From Square Pieces to Brick Walls: The Next Challenge in Solving Jigsaw Puzzles

Shir Gur, Ohad Ben-Shahar
Department of Computer Science
Ben-Gurion University of the Negev
{gursh, ben-shahar}@cs.bgu.ac.il

Abstract

Research into computational jigsaw puzzle solving, an emerging theoretical problem with numerous applications, has focused in recent years on puzzles that constitute square pieces only. In this paper we wish to extend the scientific scope of appearance-based puzzle solving and consider "brick wall" jigsaw puzzles – rectangular *pieces who may have* different sizes*, and could be placed next to each other at* arbitrary offset *along their abutting edge – a more explicit configuration with properties of real world puzzles. We present the new challenges that arise in brick wall puzzles and address them in two stages. First we concentrate on the reconstruction of the puzzle (with or without missing pieces) assuming an oracle for offset assignments. We show that despite the increased complexity of the problem, under these conditions performance can be made comparable to the state-of-the-art in solving the simpler square piece puzzles, and thereby argue that solving brick wall puzzles may be reduced to finding the correct offset between two neighboring pieces. We then move on to focus on implementing the oracle computationally using a mixture of dissimilarity metrics and correlation matching. We show results on various brick wall puzzles and discuss how our work may start a new research path for the puzzle solving community.*

1. Introduction

Although jigsaw puzzles were first introduced in 1760 as a children's game to teach geography [28], nowadays they abstract a range of computational problems in which a set of unordered fragments should be organized into visual or geometrical wholes. Indeed, applications are found in fields as diverse as archeology [4, 13, 7], biology [17], reconstructing of shredded documents or photographs [1, 16], and learning visual representations [9, 18]. Theoretical work, however, has focused on square pieces only and toward that end various square piece puzzle reconstruction methods

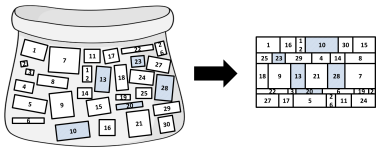


Figure 1. The structure of a "brick wall" jigsaw puzzle and its solution. A brick wall puzzle is based on organizing an unordered set of rectangular pieces into a coherent image. The spatial placement is in lines, where each brick (*i.e*. piece) can have arbitrary offset relative to its neighbors perpendicular to the offset direction. The piece numbers in this sketch replaces the pictorial content for illustration only. The shaded pieces represent the possibility of missing pieces in the puzzle (both in the input and output).

have been proposed over the years where common to many approaches is a basic operation of seeking the unassigned piece that should link to an existing piece by finding its best neighbors according to some affinity function that examines appearance differences at interfacing boundaries [23, 6, 21], or matching contours [16, 10, 24, 31, 30, 14]. Such an operation can repeat itself until a proper solution is achieved and all parts are put in place, but this somewhat naive greedy approach is likely to fail because often "best neighbors" are false-positives, a result of the occasionally unpredictable continuation between true neighbors and the properties of the metrics used to predict it. To alleviate this problem various stronger (often less local) conditions have been proposed, from the *best buddies* condition by Pomeranz *et al*. [23], through *loop constraints* by Son *et al*. [26], to *tree-based reassembly constraints* by Gallagher *et al*. [11] and *quadratic programming* by Andaló *et al*. [3] (to name but a few). These heuristics serve to avoid checking all possible permutations of piece placements in order to examine which one scores the best according to some global measure. Indeed, the problem was proved NP-complete [8] and the race has been focusing on improving result accuracy. The latter now appears to saturate at over 95% reconstruction accuracy in some cases while problem sizes (*i.e*. num-

ber of pieces) extend well beyond human solving capacity.

In this paper we wish to extend the scientific scope of computational jigsaw puzzle solving. If thus far research focused on square pieces [5, 23, 8, 2] with possible orientation uncertainty [11, 25] and missing pieces [29, 27, 20], here we wish to consider jigsaw puzzles whose pieces are *rectangular*, may have *different sizes*, and could be placed next to each other at *arbitrary offset* along their abutting edge. With these properties in mind our puzzles are best described as "brick wall" where the lines (or rows) of bricks may have different heights, each brick has its own width, and its position in its "brick line" could be arbitrary. Since "brick lines" can be either vertical or horizontal, to simplify presentation we will focus on one case only. Without loss of generality we therefore consider *vertical* brick lines and thus brick "columns". This situation abstracts the most obvious application of brick wall puzzles, namely shredded documents, whose importance was demonstrated by a DARPA challenge [1]. The latter, however, resulted in semi-automatic solutions with humans in the loop, while here not only do we seek a formal abstraction of the problem, but also fully-automatic algorithms. Without loss of generality we will also focus on the case where columns have equal width[1]. Figure 1 illustrates the geometric structure of a typical "brick wall" jigsaw puzzle.

New challenges arise in brick wall puzzles. To facilitate both the present and future research, we divide the problem into two sub-problems. The first concentrates on the reconstruction of brick wall puzzles assuming an oracle for offset assignments, and the second focuses on implementing (an imperfect version of) the oracle computationally using a mixture of dissimilarity metrics and correlation matching.

2. The "Brick Wall" puzzle problem

Recall the elements of the square piece jigsaw puzzle problem: pieces are square and identical in size and their placement is such that their vertices always meet vertices of other pieces. This setup is interesting from a computational point of view because appearance is the sole cue for reconstruction, a very different condition from the original jigsaw puzzles where pieces are endowed with shape, and a far less constrained condition compared to a typical "torn-page" puzzles where the geometrical constraints are far more informative than appearance cues [16].

