Solving Multiple Square Jigsaw Puzzles with Missing Pieces

Genady Paikin
Technion
genadyp28@gmail.com

Ayellet Tal
Technion
ayellet@ee.technion.ac.il

Abstract

Jigsaw-puzzle solving is necessary in many applications, including biology, archaeology, and every-day life. In this paper we consider the square jigsaw puzzle problem, where the goal is to reconstruct the image from a set of non-overlapping, unordered, square puzzle parts. Our key contribution is a fast, fully-automatic, and general solver, which assumes no prior knowledge about the original image. It is general in the sense that it can handle puzzles of unknown size, with pieces of unknown orientation, and even puzzles with missing pieces. Moreover, it can handle all the above, given pieces from multiple puzzles. Through an extensive evaluation we show that our approach outperforms state-of-the-art methods on commonly-used datasets.

1. Introduction

Puzzle solving is important in many applications, such as image editing [4], biology [15] and archaeology [2, 10, 3], to name a few. Though the problem is NP-complete [7, 1], various solutions have been proposed. These are based on shape matching [21, 20, 11, 9] or on a combination of shape matching and color matching [12, 6, 23, 14, 18, 16]. When the parts are square, only color matching is possible [5, 22, 17, 8, 19, 13]. The latter is the focus of our work.

The problem is introduced by [5], where a greedy algorithm, as well as a benchmark, are proposed. The algorithm discussed in [22] improves the results by using a particle filter. Pomeranz et al. [17] introduce the first fully-automatic square jigsaw puzzle solver that is based on a greedy placer and on a novel prediction-based dissimilarity. Gallagher [8] generalizes the method to handle parts of unknown orientation. Son et al. [13] demonstrate a considerable improvement for the case of unknown orientation, by adding "loop constraints" to [8]. Rather than pursuing a greedy solver, Sholomon et al. [19] present a genetic algorithm that is able to solve large puzzles.

Our work is inspired by [17]. However, it takes the next step and not only improves upon state-of-the-art results, but also solves puzzles with additional challenges. In particular, it handles puzzles with missing pieces, unknown size, and unknown orientation of the parts (Figure 1). Moreover, it concurrently solves multiple jigsaw puzzles whose pieces are mixed together, where neither the size of the puzzles nor a priori knowledge regarding possible missing pieces is given. This is illustrated in Figure 2, where more than 5000 pieces that belong to five different puzzles, some with many missing pieces, are given, and our algorithm reconstructs them faultlessly.

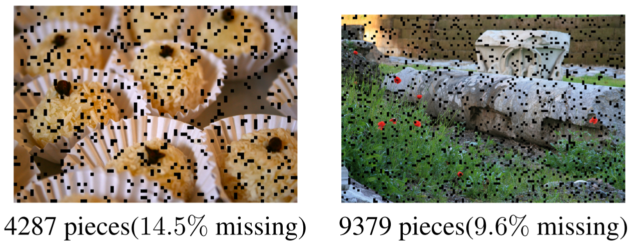
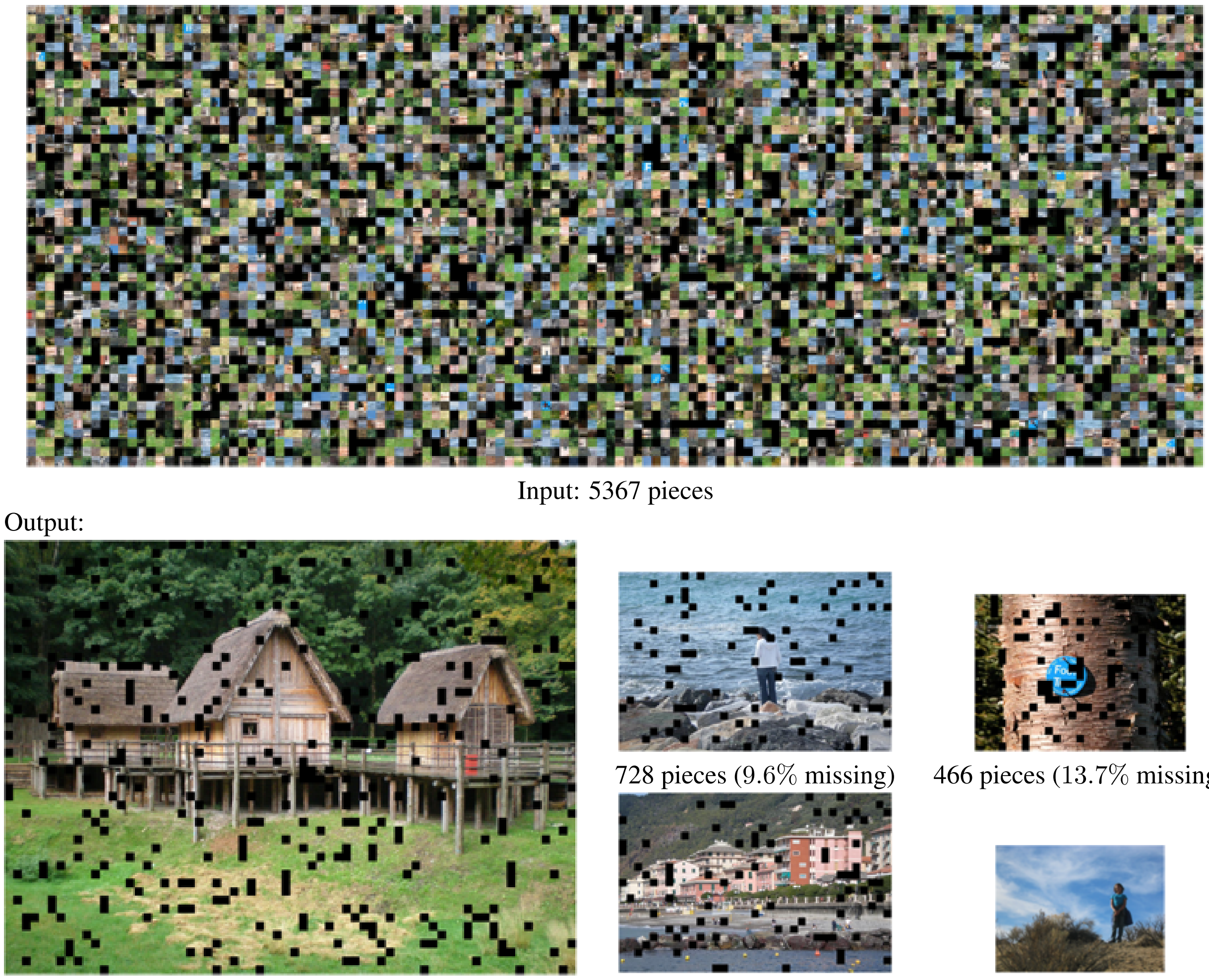


