Solving Square Jigsaw Puzzles with Loop Constraints

Kilho Son, James Hays, and David B. Cooper

Brown University

Abstract. We present a novel algorithm based on "loop constraints" for assembling non-overlapping square-piece jigsaw puzzles where the rotation and the position of each piece are unknown. Our algorithm finds small loops of puzzle pieces which form consistent cycles. These small loops are in turn aggregated into higher order "loops of loops" in a bottom-up fashion. In contrast to previous puzzle solvers which avoid or ignore puzzle cycles, we specifically seek out and exploit these loops as a form of outlier rejection. Our algorithm significantly outperforms state-of-the-art algorithms in puzzle reconstruction accuracy. For the most challenging type of image puzzles with unknown piece rotation we reduce the reconstruction error by up to 70%. We determine an upper bound on reconstruction accuracy for various data sets and show that, in some cases, our algorithm nearly matches the upper bound.

Keywords: Square Jigsaw Puzzles, Loop Constraints.

1 Introduction

Puzzle assembly problems have aroused people's intellectual interests for centuries and are also vital tasks in fields such as archeology [18]. The most significant physical remnants of numerous past societies are the pots they leave behind, but sometimes meaningful historical information is only accessible when the original complete shape is reconstructed from scattered pieces. Computational puzzle solvers assist researchers with not only configuring pots from their fragments [16] but also reconstructing shredded documents or photographs [21,11]. Puzzle solvers may also prove useful in computer forensics, where deleted block-based image data (e.g. JPEG) is difficult to recognize and organize [9]. Puzzle solvers are also used in image synthesis and manipulation by allowing scenes to be seamlessly rearranged while preserving the original content [3].

This paper proposes a computational puzzle solver for non-overlapping square-piece jigsaw puzzles. Many prior puzzle assembly algorithms [4,13,19,6,15] assume that the orientation of each puzzle piece is known and only the location of each piece is unknown. These are called "Type 1" puzzles. More difficult are "Type 2" puzzles where the orientation of each piece is also unknown [8]. Our algorithm assembles both Type 1 and Type 2 puzzles with no anchor points and no information about the dimensions of the puzzles. Our system is also capable of simultaneously reconstructing multiple puzzles whose pieces are mixed together [18].

The most challenging aspect of puzzle reconstruction is the number of successive local matching decisions that must be correct to achieve an accurate reconstruction. For example, given a puzzle with 432 pieces, an algorithm needs to return 431 true positive

D. Fleet et al. (Eds.): ECCV 2014, Part VI, LNCS 8694, pp. 32–46, 2014.

matches with no false positive matches. Even if the pairwise matches can be found with 0.99 precision (rate of true matches in positive matches), the likelihood that a naive algorithm will chain together 431 true matches is only 0.013. Our method focuses not on a dissimilarity metric between pairs of pieces but on a strategy to recover the complete shape from fragments given a dissimilarity metric. Thus, our method can be extended to the various types of puzzle problems without significant modification. The key idea in our puzzle solver, which stands in contrast to previous strategies, is to explicitly find all small loops or cycles of pieces and in turn group these small loops into higher order "loops of loops" in a bottom-up fashion. Our method uses these puzzle loops, specifically 4-cycles, as a form of outlier rejection whereas previous methods, e.g. Gallagher [8], treat cycles as a nuisance and avoid them by constructing cycle-free trees of puzzle pieces. During the bottom-up assembly, some of the small loops discovered may be spurious. Our algorithm then proceeds top-down to merge unused loop assemblies onto the dominant structures if there is no geometric conflict. Otherwise, the loop assemblies are broken into sub-loops and the merging is attempted again with smaller loops. If the loops of 4 pieces still geometrically conflict with the dominant structures, we remove them as an another form of outlier rejection. We test our method with various Type 1 and Type 2 puzzle datasets to verify that our solver outperforms state-of-the-art methods [4,13,19,6,8,15]. Our contributions are summarized below:

- We propose a conceptually simple bottom-up assembly strategy which operates on top of any existing metric for piece similarity. Our method requires no random field formulations with complex inference procedures, no learning, and no tree construction over a graph of puzzle pieces.
- The proposed square puzzle solver approaches precision 1 (perfect reconstructions) given dissimilarity metrics used in prior literature. We empirically show that the precision of pair matches is likely to increase as pieces (or small loops of pieces) are assembled into higher order loops and reaches 1. Specifically, when our solver is able to construct loop assemblies of puzzle pieces above a *dimension*[1] of 4 or 5, the configurations are always correct – piece pair matches in the assemblies are all true positives. Details are discussed in Section 3.
- Our solver significantly outperforms state-of-the-art methods [4,13,19,6,8,15] with the standard data sets [4,12,13]. For the more challenging Type 2 puzzle setup, we reduce the error rate by up to 70% from the most accurate prior work [8]. In fact, we show that our algorithm is approaching the theoretical upper bound in reconstruction accuracy on the data set from Cho *et al.* [4].
- We evaluate the robustness of Type 2 puzzle solving strategies to image noise (iid Gaussian noise). At high noise levels, the performance gap between our work and previous methods [8] is even more pronounced.