However, the geometrical setup can be more complicated than square pieces meeting in vertices and yet provide *less* constraints to reconstruction. In (vertical) brick wall puzzles pieces are rectangles, they have fixed width but at the same time can have different heights. As a consequence, pieces do not always meet at their vertices and thus may be found at arbitrary offset (or shift) relative to their neighbors. As illustrated in figure 2a, we will denote the offset between the top edges of piece x_i and piece x_j by $S^i_j(R)$, where R is the relation between the pieces. In this sense the brick wall puzzle problem is in fact a strict generalization of the square piece problem. Its new degrees of freedom greatly expands the complexity of the challenge because each puzzle piece may have many possible neighborhood configurations. Indeed, if square piece puzzles could have 4 possible neighborhood configurations between two given pieces (or 16, if piece orientation is unknown also), now we have $2(H(x_i)+H(x_j)-1)+2$ possible neighborhood configurations, where $H(x_i)$ is the height (in number of pixels) of piece x_i. For our present work we assumed that the orientation of pieces is known because only 180° rotations could keep the brick wall configuration under rectangular pieces. At the same time, we further complicate the brick wall puzzle problem by allowing missing pieces as well.

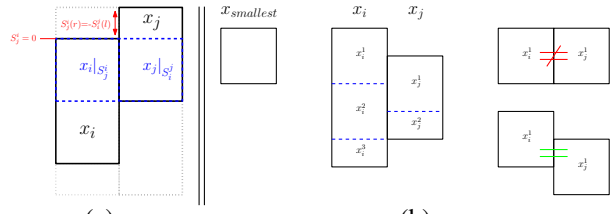


Figure 2. **(a)** Neighborhood relationships in brick wall puzzles. Two pieces x_i and x_j are aligned with a shift offset between their top edges. When $S^j_i = 0$ there is no offset between the pieces and their top edges match. By definition, the offsets $S^i_j(R)$ and $S^j_i(R^{-1})$ are opposing. **(b)** Illustration of the irreducibility of brick wall puzzles to square (of fix size) piece puzzle problem.

To address the brick wall puzzle problem we first note that it is impossible to reduce it to a square piece puzzle by cutting the pieces to fixed size, in particular to the greatest common divisor (GCD) of piece heights. Unfortunately, such manipulation would not eliminate the need to join pieces away from their vertices and therefore will not remove the need to find correct offset between pieces (see Fig. 2b), a critical factor absent from the square (or fixed size) piece puzzle problem. This observation is true even if no residuals remain after cutting the pieces to some fix size, and needless to say that it becomes even more severe if one allows missing pieces (as we do). A new type of solution is thus needed, and to pursue it we divide the problem into two parts. First we devise a reconstruction algorithm that assumes an oracle for offset assignments. More specifically, when queried on two true neighboring pieces, the oracle returns their true relative offset, but if the pieces are not neighbors, it returns a random value (*i.e.* the oracle itself cannot indicate if pieces are neighbors or not). Note that assuming such an offset oracle does *not* reduce brick wall puzzles into square piece-like jigsaw puzzle because each piece may

[1] If this is not the case then piece width adds additional constraint that in fact simplifies the solution rather than complicates it.

still have *multiple correct neighbors* at either side, a condition that is strictly prohibited in the latter classical (square piece) problem. On top of that, because we wish to allow missing pieces, the geometrical relationship between pieces both within and between columns are further loosened up to increase the complexity of the problem as a whole. As we show, despite the increased complexity that requires different treatment, performance can be made extremely good on brick wall puzzles, and comparable to previous work on square piece puzzles. Thus, in this sense we show that solving brick wall puzzles may be reduced to finding the correct offset between two pieces. We thus then turn to address this part of the problem and focus on possible implementations of the oracle to find piece offsets computationally using tools formed for this task. While results on this end are still imperfect, we hope that our present work will start a new research path by the puzzle solving community.

2.1. Problem formulation

Given a set of unordered rectangular image pieces of equal-width, with the possibility of missing pieces and an arbitrary shift at the correct assignment between them, we seek to reconstruct the original image.

We denote each piece as x_i of dimensions $H(x_i) \times W$ pixels, where $H(x_i)$ is the height of piece x_i. Each piece x_i needs to be placed correctly next to some x_j with relation $R \in$ {l=Left, r=Right, u=Up, d=Down} and a relative shift $S^i_j(R)$ where here $R \in$ {l, r}. Following the definition of $S^i_j(R)$, we define $x_i\big|_{S^i_j(R)}$ as the sub-region of x_i that overlaps x_j according to the shift $S^i_j(R)$ (see Fig. 2), where we later describe its use in our algorithm in Sec. 4.

2.2. Problem complexity

As mentioned before, the single and most critical factor that differentiates brick wall puzzles from the simpler square piece puzzles is the possibility to have multiple neighbors at either side of each piece. While we will not elaborate on the exact combinatorial formulation, a brief discussion of the key features that influence the complexity is worth telling. Let us consider the number of possibilities one needs to examine (had the search was exhaustive) in trying to assign a new neighbor to one side of a piece which already belongs to the reconstructed puzzle. Let n be the number of pieces left unassigned from which this new neighbor should be selected. Since in the original square piece puzzle problem one could have exactly one neighbor at each side, the number of possibilities in the worst case is $\binom{n}{1} = n$. To compute the analogous number in the brick wall puzzle problem, we first realize that the number of neighbors can be much larger and is dictated by the size ratio between the largest and smallest pieces in the puzzle. If this Maximum Size Ratio is MSR $= \lceil \frac{max-1}{min} \rceil$ then the maximum number of neighbors at each side can be at most MSR+1 which implies $\binom{n}{\text{MSR}}$ possible combinations, or approximately $O(n^{\text{MSR}})$ possibilities in the typical case where $n >>$ MSR. It is this polynomial factor compared to the square piece puzzle problem that makes our new problem particularly challenging.