Figure 1. Accurate reconstructions of our solver, given jigsaw puzzles with missing pieces (black squares).

Like [8, 17], our algorithm is greedy. However, it incorporates three key ideas, which not only prove beneficial for solving traditional puzzles, but also support the additional requirements mentioned above. First, similarly to previous methods, our placement is based on the compatibility between pieces. We propose a more accurate and faster compatibility function, which takes advantage both of the similarity between the pieces, and of the reliability of this similarity. Second, since greedy solvers are extremely vulnerable to the initial placement, we take special care when choosing the first piece. We require it to have distinctive borders and to be located in a distinctive region. Third, rather than choosing the best piece for a specific location, we select the piece that minimizes the likelihood of erring, regardless of its location. As a result, our algorithm is deterministic, conversely to previous approaches, which make random choices and are thus sensitive either to the selection of the first piece or to an initial random solution. This means that it suffices to run our algorithm only once.


Input: 5367 pieces
Output:
728 pieces (9.6% missing)
466 pieces (13.7% missing)

3018 pieces (8.5% missing)
723 pieces (10.2% missing)
432 pieces (0% missing)

Figure 2. **Solving multiple puzzles with mixed and missing pieces.** The input contains parts from five puzzles. Our algorithm has no knowledge regarding the size of the puzzles, the number of pieces per puzzle, or the number of missing pieces. Nonetheless, it reconstructs the puzzles accurately in 30 seconds.

Our method outperforms the state-of-the-art methods, both in accuracy and in efficiency. For instance, it achieves accuracy (to be later defined) of 97.7% on the dataset of [19], which consists of very large jigsaw puzzles, whereas [19] achieves 96.41%. This is done while accelerating the running times by a factor of 60-120.

Our contribution is hence a novel and efficient puzzle solver, which produces good results even when the input consists of pieces from multiple puzzles, each has missing pieces and unknown size. This may be the case not only in everyday situations, but also in archaeology, when torn documents or broken artifacts are found mixed and lack parts.

2. Algorithm Outline

Our goal is to reconstruct images, given a set of non-overlapping, unorganized, square pieces. We require that the algorithm solves this problem even when:

1. The size of the puzzles is unknown.
2. The orientation of the pieces is unknown.
3. There may be missing pieces.
4. The input may contain pieces from several puzzles.

Our algorithm pursues a greedy strategy. As such, early errors may lead to later failures. To avoid this case, we base our algorithm on three principles: First, only highly reliable matches should be utilized. This means not only that the compatibility function must be good, but also that smooth areas should not be handled at early stages. Second, the first piece to use should be chosen thoughtfully, in order not to diverge from the solution right away. Finally, rather than choosing the best piece for a specific location, as often done, we prefer to select the most distinctive piece, one that minimizes the likelihood of erring, regardless of its location.

Our algorithm proceeds in three steps: calculating the compatibility between the pieces, finding the best piece to start with, and assembly. The first step (Section 3) evaluates the compatibility between every pair of pieces. We note that the dissimilarity between the parts is not sufficiently informative. Hence, we also use the reliability of a match. This leaves smooth area to the end of the assembly and assists in handling puzzles with missing pieces.

