Clustering-Based, Fully Automated Mixed-Bag Jigsaw Puzzle Solving

Zayd Hammoudeh(✉) and Chris Pollett

Department of Computer Science, San José State University, San José, CA, USA
{zayd.hammoudeh,chris.pollett}@sjsu.edu

Abstract. The jig swap puzzle is a variant of the traditional jigsaw puzzle, wherein all pieces are equal-sized squares that must be placed adjacent to one another to reconstruct an original, unknown image. This paper proposes an agglomerative hierarchical clustering-based solver that can simultaneously reconstruct multiple, mixed jig swap puzzles. Our solver requires no additional information beyond an unordered input bag of puzzle pieces, and it significantly outperforms the current state of the art in terms of both the reconstructed output quality as well the number of input puzzles it supports. In addition, we define the first quality metrics specifically tailored for multi-puzzle solvers, the Enhanced Direct Accuracy Score (EDAS), the Shiftable Enhanced Direct Accuracy Score (SEDAS), and the Enhanced Neighbor Accuracy Score (ENAS).

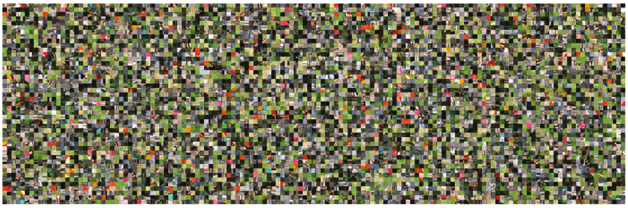
1 Introduction

The first jigsaw puzzle was introduced over 250 years ago. Despite being considered a hobby for children, puzzle solving is strongly NP-complete when inter-piece compatibility is an unreliable metric for determining adjacency [1]. Jigsaw puzzle techniques have been applied to a variety of disciplines including: archaeological artifact reconstruction [8], deleted file analysis [5], image editing [3], shredded document reconstruction [15], and DNA fragment reassembly [9].

Most recent automated puzzle solving research has focused on the jig swap puzzle, which is similar to a traditional jigsaw puzzle except that all pieces are equal-sized squares. This makes them significantly more challenging to solve since piece shape cannot be used. In addition, the original "ground-truth" solution image is generally unknown by the solver.

The jig swap puzzle problem is subclassified into three different categories based on the level of difficulty [4]. The simplest variety is the *Type 1* puzzle, which fixes piece orientation by disallowing their rotation. While the puzzle's image contents are unknown, the overall dimensions are known as well as potentially the correct location of one or more pieces. In contrast, *Type 2* jig swap puzzles allow piece rotation, which for puzzles of n pieces increases the number of possible solutions by a factor of 4^n; the dimensions for this type of puzzle may be unknown. *Mixed-bag* puzzles contain pieces from multiple input images as shown in Fig. 1. Puzzle piece orientation may be provided, but image dimensions are unknown and may vary. Most current mixed-bag solving algorithms require the specification of the number of ground-truth inputs.

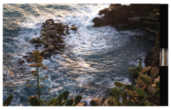
M. Felsberg et al. (Eds.): CAIP 2017, Part II, LNCS 10425, pp. 205–217, 2017.
DOI: 10.1007/978-3-319-64698-5_18



Mixed 6,255 Piece Input

540 Pieces, SEDAS=1

805 Pieces, SEDAS=1



805 Pieces, SEDAS=0.990

805 Pieces, SEDAS=0.990

3,300 Pieces, SEDAS=0.998

Fig. 1. Fully-automated mixed-bag puzzle solving: Our solver generated these results without any external information, including the number of input puzzles. The average, weighted EDAS and ENAS scores were 0.997 and 0.993 respectively.

In 2011, Pomeranz *et al.* developed a greedy, Type 1 jig swap puzzle solver that has been foundational for much of the subsequent research. They introduced the concept of *best buddies*, which are two puzzle piece sides (e.g., left, right, top, bottom) that are mutually more similar to each other than they are to any other piece's side. Pomeranz *et al.* also defined multiple test datasets, some of which are used in this paper.