3. A Base algorithm for square-piece puzzles

In order to approach the new problem as a generalization of the square piece puzzle problem we first devise a base reconstruction algorithm that is able to solve the square piece jigsaw puzzles with missing pieces. Not unlike previous work, we too employ both dissimilarity and compatibility measures (*e.g.* see [23, 11, 20, 27]), but we frame the base algorithm in a modular way that prepares the ground and permits the extensions needed for brick walls.

3.1. Dissimilarity

Let $D(x_i, x_j, R)$ be the dissimilarity between two pieces x_i, x_j with relation *R*, where *R*∈(l, r, u, d). Numerous types of dissimilarity functions have been used in previous works, including various norms between the boundary pixels of the two pieces and/or their gradients (*e.g.* [5, 23, 11]), or between the boundary pixels of one piece and the corresponding pixels *as predicted* by the other piece [23, 20]. In this paper we use the latter type of dissimilarity with L_2 norm, *i.e.* the right relation equation becomes:

$$D(x_i, x_j, r) = \sum_{h=1}^{H(x_i)} \sum_{c=1}^{3} (2x_i(h, W, c) - x_i(h, W-1, c) - x_j(h, 1, c))^2 \quad (1)$$

where h runs over the rows of the piece, W is its width and the index of the last column, and c is the color channel. The equations for the other relations are derived similarly. We note that this dissimilarity function is not a metric since $D(x_i, x_j, R)$ is not necessary equal to $D(x_j, x_i, R^{-1})$.

When extending the problem to brick wall puzzle pieces, dissimilarity for *left* and *right* relations will be consider only along the overlapping sub-region $x_i\big|_{S^i_j}, x_j\big|_{S^j_i}$ and normalize by the length (height) of the abutting boundary. More formally, the adjusted dissimilarity for left and right relations in brick wall puzzles becomes:

$$D|_{S^i_j}(x_i, x_j, R) = \frac{D(x_i\big|_{S^i_j}, x_j\big|_{S^j_i}, R)}{H(x_i\big|_{S^i_j})} \quad (2)$$

where $H(\cdot)$ is again the height of a piece (see also Sec. 2.1).

3.2. Compatibility

The compatibility function $C(x_i, x_j, R)$ is a key measure in determining how likely it is for two pieces x_i and x_j to neighbors with relation *R*. Optimally, this function, that depends strongly on the dissimilarity measure, will return 1 *iff* x_i and x_j are true neighbors with relation *R*, and 0 otherwise. If such a function existed, the jigsaw puzzle problem

could be solved in polynomial time by a simple greedy algorithm [8]. Evolving from Pomeranz *et al*. [23], here we use the compatibility function from [20], *i.e*.,

$$C(x_i, x_j, R) = 1 - \frac{D(x_i, x_j, R)}{secondBest(i, R)} \quad (3)$$

where $secondBest$ is defined as the dissimilarity $D(x_i, x_k, R)$ with x_k being the second best piece in the dissimilarity ranks of x_i.

Having the compatibility, we rank piece x_j as the best neighbor of piece x_i with relation *R iff*:

$$\forall x_k \in Parts : C(x_i, x_j, R) \geq C(x_i, x_k, R) \quad (4)$$

and x_i, x_j are called *best buddies* [23] *iff* they both agree to be best neighbors with relations *R* and R^{-1}, respectively.

As we discussed above, in the case of brick walls each piece can have multiple neighbors. It does not change the ranking method but it does influence the implementation, where instead of enabling only one neighbor at each side of x_i, we allow as many neighbors as necessary to cover the edge of a piece. We will return to this point after describing the base algorithm that handles square pieces.

3.3. The base algorithm

The base algorithm first calculates the dissimilarity and compatibility between all pieces using the definitions above. Next, it extracts a good piece to start with by looking for one that has best buddies in all four directions, such that these neighbors also have best buddies in all four directions. Multiple candidates are ranked as in [20].

The main part of the base algorithm manages a candidates set C where each piece is regarded legit for the coming placements. Except for the initial piece, new candidates are added to C when a piece is fetched out of C and added to the reconstructed puzzle. Then, its best buddies, or, if none exists, its best neighbors, are added to C.

Choosing the next piece to place from the set C is a crucial step, since a bad choice could accumulate into major errors and eventually poor results. With this in mind we compute 4 different measures that facilitate informed ranking of the candidates in C: two measures of *support*, one measure of *compatibility*, and the number of best buddies a owns has (see below). The best candidate x_i is selected according to these measures and its assigned location in the reconstructed puzzle is then computed.

It is important to realize that x_i may not have a vacant location next to its already-placed neighbors, because all desired locations are already occupied by more fit candidates. In that case we discard the connection between x_i and piece x_j who pooled him to C with the relevant relation. For that reason, when C is empty we recalculate the compatibility and neighbors connections while considering only the remaining unplaced pieces. This results in an updated compatibility which implies new best neighbors and best buddies. Next, we extract all neighbors of the partially reconstructed puzzle, henceforth the "unplaced" set. As was done with the set C, we now follow the same procedure of sorting, extracting, and placing a new piece and adding its neighbors to set C.