The second step (Section 4) finds the best piece to start with. The key idea is to select a piece that is not only distinctive, but also belongs to a distinctive region. This leads to high confidence in the first assemblies.

The last step places the pieces (Section 5). This is done by iteratively selecting a piece to be assembled, utilizing the compatibility function from Step 1 and the concept of "best buddy" [17]. A guiding principle is that the absolute location of a piece is determined only when all the pieces found their neighbors. This is very similar to the way people solve puzzles, constructing different portions of the puzzles separately, before composing them together. This not only lets us solve puzzles of unknown size and unknown orientation, but interestingly, also handles puzzles with missing pieces.

The only input of the algorithm is the number of puzzles that need to be solved out of the mixed pieces. During the assembly, when all the pieces have low compatibility values, the algorithm starts a new puzzle (up to the given number of puzzles), by selecting a new initial piece.

3. The Compatibility Metric

A good compatibility metric is a fundamental component of every jigsaw puzzle solver. Given two pieces p_i, p_j, the compatibility function $C(p_i, p_j, r)$ predicts the likelihood that p_i and p_j are neighbors in the spatial relation r, $r \in \{up, down, left, right\}$. We compute the compatibility function in two steps. First, a dissimilarity between every pair of pieces is calculated. Then, we compute the confidence in this dissimilarity.

3.1. The Dissimilarity Between Pieces

Various dissimilarity measures have been proposed in the literature, including L_2 [5, 19] and MGC (Mahalanobis gradient compatibility) [8]. Our work is inspired by that of Pomeranz *et al.* [17], which introduces a prediction-based dissimilarity. We modify it, as explained below, to make it not only simpler and faster to compute, but also perform better, as demonstrated in Figure 3.

Briefly, in [17] backward-difference estimation is employed. It uses the last two pixels in each row (column) near the boundary, from which prediction of the first pixel in the adjacent piece is obtained. The dissimilarity measure between pixels in two pieces uses the L_p^q norm, as follows,

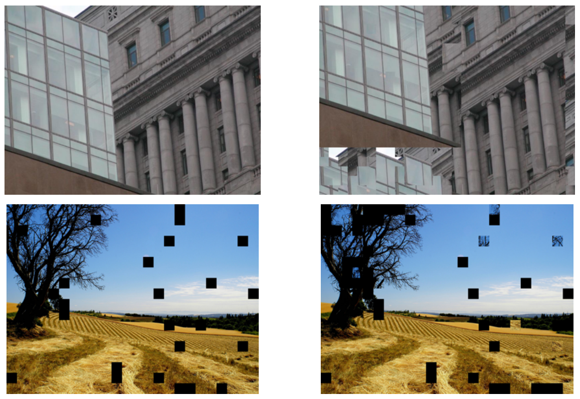

(a) Our result (b) Using [17]'s dissimilarity

Figure 3. Our dissimilarity results in a perfect reconstruction. When changing the dissimilarity to [17]'s, our algorithm errs.

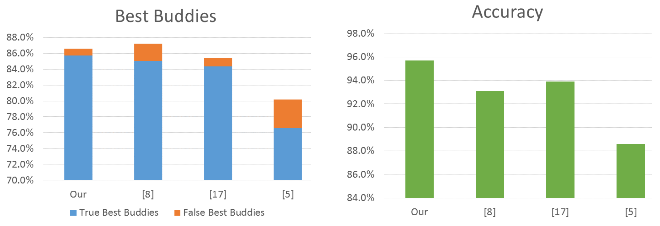


Figure 4. Our dissimilarity is better than [8, 17, 5] in terms of the correct (& incorrect) best buddies percentage and of accuracy.

where K is the piece size and d is the dimension in the LAB color space:

$$D(p_i, p_j, right) = \sum_{k=1}^{K} \sum_{d=1}^{3} \Big[\big([2p_i(k,K,d) - p_i(k,K-1,d)] - p_j(k,1,d)\big)^p + \big([2p_j(k,1,d) - p_j(k,2,d)] - p_i(k,K,d)\big)^p\Big]^{q/p}. \quad (1)$$

In practice, $p = 3/10$ and $q = 1/16$ were found to give the best results.

We modify this in two manners. First, rather than using the $L_{3/10}^{1/16}$ norm, we use the L_1 norm. This clearly accelerates the computation. In addition, we found it to improve the results, as demonstrated in Figures 3–4. This can be explained by the fact that when using L_2, even a small difference between pieces will cause them to be considered dissimilar. On the other hand, when using $L_{3/10}^{1/16}$, despite large differences, the pieces may still be considered similar. L_1 is a compromise between the two.