Paikin and Tal [11] advanced the current state of the art in 2015 with their greedy solver that supports both missing pieces and mixed-bag puzzles. Their approach has two primary limitations. First, the solver must be provided the number of ground-truth inputs. In addition, seed piece selection is based on very localized information (i.e., only 13 pieces), which often results in poor runtime decisions such as multiple puzzles spawning from the same ground-truth input. These suboptimal selections can catastrophically degrade solution quality.

This paper's primary contribution is a novel, clustering-based, mixed-bag puzzle solver that significantly outperforms the state of the art both in terms of solution quality and the number of supportable puzzles. Unlike previous work, our approach requires no externally supplied, "oracle" information including the number of ground-truth inputs.

In addition, previously proposed, single-puzzle-solver performance metrics [2] are unusable for mixed-bag puzzles since they do not account for the presence of pieces from different images in a single output nor for the dispersion of one

Algorithm 1. The Mixed-Bag Solver

```
function MixedBagSolver(pieces)
    segments ← Segmentation(pieces)
    overlap_matrix ← Stitch(segments, pieces)
    clusters ← Cluster(segments, overlap_matrix)
    seeds ← SelectSeeds(clusters)
    solved_puzzles ← FinalAssembly(seeds, pieces)
    return solved_puzzles
```

input's pieces across multiple outputs. As such, we introduce the first quality metrics for mixed-bag puzzles. We also enhance an existing metric to correct for the potential to be misleadingly punitive when puzzle dimensions are unknown.

2 Overview of the Mixed-Bag Solver

Humans commonly solve jigsaw puzzles by correctly assembling subregions and then iteratively merge those smaller reconstructions to form larger ones. This strategy forms the basis of our *Mixed-Bag Solver* shown in Algorithm 1. Its only input is the combined bag of pieces. The number of puzzles, their dimensions, and piece orientation are all unknown.

The first Mixed-Bag Solver stage identifies disjoint sets of pieces (i.e., segments) where there is strong confidence of correct placement. Next, the solver quantifies inter-segment relationships via the stitching process; agglomerative hierarchical clustering uses these quantified similarity scores to group related segments. Each resulting segment cluster represents what the solver identified as a single ground-truth input. A seed piece is selected from each cluster for use in the final assembly stage, which generates the reconstructed puzzle output(s).

Although not shown in Algorithm 1, the Mixed-Bag Solver requires a placer, which organizes (i.e., places) the individual pieces. Our architecture is independent of the specific placer used, granting it significant flexibility. For all experiments in this paper, we used the placer algorithm proposed by Paikin and Tal [11] as it is the current state of the art and due to its multiple puzzle support.

3 Segmentation

Segmentation provides basic structure to the unordered bag of pieces by partitioning it into disjoint, ordered sets, known as *segments*, which are partial puzzle assemblies where there is a high degree of confidence of correct piece placement.

Segmentation is performed across one or more rounds. Initially, pieces have no segment assignment. In each round, all unassigned pieces are assembled together as though they belong to the same input image as shown in Fig. 2; this eliminates the need to make any assumptions regarding the number of input puzzles. Once the pieces have been placed, the single, reconstructed puzzle is segmented as

