will thus be used in the solver. Instinctively, it is due to the fact that the L_2 norm penalizes a lot large boundary differences even though such differences may occur in natural images.

c) Mahalanobis Gradient Compatibility (MGC): While the first two metrics penalize differences in the pixels intensities, the MGC (introduced in [**Gallagher**]) instead penalizes differences in the intensity of the gradient. If a piece has a strong gradient near its edge, it is expected that the juxtaposed piece will continue the gradient. The penalty for a deviation is computed with the Mahalanobis distance.

Looking at the dissimilarity between the right side of x_i and the left side of x_j, let $\mu_{iL}(c)$ be the average color difference in the color channel c between the two last columns of x_i:

$$\mu_{iL}(c) = \frac{1}{M}\sum_{k=1}^{M} x_i(k, M, c) - x_i(k, M-1, c) \tag{3}$$

The array of gradients of dimension M by 3 G_{ijLR} is then introduced, with $G_{ijLR}(k, c)$ denoting the color difference between the right side of x_i and the left side of x_j for the color channel c at the row k:

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

By denoting S_{iL} the 3 by 3 covariance matrix obtained from the gradient difference at the right of x_i, the gradient dissimilarity is given by:

$$D'_{LR}(x_i, x_j) = \sum_{k=1}^{M} D'_{LR,k} \tag{5}$$

$$D'_{LR,k} = \left(G_{ijLR}(k) - \mu_{iL}\right) S_{iL}^{-1} \left(G_{ijLR}(k) - \mu_{iL}\right)^T \tag{6}$$

The final MGC symmetric dissimilarity metric for placing the piece x_i at the left of x_j is given by:

$$D_{LR,MGC}(x_i, x_j) = D'_{LR}(x_i, x_j) + D'_{RL}(x_j, x_i) \tag{7}$$

d) Combining MGC and SSD (M+S):

It was proposed in [**robust**] that the SSD and the MGC convey complementary information so that they could be combined to provide more accuracy. The M+S dissimilarity metric is hence defined as the weighed product of the SSD and the MGC:

$$D_{M+S}(x_i, x_j) = D_{MGC}(x_i, x_j) \times \left(D_{SSD}(x_i, x_j)\right)^{\frac{1}{r}} \tag{8}$$

The value of the weighing parameter r giving the best results is investigated later.

e) Combining MGC and $(L_p)^q$ (M+pq):

In the same idea, the MGC and the $(L_p)^q$ can be combined in a weighed product. This method will be referred to as the M+pq in the latter.

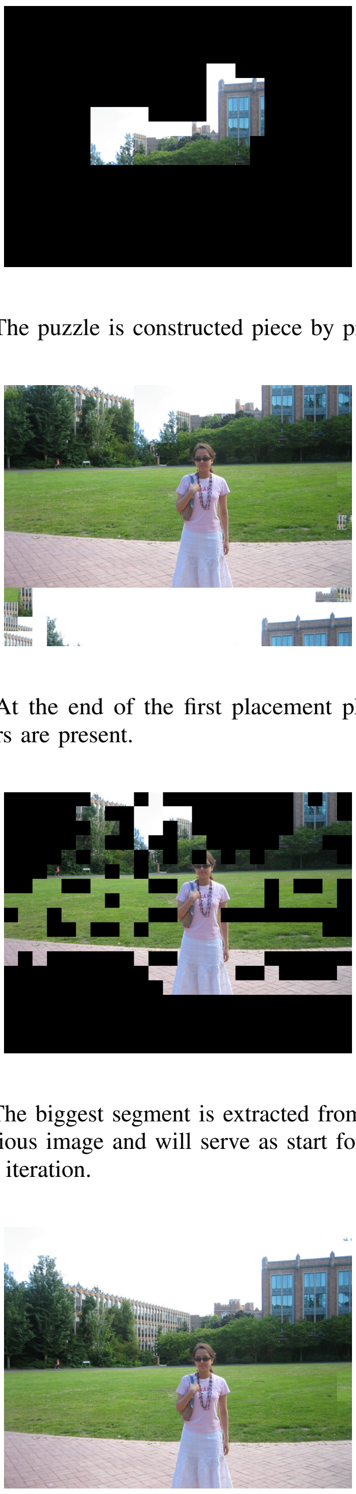
$$D_{M+pq}(x_i, x_j) = D_{MGC}(x_i, x_j) \times \left(D_{pq}(x_i, x_j)\right)^{\frac{1}{r}} \quad (9)$$

B. Placement algorithm

Once the dissimilarity is evaluated between every piece of the puzzle, for every orientation (top, bottom, right, left), the pieces should be placed accordingly to these measures. The implemented algorithm is a two step iterative method, and it is inspired from what was proposed in [**greedy**]. An illustration of its functioning is depicted in Figure 2.