1.1 Related Work

Initiated by Freeman and Gardner [7], the puzzle solving problem has been addressed numerous times in the literature. An overview of puzzle tasks and strategies is well described in [8]. Our puzzle solver is in a line with recent works [2,4,13,19,6,8,15] which

[1] For example, a loop assembly of dimension (order) 2 is a block of 2 by 2 pieces.

solve non-overlapping square-piece jigsaw puzzles. Even though Demaine *et al.* [5] discover that puzzle assembly is an NP-hard problem if the dissimilarity metric is unreliable, the literature has seen empirical performance increases in Type 1 puzzles by using better compatibility metrics and proposing novel assembly strategies such as greedy methods [13], particle filtering [19], genetic algorithms [15], and Markov Random Field formulations solved by belief propagation [4]. The most closely related prior work from Gallagher [8] defines and solves a new type (Type 2) of non-overlapping square-piece jigsaw puzzle with unknown dimension, unknown piece rotation, and unknown piece position. Gallagher [8] also proposes a new piece dissimilarity metric based on the Mahalanobis distance between RGB gradients on the shared edge. They pose the puzzle problem as minimum spanning tree problem constrained by geometric consistency such as a non-overlap between pieces and solve it by modifying Kruskal's Algorithm [10]. Even though we argue for a puzzle assembly strategy which is effectively opposite to Gallagher [8], because we try to leverage puzzle cycles early and they try to avoid them, their proposed distance metric is a huge improvement on the previous works and their method performs well.

Possibly related to our work at a high level are the "Loop Constraints" widely used to estimate extrinsic parameters of cameras [17,20]. With prior knowledge that the pose of cameras forms a loop, these constraints increase accuracy of camera pose estimation by reducing accumulated error around the loop. Multiple 3D scenes are registered by optimizing over the graph of neighboring views [14]. They optimize the global cost function by decomposing the graph into a set of cycle which can be solved in closed form. The "Best Buddies" strategy [13] that prefers pair matches when each piece in the pair independently believes that the other is its most likely match can be considered as a type of loop. Our loop constraints are different from Pomeranz *et al.* [13] because we exploit a higher-order fundamental unit of 4 pieces that agree on 4 boundaries rather than 2 pieces that agree on 1 boundary. This higher ratio of boundaries to pieces gives us more information to constrain our matching decisions. Our algorithm also proceeds from coarse to fine, so that higher-order loops can reject spurious smaller loops.

2 Square Jigsaw Puzzle Solver with Loop Constraints

We explain our puzzle assembly strategy in the case of Type 2 puzzles – non-overlapping square pieces with unknown rotation, position, and unknown puzzle dimensions. Later we will also quantify performance for the simpler Type 1 case with known piece rotation in Section 4.

The main contribution of our work is a novel, loop-based strategy to reconstruct puzzles from the local matching candidates. Using small loops (4-cycles) as the fundamental unit for reconstruction is advantageous over alternative methods in which candidates are chained together without considering cycles (or explicitly avoiding cycles) because the loop constraint is a form of outlier rejection. A loop of potential matches indicates a consensus among the pairwise distances. While it is easy to find 4 pieces that chain together across 3 of their edges with low error, it is unlikely that the fourth edge completing the cycle would also be among the candidate matches by coincidence. In fact, to build a small loop which is not made of true positive matches, *at least* two of the

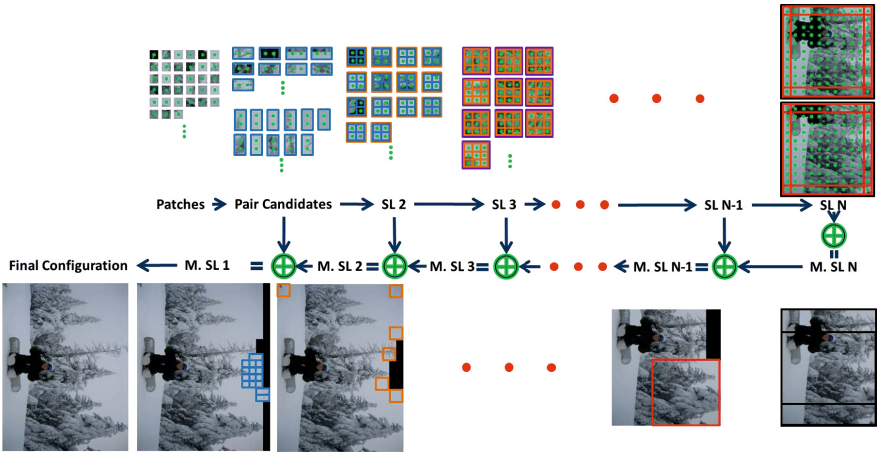


Fig. 1. We discover small loops of increasing dimension from a collection of pieces. The first small loops, made from four pieces, are of dimension 2 (SL 2). We then iteratively build higher-order, larger dimension loops from the lower order loops, e.g. 4 SL 2s are assembled into SL 3s, and so on. We continue assembling loops of loops in a bottom-up fashion until the algorithm finds some maximum sized small loop(s) (SL N). The algorithm then proceeds top-down to merge unused loop assemblies onto the dominant structures (Details are in Section 2.2). This top-down merging is more permissive than the bottom-up assembly and has no "loop" constraint. As long as two small loops share at least two pieces in consistent position they are merged. At a particular level, remaining loops are considered for merging in order of priority, where priority is determined by the mean dissimilarity value of all pair matches in that loop. After all possible merges are done for loops of a particular dimension, the unused loops are broken into their sub-loops and the merging is attempted again with smaller loops. If the reconstruction is not rectangular, trimming and filling steps are required.

edge matches must be wrong. While some small loops will contain incorrect matches that none-the-less lead to a consistent loop, the likelihood of this decreases as higher-order loops are assembled. A small loop of dimension 3, built from 4 small loops of dimension 2, reflects the consensus among many pairwise distance measurements. This is also analyzed further in Section 3.

We use the term "small loop" to emphasize that our method focuses on the shortest possible cycles of pieces at each stage – loops of length 4. Longer loops could be considered, but in this study we use only small loops because (1) longer loops are less likely to be made of entirely true positive pairwise matches and (2) the space of possible cycles increases exponentially with the length of the cycle. While it is possible for us to enumerate all 4-cycles built from candidate pairwise matches in a puzzle, this becomes intractable with longer cycles. (3) Our algorithm is coarse-to-fine, so larger scale cycles are already discovered by finding higher order assemblies of small loops.