The assignment of location for piece x_i is obviously a critical step. First, we recover the best buddies of x_i (or best neighbors, if no best buddy exists) that have already been placed. Let us denote this set as X_j. Second, we calculate candidate positions from the relative relation of x_i and its placed neighbors X_j. Finally, each assignment of x_i in a candidate position is evaluated according to the four measures, and the best position is returned. If all positions are occupied we discard the connection as mentioned before, a "false" result is then returned to indicate unresolved placement, and the next piece in the set is now examined. Alg. 1 describes the main extraction procedure, and Alg. 2 details the main reconstruction loop.

Algorithm 1 Extract

```
Require: Set s
  Sort(s) according to ranking measures
  part ←Pop(s)
  if (x, y) ←Get_Best_Placement(part) then
    Place(part, x, y)
    Add_Neighbors(part)
    return True
  else
    return False
```

Algorithm 2 Reconstruct Puzzle

```
Require: An unordered set of puzzle "bricks"
  part ←Select_Best_Seed()
  Place(part)
  Add_Neighbors(part)
  while C is not Empty do
    Extract(C)
    if C is Empty then
      Recalculate_Parameters()
      unplaced ← Get_Unplaced_Parts()
      while unplaced is not Empty do
        if Extract(unplaced) then
          break
```

Missing from the top level description of the algorithms is how we rank unassigned candidate pieces in sets C and $unplaced$ with the 4 measures mentioned. The details of these ranking, in particular the measure of support, serve the goal of reconstructing the puzzle while keeping the reconstructed regions as "convex" as possible – we will prefer to assign unassigned pieces that not only are supported (in terms of high compatibility) by as many assigned neighbors as possible, but those whose candidate neighbors are also supported in a similar way (see Sec. 3.4 below). If no support is found, that best candidate will be assigned where it

has large compatibility and as many best buddies as possible, but it is still considered as a less confident assignment.

3.4. Support

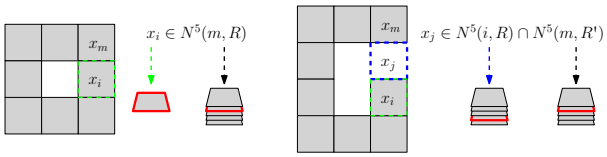


Figure 3. An illustration of 1^{st} order (left) and 2^{nd} order (right) Support. Green dashed pieces represent the current candidate x_i in its examined position for the support calculation. Red pieces indicate matches according to the support and Blue dashed piece represent an unplaced neighbor of x_i that is a k-neighbor of *both* x_i and a placed piece x_m. Both support measures count how many combinations like those shown exist for x_i and by doing so how much support this candidate location obtains from the placed pieces.

As implied above, our new reconstruction approach utilizes information from the pieces that were already assigned to the reconstructed puzzle in order to assist the placement of subsequent ones. As much as knowing the compatibility between two pieces is constructive, it still yields false assignments. By "querying" the reconstructed section about upcoming assignment we can tell if it is likely to cause problems over the next steps and thus avoid them if needed. This query attempts to find the support a piece can get, a measure divided into two parts, dubbed below a first order and second order support (*support1* and *support2*, respectively).

Toward that end, we first say that piece x_i is a *k-neighbor* of x_j with relation R if x_i is ranked at x_j top k neighbors in the opposite relation R. While other small constants are possible, we used $k = 5$ and denote:

$$N^k(j,R) = \{x_j \text{ top-k neighbors in relation R}\} \quad (5)$$

Let x_i be the piece wish to place next. The first order support measure, *Support1*, is defined as the number of placed neighbors x_m of x_i such that x_i is their *k-neighbor* in the proper relative relation, $x_i \in N^k(m,R)$. The second order support, *support2*, is defined as the number of assigned pieces x_m such that $N^k(i,R)$ and $N^k(m,R')$ share a piece x_j, where R' is the proper relation between x_i neighbors $N^k(i,R)$ *and* x_m. Figure 3 illustrates these two cases.

With the support measures defined, the algorithm just described can solve the square piece puzzle problem with performance on par with the state-of-the-art. While results are discussed in Sec. 5, this algorithm is not our goal but merely a means to devise a reconstruction algorithm for the brick wall puzzle problem, as discussed next.

4. Brick wall reconstruction algorithm

In this section we present the modifications needed for the base algorithm in order to modify it to handle the brick wall problem. There are two main issues that need to be considered. First, each piece will have multiple neighbors. Second, we will need to predict the correct offset even if neighbors are determined correctly. As we commented earlier, this second part will first be addressed by an oracle, prior handling it computationally.

4.1. Handling multiple neighbors

One way to address the multitude of possible neighbors is to handle them one at a time. Indeed, in our algorithm we maintain just one *best* neighbor for each piece despite the fact that the latter may eventually have several neighbors. The guiding assumption is that it is safer to consider (and in particular, assign) the most compatible piece before others, and thus need not worry about the latter before their turn. When a connection between two pieces is either realized (placed next to each other in the reconstructed puzzle) or discarded (due to a conflict with previously placed piece), we can ignore their compatibility and allow each of them to obtain a new best neighbor, if so possible (*i.e.* if their common edge was not exhausted by neighbors).

Another implication of multiple neighbors is that searching for the best placement for part x_i now becomes more complex, as we are not only checking if a specific side of an already placed piece x_j is available, but we also need to come up with the best offset between the pieces and to make sure that this offset does not make x_i overlap or intersect previously assigned neighboring pieces along that edge. In anticipation for errors in the offset prediction module, we endow the process with a "push" mechanism that allows placements that overlap up to M pixels to push its conflicting prospective neighbors from above or below (recall that we are speaking of vertical brick walls) accordingly in order to generate the required free space. While this parameter may be optimized, in our experiments we select $M = 5$.