The starting piece is chosen to be the one having the lowest dissimilarity with any other piece of the puzzle. During the first step, all the pieces that have already been placed are scanned. The next piece that will be placed on the puzzle is the one that has the lowest dissimilarity with any of the already placed ones. Piece by piece, the puzzle is then reconstructed. When a piece has to be placed in between multiple others, the dissimilarity with all of these is taken into account.

The second step starts when the puzzle is full. The algorithm then looks for *symmetric matches* in all the juxtaposed pieces. Two pieces x_i and x_j are symmetrically matched if the best possible match for x_i in every possible orientation is x_j and the best possible match for x_j is x_i in the opposite orientation. For example, if the pieces labeled x_3 and x_7 are juxtaposed at the end of the first step, they will be considered as symmetric matches if the best possible match for x_3 is x_7 on its right, and the best match for x_7 is x_3 on its left. From these symmetric matches, a *segment* is extracted from the image. It is defined as the biggest portion of the puzzle only composed of symmetrically matched pieces, and will serve as starting point for the next iteration. The iteration process stops when the result after the reconstruction step is not changing anymore.


(a) The puzzle is constructed piece by piece.

(b) At the end of the first placement phase, errors are present.

(c) The biggest segment is extracted from the previous image and will serve as start for the next iteration.

(d) The reconstruction is nearly perfect after two iterations.

Figure 2. Illustration of the reconstruction algorithm

The Matlab implementation of the algorithm was optimized using matrix computations, and the full solving of a 432 parts puzzle takes less than 10 seconds.

III. Results

A. Metric parameters

The parameters of the dissimilarity metrics are varied to determine which values give the best results. The methods were tested with different parameters on the set of images[1] first used by Cho et al. [**cho**] and then in [**robust**] and [**greedy**] for puzzle pieces of size 28 by 28 pixels. The number of pieces of the puzzle in then of 432. The error is expressed as the percentage of badly positioned pieces for all the 20 images of the set.

a) M+S:

The parameter r from Equation 8 is varied from 1 to 20. The accuracy for each value is shown in Figure 3.

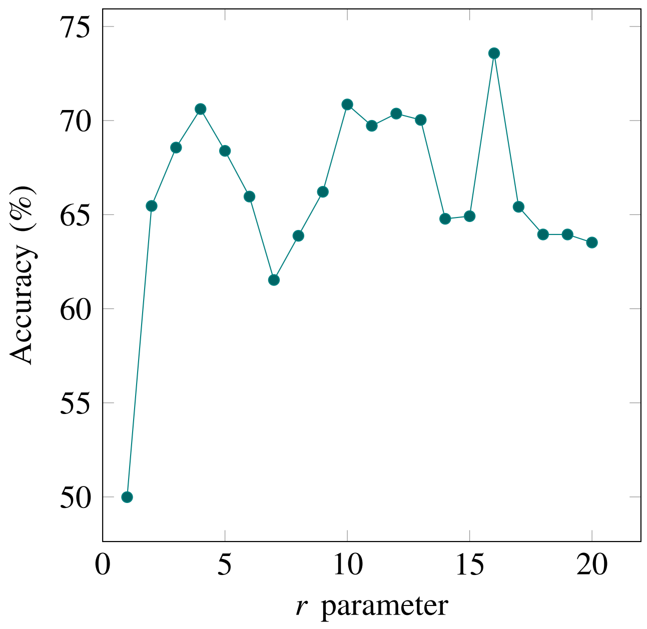


Figure 3. Performance of the M+S for various values of the r parameter

The accuracy is maximum for $r = 16$, hence this value will be retained for the final solver. This is also the value of r that yielded the best results in [**robust**], which confirms this experimental result.

b) M+pq:

The parameter r from Equation 9 is varied. The accuracy for each value is shown in Figure 4.