2.1 Local, Pairwise Matching

Before assembling loops we must consider the pairwise similarity metric which defines piece compatibility and a strategy for finding candidate pair matches. In Type 2 puzzles, two square puzzle pieces can be aligned in 16 different ways. We calculate dissimilarities between all pairs of pieces for all 16 configurations using the Sum of Squared Distances (SSD) in LAB color space and Mahalanobis Gradient Compatibility (MGC) from Cho *et al.* [4] and Gallagher [8] respectively. Absolute distances between potential piece matches are not comparable (e.g. sky pieces will always produce smaller distances), so we follow the lead of Gallagher [8] and use *dissimilarity ratios* instead. For each edge of each piece, we divide all distances by the smallest matching distance. Unless otherwise stated, edge matches with a dissimilarity ratio less than 1.07 are considered "candidate" matches for further consideration.[2] We limit the maximum number of candidate matches that one side of a piece can have to ζ (typically 10) for computational efficiency.

2.2 Puzzle Assembly with Loop Constraints

We introduce our representation of Type 2 puzzles and operations. We formulate the two major steps of our puzzle assembly: bottom-up recovery of multi-order (i.e. arbitrarily large) small loops in the puzzle and top-down merging of groups of pieces. A visual overview of the assembly process is described in Figure 1.

Formal Representation of Puzzle and Puzzle Operations. Pieces are represented by complex numbers where real and imaginary parts are IDs of pieces and their rotation, respectively. The real parts of the complex numbers range from 1 to the total number of pieces and have unique numbers. The imaginary parts of the complex numbers are $\{0, 1, 2, 3\}$ and represent the counter-clockwise number of rotations, each of 90 degree for the pieces (For Type 1 puzzles, the pieces have no imaginary component to their representation). We pose the puzzle problem as arranging 2D sets of complex numbers. The final result of the puzzle solver is 2D complex-valued matrix. To generally represent configurations in matrices, we also allow 'NaN' which means that no piece occupies the position. We define a rotational transformation function $Rot_n(.)$ where an input is a 2D complex-valued matrix. The function $Rot_n(.)$ geometrically turns the input matrix n times in a counter-clockwise direction. In other words, the individual pieces are rotated by $90 \times n$ degrees in a counter-clockwise direction and the entire matrix turns counter-clockwise (See Figure 2).

Relational symbols are defined given complex-valued matrices U and V. If matrices U and V share at least two of the same ID pieces, they are aligned by one of the shared pieces. If there is no a geometric conflict, such as a overlap with different complex

[2] Using dissimilarity ratios makes edge distances more comparable but it has the downside that the first-nearest-neighbor distance is exactly 1 for every edge and thus they are no longer comparable. To alleviate this, the smallest dissimilarity ratio is substituted with the reciprocal of the second smallest dissimilarity ratio. This is only relevant for experiments where the candidate threshold is lowered below 1.

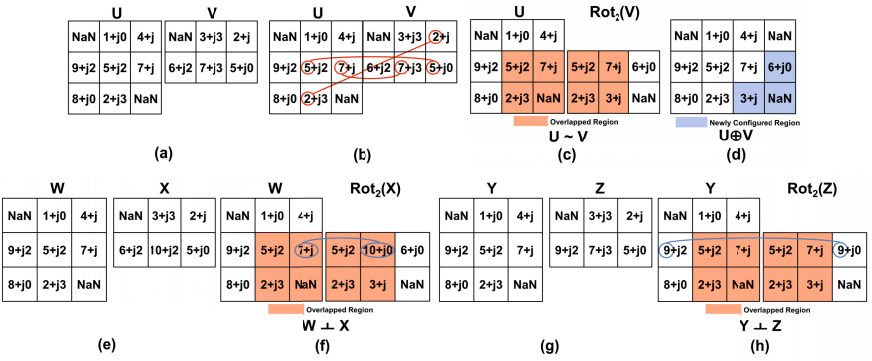


Fig. 2. An example representation of groups of pieces using complex-valued matrices and their relational operations. The top row shows two matrices U and V which are compatible for a merge, and the bottom row shows variations of matrices which are instead incompatible. (a) Given complex-valued matrices U and V, (b) which share multiple pieces with the same IDs (real parts of the complex numbers), (c) we rotate matrix V ($Rot_2(V)$) to align the shared pieces. (d) If the shared region is consistent to each matrix, we merge the two matrices. (e) Given complex-valued matrices W and X, (f) we align them by shared pieces. However, the matrices W and X are in conflict because the overlapped region includes different complex numbers (different IDs or rotations). (g,h) The matrices Y and Z also conflict because the non-overlapped regions include the same ID pieces (real parts of the complex numbers) in both matrices.

numbers (ID or rotation) or an existence of same IDs (real part of a complex number) in a non-shared region, the matrices U and V are geometrically consistent, represented by $U \sim V$. Otherwise, geometric inconsistency is represented by $U \perp V$. If the matrices U and V are geometrically consistent, we can merge the two matrices $U \oplus V$ (See Figure 2). If less than two of the same ID pieces are in both matrices, we assume that they are not related with each other $U||V$.[3]

Recovering Small Loops of Arbitrary Order in the Puzzle. In this step we discover small loops with candidate pair matches given by the local matching algorithm described in Section 2.1. In the first iteration, small loops of width 2 (SL 2) are formed from four candidate matches. Once all consistent loops are discovered, these loops (e.g. SL 2) are assembled into higher-order 4 cycles (e.g. SL 3) if piece locations are geometrically consistent (piece index and rotation are the same) among all 4 lower-order loops. The algorithm iteratively recovers SL i loops by assembling SL i-1 elements. The procedure continues until no higher-order loops can be built and some maximum size small loop is found (SL N).

[3] A single piece correspondence is enough to establish the geometric relationship between two groups of pieces, but we choose not merge on the basis of singe piece correspondences. Such correspondences are more likely to be spurious. However, if they are true correspondences it is likely that as the pieces grow they will eventually have two or more shared pieces and thus become merge-able by our criteria.