4.2. Offset estimation

In our initial stage we implemented the algorithm above while obtaining the offset values between prospective neighbors from an oracle. This oracle provided the correct offset when queried on correct neighbors and a random offset when the pieces are not true neighbors. In reality, of course, such an oracle does not exist and it needs to be estimated computationally. As the results show in Figure 6 and Table 1, this is the main challenge in brick wall puzzles and in order to address it we apply yet another metric we call the *correlation metric* between pieces.

The correlation metric is defined as a measure of dissimilarity between two series as a function of their relative phase, or offset. As it turns out, several standard tools for detecting the proper phase between signals do not perform well enough in our context. We discuss the issues with these tools and present an alternative for better estimation.

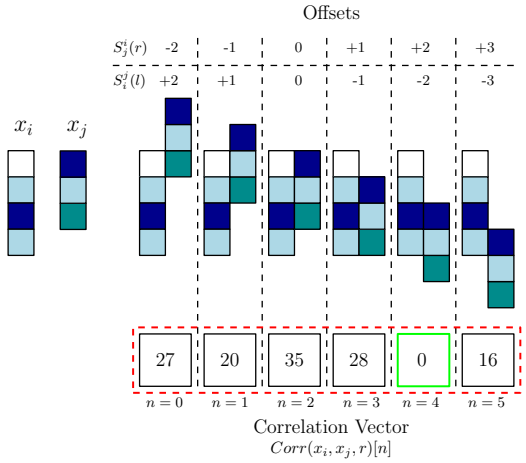


Figure 4. Illustration of correlation process between two pieces with $H(x_i) = 4$ and $H(x_j) = 3$, respectively whose abutting boundaries are shown on the left. Pixel colors and correlation results are fictitious for the illustration only. Red dotted box represents the correlation vector $Corr(x_i, x_j, r)[n]$ of a *right* relation between x_i and x_j, where here $n = 4$ is the minimal result highlighted in green. $S_i^j = -2$ and $S_j^i = +2$ are the result of the offset estimation corresponding to $n = 4$.

Indeed, one of the standard tools in computer vision for matching signals is cross-correlation [22], a measure used widely for patch matching or object tracking [15]. One of its advantages is the ability to compute it in the *Fourier domain* [12] and by that improve computational cost. In our attempt to replace the oracle computationally we employ this tool, as well as the Euclidean distance, but will also show that by themselves they are not performing well enough. In the following we develop the proposed computation for the *right* relation, though corresponding equations for the other relation are completely analogous. To further simplify the presentation we assume (without loss of generality) that x_i is the predicted boundary of x_j as described in section 3.1 and reflected in the following equation:

$$x_i(h, W, c) = 2x_i(h, W, c) - x_i(h, W-1, c) \tag{6}$$

For convenience we also refer to the vertical (*i.e.* height) indices as if the top one in the sub-region of overlap between x_i and x_j is numbered 0 (refer again to Fig. 2). Hereby we develop a few baseline equations based on intuitive function and dissimilarity measures proposed by previous work.

Consider the following two basic types of interaction:

$$\pi_{i,j}(h, c, n) = x_i(h, W, c)x_j(h, 1, c) \tag{7}$$
$$\delta_{i,j}(h, c, n) = x_i(h, W, c) - x_j(h, 1, c) \tag{8}$$

where 7 is a similarity measure at the foundations of cross-correlation and 8 is dissimilarity measure at the foundations of L_1 norm distance. We remind that the original index h of each x_i and x_j for a given offset n is different and derive from the sub-regions.

Let $Corr(x_i, x_j, R)$ be the vector consisting of all matching results for all possible relative offsets between the two pieces x_i and x_j by combining the basic measures in Eqs. 7 and 8 into a normalized cross-correlation or L_2 norm (or SSD), and normalizing accordingly. The following equations refer to 7 and 8 respectively:

$$Corr(x_i, x_j, R)[n] = 1 - \left\| \frac{1}{\tilde{H}} \sum_h \sum_c \pi_{i,j}(h, c, n) \right\| \tag{9}$$

$$Corr(x_i, x_j, R)[n] = \frac{1}{\tilde{H}} \sum_h \sum_c \left(\delta_{i,j}(h, c, n)\right)^2 \tag{10}$$

where we define $\tilde{H} = H\left(x_i\big|_{S_j^i(R)=n}\right)$ as the height of x_i sub-region when $S_j^i(R) = n$. Intuitively, we now define the shift value between two pieces x_i and x_j to be:

$$S_j^i(R) = \arg\min_n Corr(x_i, x_j, R) - \left(H(x_j) - 1\right) \tag{11}$$

or in other words, the predicted offset is one that minimize the correlation vector. Figure 4 illustrates the correlation process and the above definitions.

Further elaborating in the spirit of the L_p^q norm by Pomeranz *et al*. [23], we also defined the measure:

$$Corr(x_i, x_j, R)[n] = \frac{1}{\tilde{H}} \left[\sum_h \sum_c \left(\delta_{i,j}(h, c, n)\right)^p \right]^{\frac{q}{p}} \tag{12}$$

and one based on the Mahalanobis distance:

$$\Theta_{i,j}(h, c, n) = (\tilde{x}_i - \tilde{x}_j) COV^{-1} (\tilde{x}_i - \tilde{x}_j) \tag{13}$$

where COV is the covariance matrix and $\tilde{x}$ is x normalized by its mean and standard deviation. This results in following correlation function $Corr(x_i, x_j, R)[n]$:

$$Corr(x_i, x_j, R)[n] = \frac{1}{\tilde{H}} \sum_h \sum_c \Theta_{i,j}(h, c, n) \tag{14}$$

Combining all of the above, and given poor results for each of the measures by itself (*c.f*. Fig. 6) , we propose to combine several of the correlation measures as follows:

$$Corr(x_i, x_j, R)[n] = \frac{1}{\tilde{H}} \sum_h \sum_c \left(\Theta_{i,j}(h, c, n) + 1\right)\left(\delta_{i,j}(h, c, n) + 1\right) \tag{15}$$

Ultimately, we are able to estimate the offset more precisely and compute relative dissimilarity between x_i and x_j, but unlike the base square-piece algorithm, we of course use it only over $x_i\big|_{S_j^i}$ and $x_j\big|_{S_i^j}$ as mentioned in section 2.1.