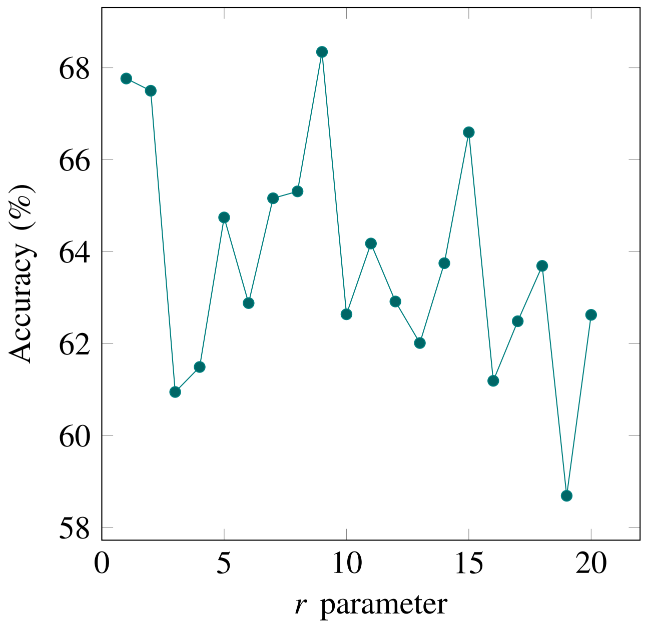


Figure 4. Performance of the M+pq for various values of the r parameter

When the $(L_p)^q$ is combined with the MGC, the best results are obtained when $r = 9$, which will be retained. This M+pq combination has not been used in the literature, so that this conclusion only relies on the present experiments.

For both the M+S and the M+pq, the performance appears to be better when the contribution from the MGC is more significant than the contribution of the methods based on the pixel intensities.

B. Performance comparison

The metrics with optimal parameters are then used again on the same set of images. Their performance is compared for piece sizes of 84 by 84 pixels, 56 by 56 pixels and 28 by 28 pixels.

[1]available here: http://people.csail.mit.edu/taegsang/Documents/jigsawCode.zip

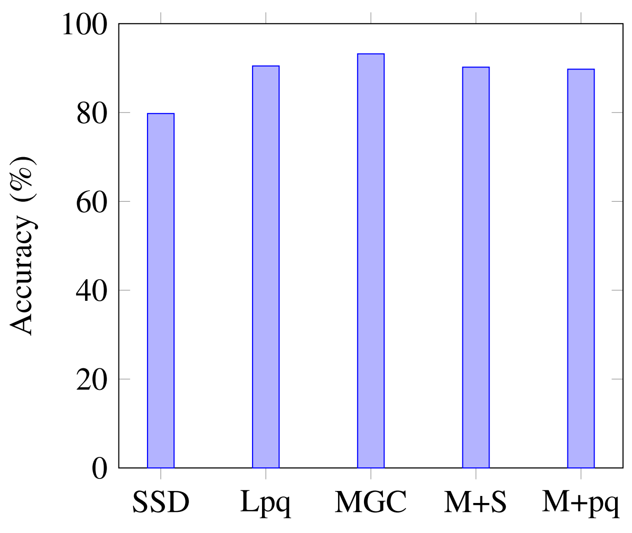


Figure 5. Performance comparison for bloc sizes of 84 pixels. The MGC is outperforming all others with an accuracy of 93.2%.

The results for pieces of 84 by 84 pixels are shown in Figure 5. When the size of the piece has a relatively high value, the solver is very accurate. As expected, the $(L_p)^q$ metric is performing better than the SSD as it has a very similar working principle but penalizes less the natural steep edges at the boundaries of the pieces. Surprisingly, the MGC is outperforming the more advanced M+S and M+pq metrics. The additional information brought by the pixel intensities degrades the information brought by the color gradients.

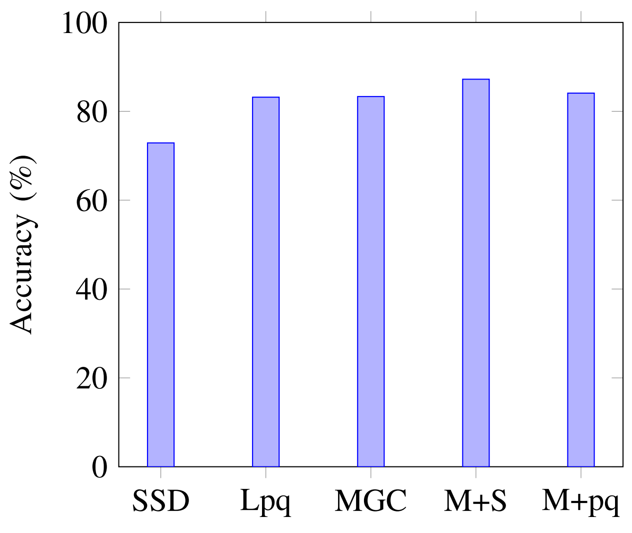


Figure 6. Performance comparison for bloc sizes of 56 pixels. The M+S is outperforming all others with an accuracy of 87.2%.