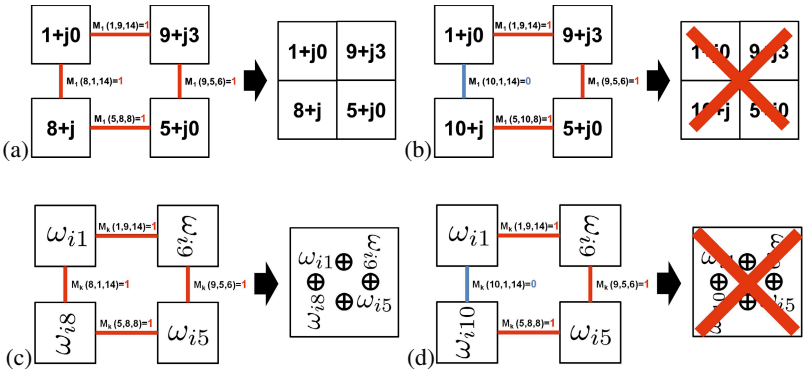


Fig. 3. *Recovering small loops of arbitrary order*. The match candidates from the local, pairwise matching step (Section 2.1) are represented in a sparse 3D binary matrix M_1 of size $K_1 \times K_1 \times 16$. (a) If M_1 indicates that all the pairs (1-9, 9-5, 5-8 and 8-1) are match candidates, the four pieces form a small loop. We make a new matrix (2×2) with the small loop and save it as an element of the 2nd-order set Ω_2. (b) If one or more pairs are not match candidates (in this case, 10-1), the four pieces do not form a small loop. (c) For the ith-order set Ω_i, matrix M_i indicates the match candidates between elements of Ω_i. M_i is size $K_i \times K_i \times 16$. If M_i indicates that all the pairs ($\omega_{i1} - \omega_{i9}, \omega_{i9} - \omega_{i5}, \omega_{i5} - \omega_{i8}$ and $\omega_{i8} - \omega_{i1}$) are match candidates the four groups of pieces form a ith-order small loop. We make a new matrix ($(i+1) \times (i+1)$) by merging the four groups of pieces and save it as an element of the $(i+1)$th-order set Ω_{i+1}. (d) If one or more pairs are not match candidates (in this case, $\omega_{i10} - \omega_{i1}$), the four groups of pieces do not form the ith-order small loop.

More formally, the input puzzle is represented by a set of complex numbers $\Omega_1 = \{\omega_{11}, \omega_{12}, \ldots \omega_{1K_1}\}$ where K_1 is a total number of pieces. Match candidates from the local, pairwise matching step (Section 2.1) are stored in a 3D binary matrix M_1 of size $K_1 \times K_1 \times 16$. If $M_1(x, y, z)$ is 1, the piece ID x and the piece ID y are a match candidate with the configuration z. The piece ID y turns $\lfloor \frac{z-1}{4} \rfloor + 1$ times counter-clockwise and places above, to the right, below or to the left of the piece ID x according to $(z-1) \bmod 4 + 1$.

In the same way, we search for all small loops with combinations of 4 pairs of match candidates given Ω_1 and the binary match candidate matrix M_1 (See Figure 3). All the small loops are saved as elements of the 2nd-order set Ω_2.

Iteratively, given the i th-order set $\Omega_i = \{\omega_{i1}, \omega_{i2}, \ldots \omega_{iK_i}\}$ where the size of the each element (ω_{ix}) is $i \times i$, we generate a 3D binary matrix M_i of size $K_i \times K_i \times 16$. M_i stores match candidates between the elements in the i th-order set Ω_i. If the size of the overlap between the elements ω_{ix} and ω_{iy} is $i \times (i-1)$ or $(i-1) \times i$ and there is no geometric conflict ($\omega_{ix} \sim \omega_{iy}$), we set $M_i(x, y, z)$ as 1, otherwise it is 0 where the number z encodes the relative rotation and position of the two matrices. Iteratively, we find small loops of dimension i+1 from elements in the Ω_i with the matrix M_i and save them as elements of the (i+1) th-order set Ω_{i+1} (See Figure 3 (b)). We find a maximum-order of set (Ω_N) until the set is not a null set.

Algorithm 1: Merge matrices in the set Ω_N

Data: Ω_N
Result: Λ_N

$\Lambda_N = \Omega_N$
while *Two or more elements are overlapped between two matrices in the set* Λ_N **do**
  ω_{Nj} and ω_{Nk} are the overlapped pairs with the highest priority among them.
  $\Lambda_N = \Lambda_N \backslash \{\omega_{Nj}, \omega_{Nk}\}$; $\Lambda_N = \Lambda_N \cup f_m(\omega_{Nj}, \omega_{Nk})$;
end
return Λ_N

Merging Groups of Pieces. At this point the algorithm proceeds in a top-down fashion and merges successively smaller remaining small loops without enforcing loop constraints. Merges are performed when two piece assemblies overlap by two or more pieces in geometrically consistent configuration. If a small loop conflicts geometrically with a larger assembly, the small loop is broken into its constituent lower order loops. If two small loops of the same dimension conflict, the loop with the smaller mean value of dissimilarity among its edge matches is used. If the assembly of pieces is not a rectangular shape, the algorithm estimates the dimension of the configuration given the number of pieces and the current configuration. Based on the estimated dimension, the method configures the final shape of the puzzle by trimming (so as to break minimum dimension of small loops in the biggest configuration) and filling.