Second, we use an asymmetric dissimilarity (that is, $D(p\_i, p\_j, right) \neq D(p\_j, p\_i, left)$). We empirically found it to be beneficial for calculating the compatibility function that is performed next (Section 3.2). This is especially important when the puzzles have missing pieces, yet the results improve regardless. Thus, our dissimilarity is:

$$D(p_i, p_j, right) = \sum_{k=1}^{K} \sum_{d=1}^{3} \|([2p_i(k, K, d) - p_i(k, K-1, d)] - p_j(k, 1, d))\| \quad (2)$$

Figure 4 compares the results using our dissimilarity measure to those obtained using the dissimilarities of [8, 17, 5], when run on the dataset of [17] (66 images). The number of correct *best buddies,* as well as the accuracy of the algorithm, improve, while the number of incorrect best buddies decrease. We define these terms later, yet intuitively best buddies are pieces that agree that they are the most compatible neighbors of each other, and accuracy is measured by the commonly-used neighbor count, as explained in Section 6. It should be mentioned that using our dissimilarity measure, 37 puzzles are reconstructed perfectly, compared to only 25 when using [17]'s. Finally, our dissimilarity computation is three times faster than [17]. This is important since the dissimilarity calculation accounts for about 80% of the running time of our algorithm.

Acceleration: We found the following modification to be very effective. The goal is to roughly estimate the likelihood of two pieces to be adjacent, before calculating the full dissimilarity. We calculate the average color (in LAB color space) for each boundary of every piece in advance. Then, only if the difference of the average color of two boundaries is sufficiently small in L_2 (16 in practice), the exact dissimilarity between these pieces is calculated. Otherwise, the dissimilarity is set to a maximal value. This accelerates the algorithm by a factor of 2, while hardly sacrificing the quality of the results.

3.2. The Compatibility Function

The goal is to evaluate the reliability that a small dissimilarity between two pieces indeed indicates adjacency in the puzzle. Intuitively, in smooth areas every piece has a small dissimilarity to every other piece in the region. Conversely, in "interesting" regions, there are very few pairs that have a small dissimilarity value. Hence, having a small dissimilarity value by itself does not tell the full story. Therefore, we seek out pieces that have unique dissimilarities.

To realize this, given a piece, if its closest neighbor and its second closest neighbor have similar values, we do not conclude that they are necessarily neighbors. However, if the dissimilarity to the closest neighbor and that to second closest neighbor differ, the closest pieces are more likely to be neighbors. Let $secondD(p_i, r)$ be the value of the second best dissimilarity of piece p_i to all other pieces with relation r. We define the compatibility function as:

$$C(p_i, p_j, r) = 1 - \frac{D(p_i, p_j, r)}{secondD(p_i, r)}, \quad (3)$$

The higher the value of the compatibility function C, the more reliable the match is. We also experimented with other options (e.g., the quartile neighbor, the median) and found neither to be as indicative as the second best neighbor.

The "Best Buddies" metric: This metric was first introduced by [17]. Intuitively, two pieces are best buddies if both agree that the other piece is their most likely neighbor in a certain spatial relation. Two pieces p_i, p_j with relation r_1 and opposite relation r_2 are best buddies if both hold:

$$\forall p_k \neq p_j, C(p_i, p_j, r_1) \geq C(p_i, p_k, r_1)$$
$$\forall p_k \neq p_j, C(p_j, p_i, r_2) \geq C(p_j, p_k, r_2). \quad (4)$$

In practice, it is very rare that two pieces are best buddies when they are not real neighbors. It is still rare even if the input lacks many pieces, as it means that both pieces choose each other as most similar.

4. Finding the First Piece

Selecting a good starting point is crucial in greedy algorithms, as early errors lead to later failures. Nevertheless, in [17, 19], a random piece is selected as the first one. These algorithms are then run several times and present multiple solutions. Conversely, we take special care in selecting the first piece.

We look for a piece that is both distinctive and lies in a distinctive region. The question is how to define distinctiveness. Recall that our compatibility function reflects how much a border of a piece is distinctive. The best buddies condition also reflects how much a border is mutually distinctive. Moreover, best buddies of distinctive pieces are usually correct. Therefore, we use the best buddies measure to determine the first piece.

We look for a piece that has best buddies in all four spatial relations, which indicates that it is a distinctive piece. Moreover, we require that the four neighbors of this piece have best buddies in all four spatial relations as well, i.e., the piece lies in a distinctive region.

Out of all the pieces that satisfy the above conditions, we should choose one. We wish to select a piece that has the "strongest" best buddies in all four direction. To do it, we define *mutual compatibility* as follows (recall that our dissimilarity function is asymmetric):

$$\widetilde{C}(p_i, p_j, r_1) = \widetilde{C}(p_j, p_i, r_2) = \frac{C(p_i, p_j, r_1) + C(p_j, p_i, r_2)}{2}, \quad (5)$$

where the relation r_2 is the opposite of relation r_1. Now we select as the first piece as the one that maximizes the sum of the values of the mutual compatibility function (Equation 5) of its four spatial relations.

5. Greedy Placement

Our algorithm's final step assembles the pieces. This is done without prior knowledge regarding missing pieces or the size of the puzzle. Our placer (Algorithm 1) is greedy and relies on our mutual compatibility function $\widetilde{C}$ (Equation 5). It maintains a pool of candidate pieces to be placed. It iterates on placing a piece from this pool and then adding candidates to it. We elaborate below.