Two main issues arise in this process. First, the computational time of this metric is two orders of magnitude (*i.e.* $\sim$ 100) slower than calculating the dissimilarity function, the slowest computational component in previous work and in our base algorithm. Second, it is not guaranteed that the "best" result obtained by this computation is also the correct one. We discuss this issue in Sec. 5.3 where we consider two cases, one when a wrong placement is made due

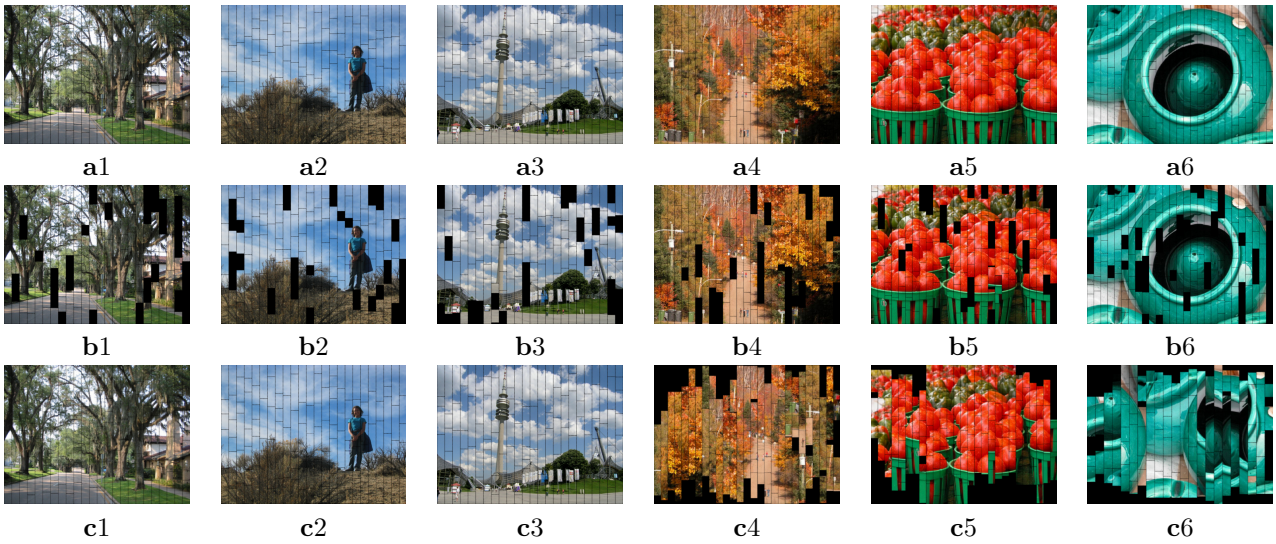


Figure 5. Results of three types of puzzles over two datasets. images 1-3 from [5] and 4-6 from [19]. **(a)** Brick Wall puzzles with oracle. **(b)** Brick Wall puzzles with oracle and 15% missing pieces. **(c)** Brick Wall puzzles without oracle.

to wrong offset and the other when wrong offset is not realized thanks to other neighbor who has stronger relation and a correct offset with the current piece.

5. Results

In this section we discuss three types of brick wall puzzle results. As our work addresses a new type of jigsaw puzzle problem, a comparison to a state-of-the-art in brick wall puzzles is impossible. However, to set some baseline comparison, we first tested our base algorithm on square piece puzzles and compared it to the prior art. These results are shown in Table 1. Although the base algorithm is designed to be "looser" than previous square puzzle algorithms in order to allow the extension to brick walls, performance is comparable and only slightly inferior to the state-of-the-art. Another baseline comparison to the prior art focuses on the estimation metrics and is described below (Sec. 5.2).

With the baseline of the base algorithm established, we continue to our primary evaluation. We first show results of brick wall reconstructions with offset oracle, evaluated on brick wall puzzles *with* and *without* missing pieces. Next, we evaluate and compare the performance of the 5 different correlation metrics to reveal limitations and opportunities ,and finally we show results for brick wall algorithm with predicted offset (*i.e.*, *without* offset oracle) based on our proposed correlation measure. For each of the above, puzzles are generated by randomly splitting each test image to 10 different sets of brick pieces from which average performance and standard deviation are presented.

The running time of our brick wall (with oracle) and base solvers is faster than most previously proposed algorithms, while without an oracle it is roughly ~ 50 times slower.

5.1. "Brick walls" with oracle

Results of the brick wall algorithm with oracle are presented in Fig. 5 a-b. Row (a) shows perfect reconstruction of images, indicating that the proposed algorithm is capable of dealing with the new problem and handles well the new degrees of freedom. Row (b) shows solutions of brick wall puzzles with missing pieces. Table 1 presents results of the percentage of correct neighbors as a function of the MSR, where the pieces that were curved off the images to form the input of the algorithms were set to have uniformly distributed random height from 28 pixels up to 28·MSR pixels.