More formally, we define a merge function given two matrices ω_x, ω_y below

$$f_m(\omega_x, \omega_y) = \begin{cases} \omega_x \oplus \omega_y, & \text{if } \omega_x \sim \omega_y \\ \omega_x, & \text{if } \omega_x \perp \omega_y \wedge f_p(\omega_x) \geq f_p(\omega_y) \\ \omega_y, & \text{if } \omega_x \perp \omega_y \wedge f_p(\omega_x) < f_p(\omega_y) \\ \omega_x, \omega_y, & \text{if } \omega_x || \omega_y \end{cases} \tag{1}$$

where the priority function $f_p(\omega_x, \omega_y)$ is defined below

$$\begin{cases} f_p(\omega_x) \geq f_p(\omega_y), & \text{if } \#(\omega_x) > \#(\omega_y) \\ f_p(\omega_x) \geq f_p(\omega_y), & \text{if } \#(\omega_x) = \#(\omega_y) \wedge \bar{\omega_x} < \bar{\omega_y} \\ f_p(\omega_x) < f_p(\omega_y), & \text{Otherwise} \end{cases} \tag{2}$$

where $\#(.)$ is a number of elements in the matrix (except NaN) and $\bar{\omega_x}$ is a mean value of the dissimilarity metrics of all pairs in the group ω_x.

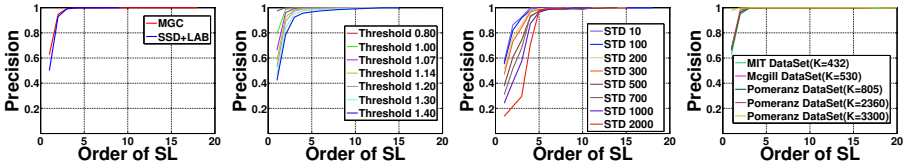
Given the sets of sets $\{\Omega_1, \Omega_2, \Omega_3, \ldots \Omega_N\}$ from the previous step, the method begins with performing the Algorithm 1 to generate a new set Λ_N from the Ω_N. The Λ_N is a set of matrices which are results of merging the matrices in the Ω_N. We performs the Algorithm 2 iteratively to generate a Λ_{i-1} from the Λ_i and Ω_{i-1} until we generate a Λ_2. The Λ_{i-1} is a set of output matrices by merging the matrices in the Λ_i and Ω_{i-1}. We generate Λ_1 by merging remaining pair matching candidates to the set Λ_2.

Elements of a set (Λ_1) of 2D matrices are either a complex number or NaN. The 2D matrices in a set Λ_1 are final configurations by loop constraints. Λ_1 normally contains

Algorithm 2: Merge matrices in the sets Λ_i and Ω_{i-1}

Data: Λ_i and Ω_{i-1}
Result: Λ_{i-1}

$\Lambda_{i-1} = \{\Lambda_i \cup \Omega_{i-1}\} = \{s_1, s_2, \ldots\}$
while *Two or more elements are overlapped between two matrices in the set* Λ_{i-1} **do**
  s_j and s_k are the overlapped pairs with the highest priority among them.
  $\Lambda_{i-1} = \Lambda_{i-1} \backslash \{s_j, s_k\}$; $\Lambda_{i-1} = \Lambda_{i-1} \cup f_m(s_j, s_k)$;
end
return Λ_{i-1}



(a) (b) (c) (d)

Fig. 4. The average precision of pair matches changes as a function of small loop dimension (order of small loops) (a) for different distance metrics (with the MIT dataset), (b) at different local matching thresholds (with the MIT dataset), (c) at different noise levels (with the MIT dataset), and (d) with different datasets (with the MGC metric). The leftmost point, SL1, represents the performance of the local matching by itself with no loop constraints. The noise levels in (c) are high magnitude because the input images are 16 bit. The patch size is $P = 28$ for all the experiments.

a single 2D large matrix, which is a main configuration from the puzzle. If Λ_1 contains multiple matrices, we consider a matrix that contains complex numbers maximally as a main configuration and break the other configurations into pieces[4]. If the main matrix (configuration) contains NaN it means that the final configuration of the puzzle is not rectangular. In this case we estimate most probable dimension of the puzzle given the current main configuration and the total number of pieces. With the estimated dimension of the puzzle, the method trims the main configuration so that it cuts minimum order of small loops in the main configuration. This is because higher order of small loops are more reliable than smaller one. We fill the NaN with the remaining pieces in the order of minimum total dissimilarity score across all neighbors.

3 Empirical Verification of Small Loop Strategy

We observe in Figure 4 that as higher-order small loops are built from smaller dimension loops the precision of pair matches increases significantly even in the presence of noise with dataset from Cho *et al.* [4] (MIT dataset). In figure 4 (a), for the MGC and SSD+LAB distance metrics, the precisions are 0.627 and 0.5 for local, pairwise edge

[4] If we search multiple configurations from mixed puzzles, we consider all matrices in Λ_1 as main configurations.

matches (see order 1 of SL) and jumps dramatically to 0.947 and 0.929 (see order 2 of SL) with our method with the lowest dimension small loop. The precision keeps increasing as order of small loops grows eventually reaching 1 for both metrics (although the recall has dropped considerably by this point). This tendency of the precision to reach 1 as higher order small loops are assembled persists even when the threshold for candidate matches is varied (Figure 4 (b)) and when noise (pixel-wise Gaussian) is added to the puzzle pieces (Figure 4 (c)). With the various datasets, the precision also reaches to 1 as the order of small loops increases (Figure 4 (d)). For most of the experiments, precision approaches 1 when the order of small loops is above 4 or 5. Notably, although the precision of pair matches is below 15% under severe noise (2000 STD), it increases significantly and reaches to 1 as order of small loops grows.

4 Experiments

We verify our square jigsaw puzzle assembly algorithm with sets of images from Cho *et al.* [4] (MIT dataset), Olmos *et al.* [12] (Mcgill dataset) and Pomeranz *et al.* [13]. Each paper provides a set of 20 images and Pomeranz *et al.* [13] additionally presents 2 sets of 3 large images. All the data sets are widely used as a benchmark to measure the performance of square jigsaw puzzle solvers. For some images, the camera is perfectly aligned with the horizon line and image edges (e.g. building boundaries) align exactly with puzzle edges. Some patches contain insufficient information (homogeneous region such as sky, water and snow) and others present repetitive texture (man-made textures and windows). As a result, the pairwise dissimilarity metrics return many false positives and false negatives on these data sets.