Algorithm 1 Placer

```
1: While there are unplaced pieces
2:     if the pool is not empty
3:         Extract the best candidate from the pool
4:     else
5:         Recalculate the compatibility function
6:         Find the best neighbors (not best buddies)
7:     Place the above best piece
8:     Add the best buddies of this piece to the pool
```

The selected piece is the one having the highest mutual compatibility function among the pieces in the pool. If the candidates' pool is empty, but the jigsaw puzzle is not yet solved, the compatibility function is recalculated. This is performed by considering only the pieces that have not yet been placed. Then, the next piece to be placed is selected according to the mutual compatibility function, regardless of whether it is a best buddy of an already-placed piece. It is placed, as before, next to an already-placed piece. Then, the best buddies (at the various directions) of the newly-place piece are added to the pool.

The placement is performed without determining the exact location. That is to say, a piece is placed in a location that is relative to other pieces and not in a specific absolute location. However, if the algorithm gets as input the puzzle's dimensions, the placement is limited to the given height and width.

The algorithm performs well even if the puzzle lacks pieces. This is because the algorithm never searches for the best piece for a specific place, as previously done. Thus, holes need not be filled. Moreover, only highly reliable neighbors are selected—those having distinctive borders. This makes the likelihood that a chosen piece is not a real neighbor low.

Similarly, when the algorithm is required to solve mixed puzzles, the only adjustment needed is the following. When all the placed pieces have a relatively small value of mutual compatibility to all the unplaced pieces (0.5 in our experiments), a new puzzle is started. In this case, the candidates' pool is cleared, and a new first piece is selected. This process repeats until the given number of puzzles is reached. The placement proceeds simultaneity on all the partial solutions, until there are no remaining pieces.

6. Results

Our puzzle solver is applied to the datasets of [5, 17, 19]. These contain three sets, each of 20 images, with 432 [5], 540 and 805 pieces [17] respectively, two sets of medium-size puzzles, each containing 3 images with 2360 and 3300 pieces [17], and three sets of large puzzles, each containing 20 images with 5015, 10375 and 22834 pieces [19] respectively. We first quantitatively compare our results to those of previous works when applied to the traditional jigsaw puzzle problem, where the size of puzzle is known and there are no missing pieces. Then we test our algorithm qualitatively (as there is no previous work) on additional cases, including puzzles of unknown size, puzzles with missing pieces, and mixed puzzles.

Evaluation when applied to the classical problem: Table 1 reports the results, averaged per set, using the three common measures. The average *neighbor measure* [5] is the standard measure, which considers the fraction of the correct pairwise adjacencies. It shows the average result on [5, 17, 19] datasets and compares the results to those of the state-of-the-art algorithms of [13, 17, 19]. It can be seen that our algorithm is competitive with [13, 19] and outperforms [17]. However, it should be noted that while our result is deterministic, [17, 19] run their algorithms 10 times and report the results, in particular the best one.

The *direct measure* [5] considers the fraction of the pieces in the assembled puzzle that are in their correct absolute position. Generally, this measure is considered to be less accurate and less meaningful due to its inability to cope with slightly shifted puzzle. Our algorithm performs better than [17, 19] and is comparable to [13].

Finally, the *perfect* columns indicate the number of puzzles for which the algorithms produced perfect reconstructions of the puzzle. Our algorithm has a clear advantage in this regard.

Table 2 reports the results when applied to the dataset of [19] that contains large puzzles. As only the algorithm of [19] was applied to this dataset, we compare only to it. It can be seen that our algorithm is beneficial for large puzzles when considering each of the measures.

Our algorithm is advantageous also in terms of running times. Table 3 compares our running times to those reported in [19], which was applied to large puzzles. Our algorithm is considerably faster for all puzzle sizes.

As another comparison, it is reported in [13] that the algorithm solves a 9801-piece jigsaw puzzle with unknown orientation in 25.6 hours. Our algorithm solves a 10375-pieces puzzle with unknown orientation in 2 minutes.

Our algorithm was implemented in Java and ran on an Intel i7 processor with 32GB RAM (all recent work ran on similar PCs). It should be noted that as our algorithm need not be adapted in order to solve puzzles with missing pieces,

	neighbor				direct				perfect			
# of pieces	Our	[13]	[19]	[17]	Our	[13]	[19]	[17]	Our	[13]	[19]	[17]
432	95.82%	95.5%	96.16%	94.25%	96.16%	95.6%	86.19%	90.95%	13	13	9	13
540	96.1%	95.2%	95.96%	90.9%	93.22%	92.2%	92.75%	83.45%	13	-	8	9
805	95.09%	94.9%	96.26%	89.7%	92.47%	93.1%	94.67%	80.25%	9	-	10	7
2360	96.26%	96.4%	88.86%	84.67%	94.01%	94.4%	85.73%	33.4%	1	-	1	1
3300	95.29%	96.4%	92.76%	85%	90.69%	92%	89.92%	80.67%	1	-	1	1
Overall	95.68%	95.31%	95.64%	91%	93.8%	93.59%	90.9%	82.35%	37	-	29	31