It is worth noting that a mean accuracy of 90% does not indicate in this case that each puzzle is reconstructed with 10% of errors. Actually, most images are reconstructed very well, but a few of them are not accurately at all, dropping the mean.

The tendencies observed above are not true anymore when the size of the puzzle pieces goes down. In Figure 6 are shown the results for 54 by 54 pixels pieces, where it appears that the M+S is the most accurate method. When the border is smaller, less information is available and the combination of the color and gradient information allows to get more accuracy. An unexpected observation is that even if the SSD alone is far from being as accurate as the $(L_p)^q$, the combination of the SSD and the MGC outperforms the combination of the $(L_p)^q$ and the MGC.

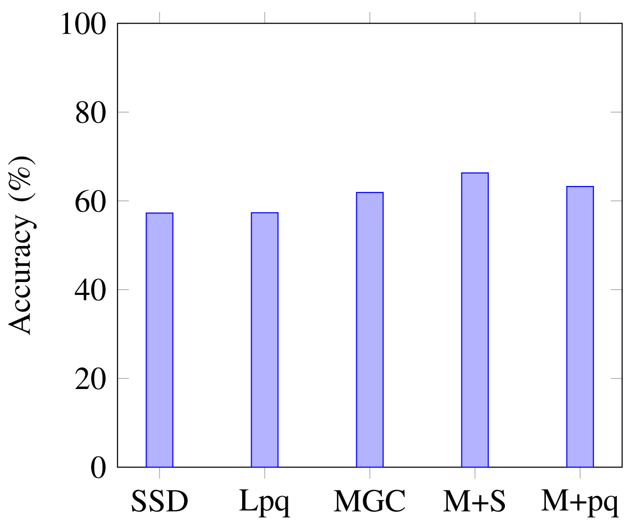


Figure 7. Performance comparison for bloc sizes of 28 pixels. The M+S is outperforming all others with an accuracy of 66.3%.

Lastly, the solver was tested on small pieces of 28 by 28 pixels. The side of the pieces being very small, there is little information available and the accuracy drops to 66%. Still, the M+S metric is the one outperforming the others, and the conclusions from the 56 by 56 case hold.

IV. Conclusion

Based on the quite active research on the subject of automatic puzzle solving, several dissimilarity metrics have been introduced. The SSD and the $(L_p)^q$) are based on the color intensities at the boundaries of the pieces and the MGC is based on the gradient of these intensities. The two types of dissimilarity estimations can be combined into the M+S and the M+pq.

A placement algorithm has been developed, whose objective was to reconstruct the puzzle based on the dissimilarity between all the pieces of the puzzles in all the possible configurations. To lower the number of errors due to the placement, a segment of symmetric matches is extracted at the end of the placement and will serve as a start for the next iteration.

A set of images widely used in the literature has been used first to tune the parameters of the metrics, and next to quantify the performance of the puzzle solver. An accuracy of 93% was reached on puzzles composed of pieces of 84 by 84 pixels using the MGC method. For lower sizes, the M+S method is the best one, and the accuracy to 87% for 56 by 56 and to 66% for 28 by 28.

The code is implemented in Matlab, and C/MEX functions are available to fasten the execution.

Performance could be improved at both phases of the reassembly. Firstly, the dissimilarity metrics could be improved to use more information than the content of the border only. Prediction-based or probabilistic metrics also exist. Secondly, the placement algorithms used in the literature are a bit more involved.

The jigsaw puzzle problem stays an open problem as no one has yet came up with a solver able to reconstruct any puzzle with 100% accuracy.

APPENDIX

The work separation was done as follows:

- Mathieu Petitjean did the research of the existing techniques for computing the dissimilarity between pieces and implemented them in Matlab.
- Cédric Hannotier implemented the iterative placement algorithm on the basis of theses dissimilarity measures. He also optimized the Matlab code to reach fast execution time.
- Victor Artois ran the expirements to determine the optimal values of the parameters and compared the performance of the prosposed metrics for various puzzle sizes.