Two interesting points emerge from these results. First, being so excellent, the result with an oracle suggests that brick wall puzzles can be essentially reduced to the problem of correctly estimating the offset between prospective neighbors. Second, and somewhat counter-intuitive, is the observation that results appear to improve as the MSR increases. This happens due to our greedy assembly that considers only one best neighbor, therefore the amount of pieces is highly significant.

5.2. Correlation metrics performance

Figure 6 shows statistics of the five suggested correlation functions from Sec 4.2. We tested these functions between two pieces of 75 pixels in height [2]. The graph shows the prediction value of each possible offset (shift) averaged over 360 piece samples. Here the optimal line represents the target function and it is clear that the intuitive functions under-preform the proposed metrics for dissimilarity. This

[2]This size is larger than the typical 28 pixel piece used in most previous square piece puzzle solvers, but for brick walls it is reasonable to allow a descent range of MSRs.

	MIT [5]		McGill [19])		BGU [23] (980 × 644)		BGU [23] (1652 × 1120)		BGU [23] (1848 × 1400)	
	Mean	STD	Mean	STD	Mean	STD	Mean	STD	Mean	STD
Algorithm	**Original square-pieces puzzles**									
Pomeranz *et al.* [23]	95.0%	-	90.9%	-	89.7%	-	84.7%	-	85.0%	-
Sholomon *et al.* [25]	**96.2**%	-	96.0%	-	**96.3**%	-	88.9%	-	92.8%	-
Son *et al.* [26]	95.5%	-	95.2%	-	94.9%	-	96.4%	-	**96.4**%	-
Paikin *et al.* [20]	95.8%	-	**96.1**%	-	95.1%	-	96.3%	-	95.3%	-
Son *et al.* [27]	95.5%	-	**96.1**%	-	95.0%	-	**96.7**%	-	95.1%	-
Proposed Base Alg.	95.8%	-	91.6%	-	91.7%	-	91.8%	-	92%	-
MSR	**"Brick wall" puzzles - With Oracle**									
2	93.5%	11.0%	96.1%	11.4%	92.8%	10.2%	96.5%	3.3%	99.5%	0.8%
3	93.6%	11.6%	98.4%	5.4%	95.6%	6.5%	96.7%	3.1%	99.6%	0.7%
4	**94.4**%	9.8%	97.8%	7.1%	96.6%	6.9%	97.5%	3.8%	99.7%	0.4%
5	**94.4**%	9.8%	**99.0**%	2.0%	98.0%	3.9%	**98.9**%	1.5%	**99.8**%	0.3%
6	93.4%	13.0%	97.9%	2.8%	**98.5**%	2.8%	97.5%	3.3%	**99.8**%	0.4%
MSR	**"Brick wall" puzzles - Without Oracle**									
2	70.5%	18.3%	56.1%	19.7%	**52.1**%	13.2%	**54.2**%	7.6%	**55.9**%	7.2%
3	70.2%	17.8%	**57.5**%	20.8%	51.4%	14.6%	51.9%	8.5%	53.4%	5.4%
4	**74.7**%	17.3%	52.3%	17.2%	51.6%	11.0%	52.1%	6.8%	53.3%	5.9%
5	71.7%	20.6%	51.2%	15.5%	51.5%	11.3%	52.0%	7.3%	52.5%	6.8%
6	73.6%	20.5%	51.3%	14.9%	51.9%	14.7%	51.3%	7.1%	52.5%	7.4%

Table 1. **Correct neighbors performance**. **(Top table)** Square-pieces jigsaw puzzles. **(Bottom tables)** "Brick Wall" with random heights.

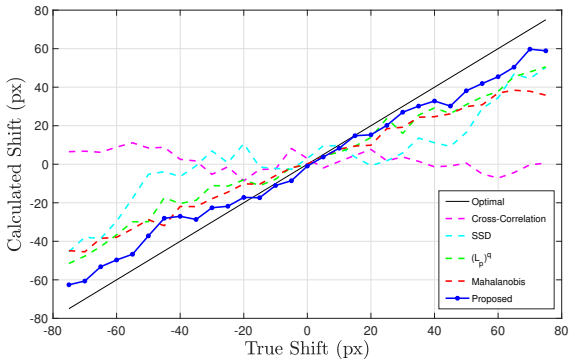


Figure 6. Average correlation metrics performance over pieces of height 75px. The optimal line represent the target function we wish to predict. Some metrics perform poorly while the one we propose in Eq. 15 outperform the rest.

implies that a good dissimilarity function may be good also for correlation. Note how our proposed correlation function outperforms the rest of the baseline measures.

5.3. "Brick wall" without oracle

Fig. 5 (c) and Table 1 show results of brick wall puzzles with predicted offsets (*i.e.*, where the oracle is implemented computationally). As expected, compared to the use of an oracle, the average result are degraded. However, both Fig. 5 and standard deviation results suggest that while the proposed algorithm can fail on some puzzles, it can achieve excellent (and sometimes perfect) accuracy on others.

A further examination of results reveals two cases for success and failure. Consider two pieces x_i and x_j for which one computes a wrong offset estimation. Best case scenario dictates that it will result with a correct assignment of x_i with another neighbor x_k, forcing the correct later assignment of x_i and x_j. Indeed, the possibility to have several neighbors in the brick wall puzzle problem implies opportunities for overcoming bad placement estimations but also more room for mistakes. Worst case scenario suggests that x_i and x_j will be placed as neighbors at that wrong offset, preventing the true neighbor(s) from being placed correctly later on. Panels $c4$-$c6$ in Fig. 5 show fragments of correct matches in harder images.