Table 1. **Comparisons of our results to the state-of-the-art.** Our algorithm outperforms state-of-the-art algorithms in all the commonly-used measures. ('-' mean that the results are not reported.) The comparison is made to the best-out-of-10 result of [17, 19]. When considering the worst or the average case, their results are slightly worse.

	neighbor			direct			perfect		
		best	worst		best	worst		best	worst
# of pieces	Our	[19]		Our	[19]		Our	[19]	
5015	96.43%	95.25%	94.87%	95.79%	94.78%	90.76%	12	11	7
10375	98.94%	98.47%	98.2	98.63%	97.69%	96.08%	8	6	5
22755	97.74%	96.28%	96.17	96.78%	92.02%	91.74%	3	4	4
Overall	97.7%	96.67%	96.41	97.07%	94.83%	92.77%	23	21	16

Table 2. **Comparisons of our results to the state-of-the-art.** Our algorithm outperforms the best solutions of [19] considering the common measures on the dataset of [19], which contains large jigsaw puzzles. The other methods do not provide results on this dataset.

# of pieces	# of images	[19]	Our	Ratio
432	20	48.73 [sec]	0.59 [sec]	82.59
540	20	64.06 [sec]	0.92 [sec]	69.63
805	20	116.18 [sec]	1.52 [sec]	76.43
2360	3	17.6 [min]	8.73 [sec]	120.96
3300	3	30.24 [min]	14.99 [sec]	121.04
5015	20	61.06 [min]	28.77 [sec]	127.34
10375	20	3.21 [hr]	2.06 [min]	93.5
22755	20	13.19 [hr]	14.75 [min]	60.7

Table 3. **Comparison of the average running times.** Our algorithm is 60-127 times faster.

the running times are similar in this case as well.

Evaluation when applied to puzzles with missing pieces and to mixed puzzles: Figures 1, 3, 5 show our results on puzzles with missing pieces, where the missing pieces are uniformly distributed. In particular, Figure 5 demonstrated the quality of our results on a very large puzzle.

Figure 6 illustrates our results on three different patterns of missing continuous regions, some of which are pretty large. Patterns with many missing pieces in the center are especially challenging, because in many cases the salient region of an image lies in its center.

When the orientation of the pieces is unknown, our algorithm achieves a 95.41% success rate with direct comparison and 95.4% with neighbors comparison, on the set with puzzles of 432 pieces . In comparison, the algorithm of [13] achieves a 94.7% and 94.9% success rate respectively.

Finally, Figures 2 and 7 demonstrate the results of our algorithm on complex inputs of mixed puzzles with missing pieces, where the size of every puzzle is unknown. Other examples can be found in the supplementary material.

Importance of the first piece: To understand the contribution of choosing the first piece carefully, we started from a random piece, as done in earlier works, and ran our algorithm 10 times on the dataset of Table 1. The average worst result dropped from 93.8 to 82.09 (direct comparison) & from 95.68 to 94.36 (neighbor comparison).

Limitations: Our algorithm may produce unsatisfactory results when the images are smooth or noisy, as demonstrated in Figure 8. It should be mentioned that other algorithms fail in these cases as well.

7. Conclusions

In this paper we have introduced a novel algorithm for square jigsaw puzzle assembly. We utilized a greedy approach that makes use of a compatibility function between pieces. Our function is more accurate and faster to compute than previous functions. Moreover, since greedy solvers highly depend on the initial placement, we take special care when choosing the first piece. As a result, our algorithm is

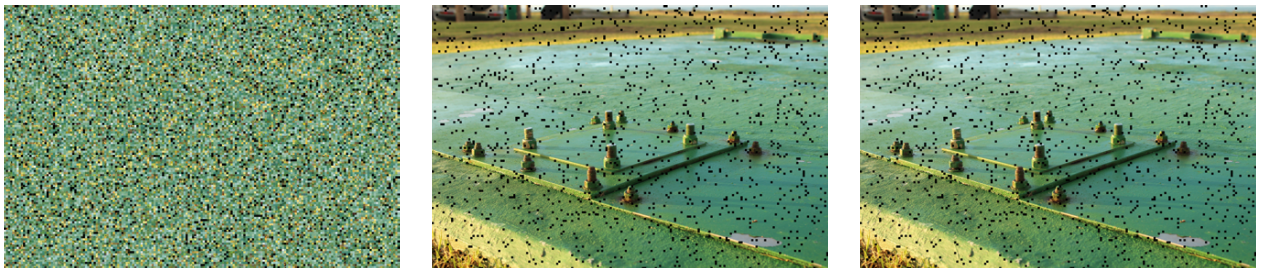


Figure 5. Our solution to a puzzle with 22755 piece, where 1160 of the pieces are missing. The puzzle is solved accurately.