We measure performance using metrics from Cho *et al.* [4] and Gallagher [8]. **"Direct Comparison"** measures a percentage of pieces that are positioned absolutely correctly. **"Neighbor Comparison"** is a percentage of correct neighbor pairs and **"Largest Component"** is a percentage of image area occupied by the largest group of correctly configured pieces. **"Perfect Reconstruction"** is a binary indicator of whether or not all pieces are correctly positioned with correct rotation.

For many experiments we also report an "upper bound" on performance. Particular puzzles may be impossible to unambiguously reconstruct because certain pieces are identical, as a result of camera saturation. The upper bound we report is the accuracy achieved by correctly placing every piece that is not completely saturated.

Type 1 Puzzles (known orientation and unknown position): We test on 20 puzzles of 432 pieces each ($K = 432$) with the piece size of 28×28 pixels (P=28) from the MIT dataset. We use MGC as a dissimilarity metric for solving Type 1 puzzles. Table 1 reports our performance for solving Type 1 puzzles. The proposed algorithm outperforms prior works [4,19,13,6,8,15]. Our improvement is especially noteworthy because our algorithm recovers the dimension of the puzzles rather than requiring it as an input as in [4,19,13,6,15]. Our performance is very near the upper bound of the MIT dataset for Type 1 puzzles.

We also test our method with 20 puzzles ($K = 540, P = 28$) from the Mcgill dataset and 26 puzzles (20 ($K = 805, P = 28$), 3 ($K = 2360, P = 28$) and 3 ($K = 3300, P = 28$)) from Pomeranz *et al.* [13] (See Table 2). Notably, for the puzzles

Table 1. Reconstruction performance on Type 1 puzzles from the MIT dataset, The number of pieces is $K = 432$ and the size of each piece is $P = 28$ pixels

	Direct	Nei.	Comp.	Perfect
Cho *et al.* [4]	10%	55%	-	0
Yang *et al.* [19]	69.2%	86.2%	-	-
Pomeranz *et al.* [13]	94%	95%	-	**13**
Andalo *et al.* [6]	91.7%	94.3%	-	-
Gallagher *et al.* [8]	95.3%	95.1%	95.3%	12
Sholomon *et al.* [15]	80.6%	95.2%	-	-
Proposed	**95.6%**	**95.5%**	**95.5%**	**13**
Upper Bound	96.7%	96.4%	96.6%	15

Table 2. Reconstruction performance on Type 1 puzzles from Olmos *et al.* [12] and Pomeranz *et al.* [13]. The size of each piece is $P = 28$ pixels.

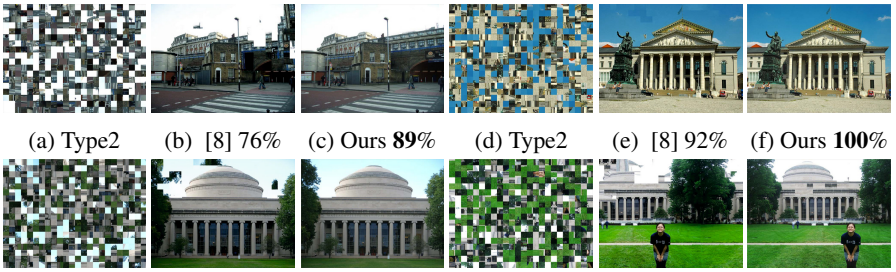
	540Pieces [12]		805Pieces [13]		2360Pieces [13]		3300Pieces [13]	
	Direct	Nei.	Direct	Nei.	Direct	Nei.	Direct	Nei.
Pomeranz [13]	83%	91%	80%	90%	33.4%	84.7%	80.7%	85.0%
Andalo [6]	90.6%	**95.3%**	82.5%	93.4%	-	-	-	-
Sholomon [15]	90.6%	94.7%	92.8%	**95.4%**	82.7%	87.5%	65.4%	91.9%
Proposed	**92.2%**	95.2%	**93.1%**	94.9%	**94.4%**	**96.4%**	**92.0%**	**96.4%**

with large numbers of pieces ($K = 2360, K = 3300$), our method improves the reconstruction performance significantly (more than 10%) under both Direct and Neighbor Comparison. As the number of puzzle pieces increases, our algorithm has the opportunity to discover even higher order loops of pieces which tend to be high precision.

Type 2 Puzzles (unknown orientation and position): Type 2 puzzles are a challenging extension of Type 1 puzzles. With K pieces, Type 2 puzzles have 4^K times as many possible configurations as Type 1 puzzles. For small puzzles with $K = 432$ this means that Type 2 puzzles have $4^{432} \approx 1.23 \times 10^{26}$ times as many solutions. Due to this increased complexity, there is still room for improvement in Type 2 puzzles solving accuracy whereas performance on Type 1 puzzles is nearly saturated by our algorithm and previous methods.

With Type 2 puzzles ($K = 432, P = 28$) from the MIT dataset, we examine our proposed method with the sum of squared distance (SSD) in LAB color space and MGC as metrics and compare with Gallagher [8] (Table 3). Given the same dissimilarity metric, our method increases the performance by 12% under the Direct Comparison, thus reducing the error rate by up to 70%. Because both methods use the same local distance metric, this difference is entirely due to loop assembly strategy versus the tree-based algorithm in Gallagher [8] (See visual comparisons of results in Figure 5).