6. Conclusion

In this paper we extended the scope of visual jigsaw puzzle problem to 'brick walls" - a strict generalization of the square piece puzzles with rectangular pieces of different shapes and multiple neighbors. We presented a new algorithm that addressed such problems with missing pieces. When the offset between prospective neighbors are handled by an oracle, result are superior, suggesting that the main aspect in solving brick wall puzzles is the correct estimation of these offsets. Combined with a possible approach for making these estimations, we conclude that future research on brick wall puzzles is likely to steer this way.

Acknowledgments

This research was supported in part by the Israel Science Foundation (ISF FIRST/BIKURA Grant 281/15) and the European Commission (Horizon 2020 grant SWEEPER GA no 644313). We also thank the Frankel Fund and the Helmsley Charitable Trust through the ABC Robotics Initiative, both at Ben-Gurion University of the Negev.

References

[1] The darpa shredder challenge. 1, 2

[2] N. Alajlan. Solving square jigsaw puzzles using dynamic programming and the hungarian procedure. *American Journal of Applied Sciences*, 11:1942–1948, 6 2009. 2

[3] F. A. Andaló, G. Taubin, and S. Goldenstein. Solving image puzzles with a simple quadratic programming formulation. In *Graphics, Patterns and Images (SIBGRAPI), 2012 25th SIBGRAPI Conference on*, pages 63–70. IEEE, 2012. 1

[4] 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

[5] T. S. Cho, S. Avidan, and W. T. Freeman. A probabilistic image jigsaw puzzle solver. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 183–190, 2010. 2, 3, 7, 8

[6] M. G. Chung, M. M. Fleck, and D. A. Forsyth. Jigsaw puzzle solver using shape and color. In *Signal Processing Proceedings, 1998. ICSP'98. 1998 Fourth International Conference on*, volume 2, pages 877–880. IEEE, 1998. 1

[7] M. E. C.-s. T. D. F. Constantin Papaodysseus, Thanasis Panagopoulos and C. Doumas. Contour- shape based reconstruction of fragmented, 1600 b.c. wall paintings. 50:1277–1288, 2002. 1

[8] E. Demaine and M. Demaine. Jigsaw puzzles, edge matching, and polyomino packing: Connections and complexity. *Graphs and Combinatorics*, 23:195–208, 2007. 1, 2, 4

[9] C. Doersch, A. Gupta, and A. A. Efros. Unsupervised visual representation learning by context prediction. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1422–1430, 2015. 1

[10] H. Freeman and L. Garder. Apictorial jigsaw puzzles: the computer solution of a problem in pattern recognition. *Electronic Computers, IEEE Transactions on*, 13:118–127, 1964. 1

[11] 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. 1, 2, 3

[12] F. J. Harris. On the use of windows for harmonic analysis with the discrete fourier transform. *Proceedings of the IEEE*, 66(1):51–83, 1978. 6

[13] 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

[14] W. Kong and B. Kimia. On solving 2d and 3d puzzles using curve matching. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2001. 1

[15] J. Lewis. Fast normalized cross-correlation. 10(1):120–123, 1995. 6

[16] H. Liu, S. Cao, and S. Yan. Automated assembly of shredded pieces from multiple photos. *Multimedia, IEEE Transactions on*, 13(5):1154–1162, 2011. 1, 2

[17] W. Marande and G. Burger. Mitochondrial dna as a genomic jigsaw puzzle. In *Science*, volume 318, page 415, 2007. 1

[18] M. Noroozi and P. Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. *arXiv preprint arXiv:1603.09246*, 2016. 1

[19] A. Olmos and F. A. A. Kingdom. McGill calibrated colour image database. http://tabby.vision.mcgill.ca., 2005. 7, 8

[20] G. Paikin and A. Tal. Solving multiple square jigsaw puzzles with missing piece. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4832–4839, 2015. 2, 3, 4, 8

[21] C. Papaodysseus, T. Panagopoulos, M. Exarhos, C. Triantafillou, D. Fragoulis, and C. Doumas. Contour-shape based reconstruction of fragmented, 1600 bc wall paintings. *IEEE Transactions on Signal Processing*, 50(6):1277–1288, 2002. 1

[22] K. Pearson. Mathematical contributions to the theory of evolution. iii. regression, heredity, and panmixia. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 187:253–318, 1896. 6

[23] D. Pomeranz, M. Shemesh, and O. Ben-Shahar. A fully automated greedy square jigsaw puzzle solver. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9–16, 2011. 1, 2, 3, 4, 6, 8

[24] G. M. Radack and N. I. Badler. Jigsaw puzzle matching using a boundary-centered polar encoding. *Computer Graphics and Image Processing*, 19(1):1–17, 1982. 1

[25] D. Sholomon, O. David, and N. S. Netanyahu. A genetic algorithm-based solver for very large jigsaw puzzles. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 1767–1774. IEEE, 2013. 2, 8

[26] K. Son, J. Hays, and D. B. Cooper. Solving square jigsaw puzzles with loop constraints. In *European Conference on Computer Vision*, pages 32–46. Springer, 2014. 1, 8

[27] K. Son, d. Moreno, J. Hays, and D. B. Cooper. Solving small-piece jigsaw puzzles by growing consensus. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. 2, 3, 8

[28] R. Tybon. *Generating Solutions to the Jigsaw Puzzle Problem*. PhD thesis, Griffith University, 2004. 1

[29] R. Tybon and D. Kerr. Automated solutions to incomplete jigsaw puzzles. *Artificial Intelligence Review*, 32(1-4):77–99, 2009. 2

[30] R. W. Webster, P. S. LaFollette, and R. L. Stafford. Isthmus critical points for solving jigsaw puzzles in computer vision. *IEEE transactions on systems, man, and cybernetics*, 21(5):1271–1278, 1991. 1

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