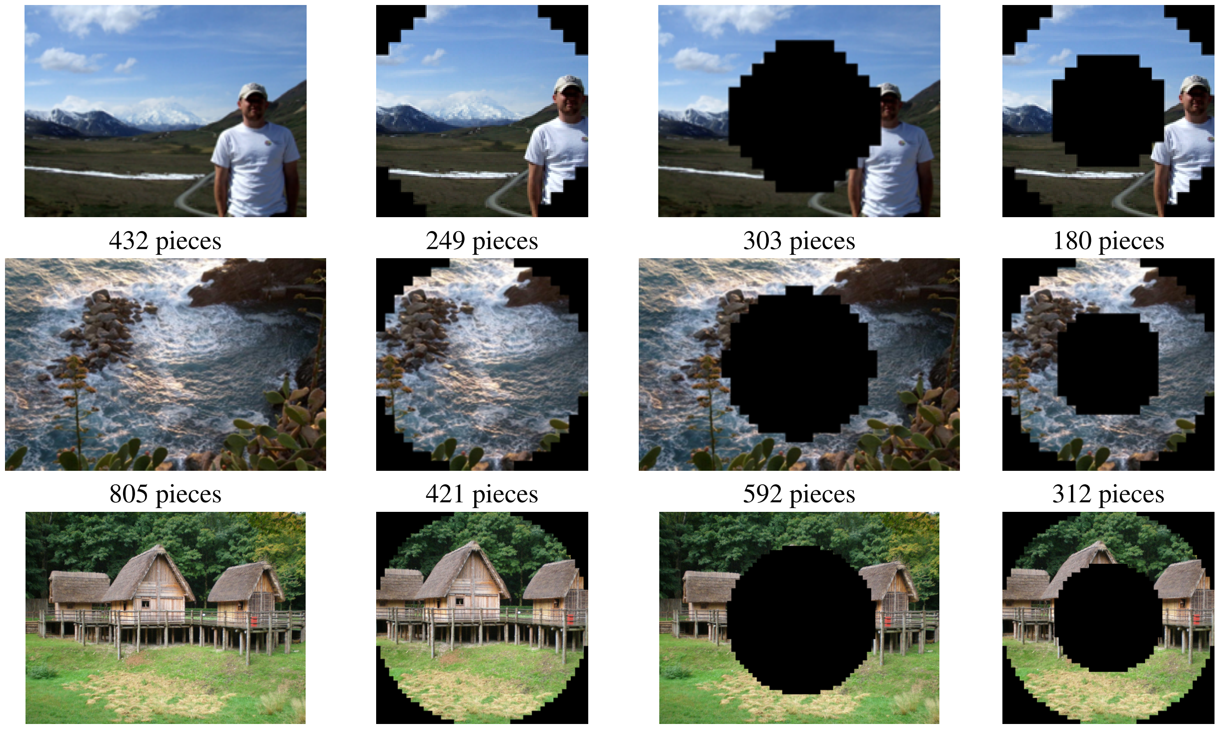


Figure 6. Our solutions to puzzles having different patterns of missing pieces. Our algorithm solved the puzzles perfectly, even though the missing regions are large.

deterministic and we need not involve the user in choosing the best solution.

We have shown that our algorithm outperforms the state-of-art results for regular puzzles, in terms of both accuracy and efficiency. Moreover, our algorithm handles successfully puzzles with missing pieces and input that contains pieces from multiple puzzles.

Acknowledgments: This research was funded in part by the Binational Science Foundation (BSF), the Israel Science Foundation (ISF) 1420/12, GRAVITATE under 2020-REFLECTIVE-7-2014, and Ollendorff Foundation.

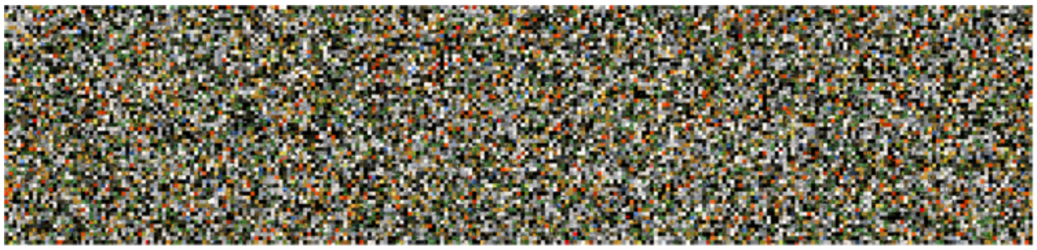
References

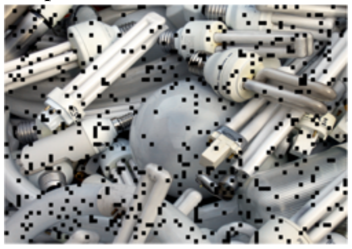
[1] T. Altman. Solving the jigsaw puzzle problem in linear time. *Applied Artificial Intelligence*, 3(4):453–462, 1990. 1