We compare our method with Gallagher [8] with different piece sizes (P=14 and 28) and numbers of pieces ($K = 221, 432$ and 1064) on puzzles from the MIT dataset. We use MGC as a dissimilarity metric from now on unless otherwise stated. In all cases, our



(a) Type2 (b) [8] 76% (c) Ours **89%** (d) Type2 (e) [8] 92% (f) Ours **100%**

(g) Type2 (h) [8] 82% (i) Ours **99%** (j) Type2 (k) [8] 0% (l) Ours **84%**

Fig. 5. Visual comparisons of the results on Type 2 puzzles ($P = 28, K = 432$). The percentage numbers indicate Direct Comparisons.

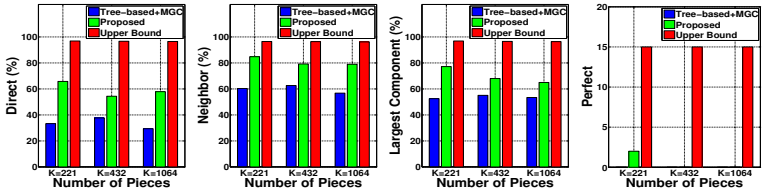
Table 3. Reconstruction performance on Type 2 puzzles ($P = 28, K = 432$) from the MIT dataset

	Direct	Nei.	Comp.	Perfect
Tree-based+L.SSD [8]	42.3%	68.2%	63.6%	1
S.L.+L.SSD (Proposed)	54.3%	79.7%	66.3%	2
Tree-based+MGC [8]	82.2%	90.4%	88.9%	9
S.L.+MGC (Proposed)	**94.7%**	**94.9%**	**94.6%**	**12**
Upper Bound	96.7%	96.4%	96.6%	15

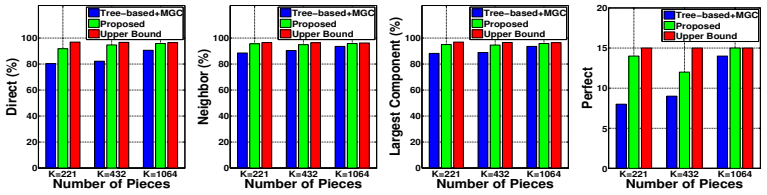
method outperforms Gallagher's algorithm [8] (See Figure 6 (a) and (b)). Notably, we almost achieve the upper bound of the performance in the case $K = 1064, P = 28$. Our method is verified with more puzzles ($K = 550$, 805, 2260 and 3300) from the Mcgill dataset and Pomeranz *et al.* [13] (Figure 6 (c)). Our puzzle solver distinctively outperforms Gallagher [8] when the number of puzzle pieces increases ($K = 2260, 3300$).

Noise Analysis on Type 2 Puzzles: We further analyze the robustness of our puzzle solver by adding pixel-wise Gaussian noise to the MIT dataset. Experiments are conducted 5 times ($P = 28$, $K = 432$) and the performance values are averaged. Figure 7 shows that our method tends to outperform Gallagher [8] as noise increases (26% improvement in 2000 STD Gaussian noise under Neighbor Comparison). As pixel-wise Gaussian noise increases in the pieces, the dissimilarity metrics are no longer reliable. The constrained Kruskal's algorithm in [8] has a strong implicit belief in dissimilarity metrics so performance decreases considerably as noise increases. Our method, however, is more robust to spurious pairwise comparisons because loops require consensus between many dissimilarity measurements and thus avoid many false pairwise matches.

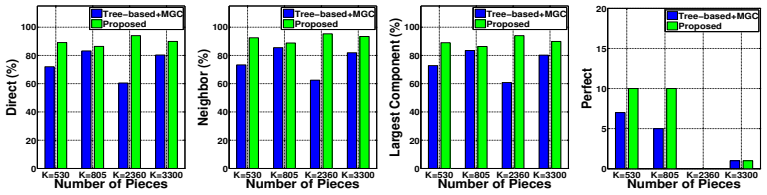
Extra Type 2 Puzzles: As opposed to prior works [4,19,13,6,15], our method does not require the dimension of resulting puzzles as an input. This allows us to solve puzzles from multiple images with no information except the pieces themselves. Our solver perfectly assembles 1900 mixed Type 2 puzzle pieces from [1] (See Figure 8).



(a) Performance comparison with puzzles from the MIT dataset. Size of patch is $P = 14$.



(b) Performance comparison with puzzles from the MIT dataset. Size of patch is $P = 28$.



(c) Performance comparison with puzzles from the Mcgill dataset and Pomeranz *et al.* [13]. Size of patch is $P = 28$.

Fig. 6. Performance comparison between ours and Tree-based MGC [8] on Type 2 with various cases. A table is presented in a supplemental material.

As observed in the previous experiments, our solver significantly outperforms prior works especially as a number of puzzle pieces increases in both Type 1 and Type 2 puzzles. This is because the opportunity to recover high order small loops (above 4 or 5 orders) increases. (the precision approaches to 1 if the order of small loops is above 4 or 5.) Big images from [1] are used for more intensive experiments with large numbers of puzzle pieces. Our solver configures 9801 and 4108 piece Type 2 puzzles perfectly (See Figure 8). We believe that these are the largest puzzles to date that are perfectly reconstructed with unknown orientation and position (Type 2).

The complexity for searching all small loops (4-cycles) is $O(\zeta^3 * N_p)$, where ζ is the maximum number of positive pair matches that one side of a piece can have and N_p is a number of pair matching candidates. ζ is normally from 1 to 3 and maximally 10 in our experiments and each operation is just an indexation of a binary matrix. The average time for finding all small loops is 0.308 second with the MIT dataset (432-piece Type 2 puzzles) in Matlab. Most of the time is spent in pairwise matching, unoptimized merging, trimming and filling steps. Using MGC, our algorithm spends 140 seconds for 432 pieces and 25.6 hours for 9801 pieces (Type 2) on a modern PC.