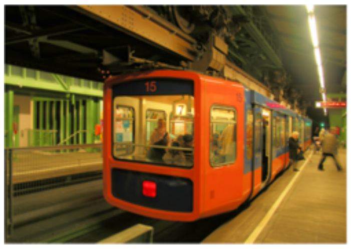
[2] B. J. Brown, C. Toler-Franklin, D. Nehab, M. Burns, D. Dobkin, A. Vlachopoulos, C. Doumas, S. Rusinkiewicz, and T. Weyrich. A system for high-volume acquisition and matching of fresco fragments: Reassembling theran wall paintings. *ACM Transactions on Graphics*, 27(3), 2008. 1

[3] S. Cao, H. Liu, and S. Yan. Automated assembly of shredded pieces from multiple photos. In *IEEE Int. Conf. on Multimedia and Expo*, pages 358–363, 2010. 1

[4] T. Cho, S. Avidan, and W. Freeman. The patch transform.







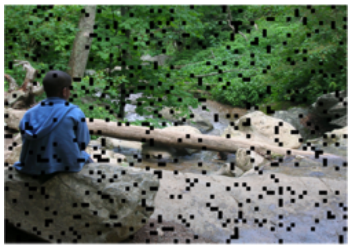


Figure 7. **Solving multiple puzzles with mixed and missing pieces.** The input contains parts from 3 puzzles. Our algorithm has no knowledge regarding the size, the number of pieces per puzzle, or the number of missing pieces. Nonetheless, it reconstructs the puzzles perfectly in 77 seconds.

PAMI, 32(8):1489–1501, 2010. 1

[5] T. Cho, S. Avidan, and W. Freeman. A probabilistic image jigsaw puzzle solver. In *CVPR*, 2010. 1, 3, 4, 5

[6] M. G. Chung, M. M. Fleck, and D. A. Forsyth. Jigsaw puzzle solver using shape and color. In *ICSP*, 1998. 1

[7] E. Demaine and M. Demaine. Jigsaw puzzles, edge matching,and polyomino packing: Connections and complexity. *Graphs and Combinatorics*, 23:195–208, 2007. 1

[8] A. Gallagher. Jigsaw puzzles with pieces of unknown orientation. In *CVPR*, 2012. 1, 3, 4

[9] D. Goldberg, C. Malon, and M. Bern. A global approach to solution of jigsaw puzzles. In *Symposium on Computational Geometry*, 2002. 1

[10] D. Koller and M. Levoy. Computer-aided reconstruction and new matches in the forma urbis romae. *Bullettino Della Commissione Archeologica Comunale di Roma*, 15:103–125, 2006. 1

[11] W. Kong and B. Kimia. On solving 2d and 3d puzzles using curve matching. In *CVPR*, 2001. 1

[12] D. A. Kosiba, P. M. Devaux, S. Balasubramanian, T. L. Gandhi, and K. Kasturi. An automatic jigsaw puzzle solver. In *ICPR*, 1994. 1

[13] K.Son, J. Hays, and D. Cooper. Solving square jigsaw puzzles with loop constraints. In *ECCV*, 2014. 1, 5, 6

[14] M. Makridis and N. Papamarkos. A new technique for solving a jigsaw puzzle. In *IEEE Image Processing*, 2006. 1

[15] W. Marande and G. Burger. Mitochondrial dna as a genomic jigsaw puzzle. *Science*, pages 318–415, 2007. 1

[16] T. R. Nielsen, P. Drewsen, and K. Hansen. Solving jigsaw puzzles using image features. *Pattern Recogn. Lett*, 2008. 1

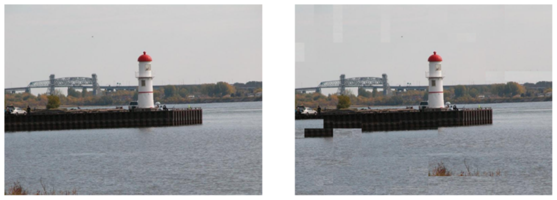


Figure 8. **Limitations:** Our algorithm may fail on smooth images.

[17] D. Pomeranz, M. Shemesh, and O. Ben-Shahar. A fully automated greedy square jigsaw puzzle solver. In *CVPR*, 2011. 1, 3, 4, 5, 6

[18] M. Sagiroglu and A. Ercil. A texture based matching approach for automated assembly of puzzles. In *ICPR*, 2006. 1

[19] D. Sholomon, O. David, and N. Netanyahu. A genetic algorithm-based solver for very large jigsaw puzzles. In *CVPR*, 2013. 1, 2, 3, 4, 5, 6

[20] R. Webster, P. LaFollette, and R. Stafford. Isthmus critical points for solving jigsaw puzzles in computer vision. In *IEEE Trans. Systems, Man and Cybernetics*, 1991. 1

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

[22] X. Yang, N. Adluru, and L. Latecki. Particle filter with state permutations for solving image jigsaw puzzles. In *CVPR*, 2011. 1

[23] F. Yao and G. Shao. A shape and image merging technique to solve jigsaw puzzles. *Pattern Recognition Let.*, 2003. 1