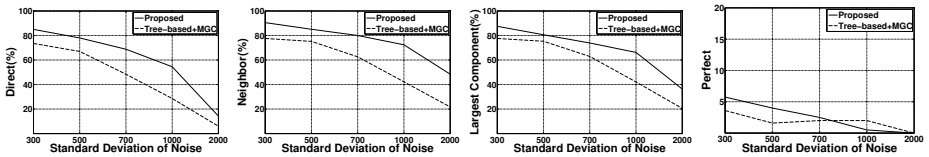
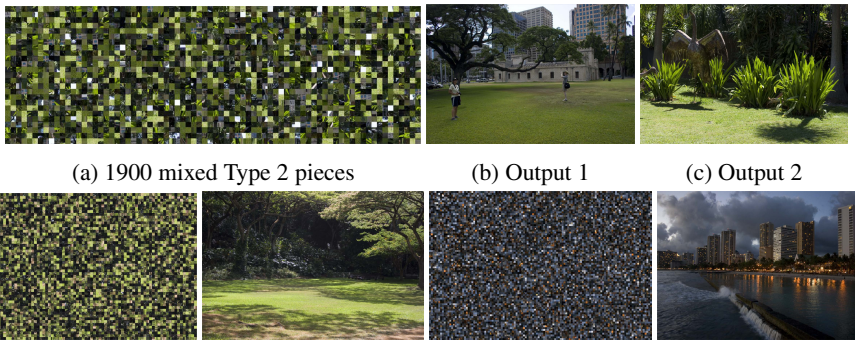


Fig. 7. Performance comparison in the presence of noise. Experiments are conducted 5 times ($P = 28$,$K = 432$) and the performance values are averaged. Our method outperforms Gallagher [8], especially as noise increases.



(a) 1900 mixed Type 2 pieces (b) Output 1 (c) Output 2

(d) 4108 Type 2 pieces (e) Output (f) 9801 Type 2 pieces (g) Output

Fig. 8. Reconstructions on mixed Type 2 puzzles and very large Type 2 puzzles (P = 28)

5 Conclusion

We propose a non-overlapping square-piece jigsaw puzzle solver based on loop constraints. Our algorithm seeks out and exploits loops as a form of outlier rejection. The proposed square-piece jigsaw puzzle solver approaches precision 1 given existing dissimilarity metrics. As a result, our method outperforms the state of the art on standard benchmarks. The performance is even better when the number of puzzle pieces increases. We perfectly reconstruct what we believe to be the largest Type 2 puzzles to date (9801 pieces). Our algorithm outperforms prior work even in the presence of considerable image noise.

Acknowledgments. This work was partially supported by NSF Grant # 0808718 to Kilho Son and David B. Cooper, and NSF CAREER award 1149853 to James Hays.

References

1. `http://cdb.paradice-insight.us/`
2. Alajlan, N.: Solving square jigsaw puzzles using dynamic programming and the hungarian procedure. American Journal of Applied Science 5(11), 1941–1947 (2009)
3. Cho, T.S., Avidan, S., Freeman, W.T.: The patch transform. PAMI (2010)

4. Cho, T.S., Avidan, S., Freeman, W.T.: A probabilistic image jigsaw puzzle solver. In: CVPR (2010)
5. Demaine, E.D., Demaine, M.L.: Jigsaw puzzles, edge matching, and polyomino packing: Connections and complexity. Graphs and Combinatorics 23(suppl.) (June 2007)
6. Andal, F.A., Taubin, G., Goldenstein, S.: Solving image puzzles with a simple quadratic programming formulation. In: Conference on Graphics, Patterns and Images (2012)
7. Freeman, H., Garder, L.: Apictorial jigsaw puzzles: the computer solution of a problem in pattern recognition. Electronic Computers 13, 118–127 (1964)
8. Gallagher, A.C.: Jigsaw puzzles with pieces of unknown orientation. In: CVPR (2012)
9. Garfinkel, S.L.: Digital forensics research: The next 10 years. Digit. Investig. 7, S64–S73 (2010), http://dx.doi.org/10.1016/j.diin.2010.05.009
10. Joseph, B., Kruskal, J.: On the shortest spanning subtree of a graph and the traveling salesman problem. American Mathematical Society (1956)
11. Liu, H., Cao, S., Yan, S.: Automated assembly of shredded pieces from multiple photos. In: ICME (2010)
12. Olmos, A., Kingdom, F.A.A.: A biologically inspired algorithm for the recovery of shading and reflectance images (2004)
13. Pomeranz, D., Shemesh, M., Ben-Shahar, O.: A fully automated greedy square jigsaw puzzle solver. In: CVPR (2011)
14. Sharp, G.C., Lee, S.W., Wehe, D.K.: Multiview registration of 3D scenes by minimizing error between coordinate frames. PAMI 26(8), 1037–1050 (2004)
15. Sholomon, D., David, O., Netanyahu, N.: A genetic algorithm-based solver for very large jigsaw puzzles. In: CVPR (2013)
16. Son, K., Almeida, E.B., Cooper, D.B.: Axially symmetric 3D pots configuration system using axis of symmetry and break curve. In: CVPR (2013)
17. Williams, B., Cummins, M., Neira, J., Newman, P., Reid, I., Tardós, J.: A comparison of loop closing techniques in monocular slam. Robotics and Autonomous Systems (2009)
18. Willis, A., Cooper, D.B.: Computational reconstruction of ancient artifacts. IEEE Signal Processing Magazine, 65–83 (2008)
19. Yang, X., Adluru, N., Latecki, L.J.: Particle filter with state permutations for solving image jigsaw puzzles. In: CVPR (2011)
20. Zach, C., Klopschitz, M., Pollefeys, M.: Disambiguating visual relations using loop constraints. In: CVPR (2010)
21. Zhu, L., Zhou, Z., Hu, D.: Globally consistent reconstruction of ripped-up documents. PAMI 30(1), 1–13 (2008)