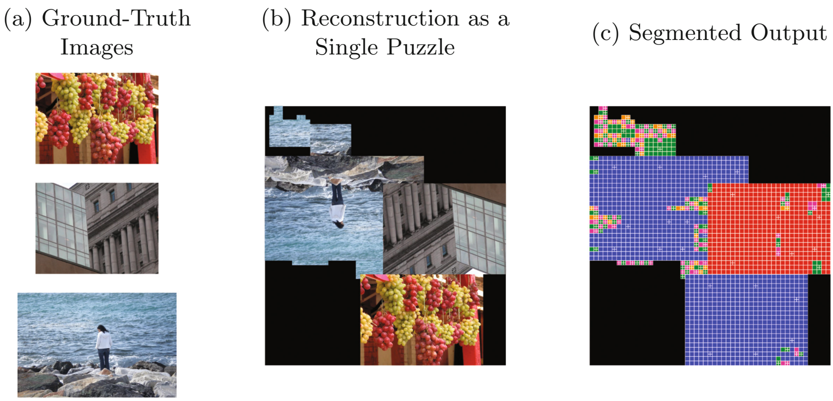


Fig. 2. Segmentation example: Three ground-truth inputs of two different sizes are shown in (a). All pieces are placed in the single, reconstructed output puzzle in (b). Segmented output in (c) is shown with any contiguous group of matching colored pieces belonging to the same segment. Stitching pieces are denoted with a white "+" mark.

described in Algorithm 2, which is partially based on the approach originally proposed by Pomeranz *et al.* in [13].

Segments in the single, reconstructed output are found iteratively, with all pieces eventually assigned to a single segment. Each segment's growth starts by adding one *seed* piece from the *unassigned* pool to an empty queue. Pieces are popped from the queue and added to the current, expanding segment. If the popped piece's neighbor in the reconstructed output is both in *unassigned* and also its best buddy, then that neighbor is added to the queue. A segment's growth terminates once no pieces remain in the queue to be popped.

As mentioned previously, two puzzle pieces, p_i and p_j, are *best buddies* on their respective sides, s_x and s_y, if they mutually more similar to each other than they are to a side, s_z, of any other piece, p_k. Given a metric, C, that quantifies inter-piece similarity, we define the best buddy relationship as:

$$\begin{gathered}\forall p_k \neq p_j \forall s_z, C(p_i, s_x, p_j, s_y) > C(p_i, s_x, p_k, s_z) \\ \text{and} \\ \forall p_k \neq p_i \forall s_z, C(p_j, s_y, p_i, s_x) > C(p_j, s_y, p_k, s_z).\end{gathered} \tag{1}$$

This approach differs slightly from that of [11,13,14] by limiting best buddies to between exclusively two piece sides. This change is required because images with very low variation (e.g., those generated by a computer) often have large numbers of "best buddy cliques" that significantly degrade segmentation performance.

Correctly assembled regions from multiple ground-truth inputs commonly merge into a single segment via very tenuous linking. Our segmentation algorithm trims each segment by removing all articulation points, which is any piece whose removal increases the number of connected segment components. Also removed are any pieces disconnected from the segment's seed after articulation

Algorithm 2. Pseudocode for segmenting the single, reconstructed puzzle

```
function SEGMENT(puzzle)
    puzzle_segments ← {}
    unassigned ← {all pieces in puzzle}
    while |unassigned| > 0 do
        segment ← new empty segment
        seed ← next piece in unassigned
        queue ← [seed]
        while |queue| > 0 do
            piece ← next piece in queue
            add piece to segment
            for each neighbor in NEIGHBORS(puzzle, piece) ⋂ unassigned do
                if ISBESTBUDDY(neighbor, piece) then
                    add neighbor to queue
                    remove neighbor from unassigned
        remove segment articulation pieces
        remove segment pieces disconnected from seed
        add removed pieces back to unassigned
        add segment to puzzle_segments
    return puzzle_segments
```

point deletion. All pieces no longer part of the segment are returned to the unassigned pool. Once this is completed, the segment is in its final form.

At the end of a segmentation round, only segments meeting a set of criteria are saved. First, all segments must exceed a minimum size. In our experiments, a minimum segment size of seven resulted in the best solution quality. If the largest segment exceeds this minimum size, it is automatically saved. Any other segment is saved if its size exceeds both the minimum and some fraction, α (where $0 < \alpha \leq 1$), of the largest segment. We found that $\alpha = 0.5$ provided appropriate balance between finding the largest possible segments and reducing segmentation's execution time.

The only change in subsequent segmentation rounds is the exclusion of all pieces already assigned to a saved segment. Segmentation terminates once either all pieces are assigned to a saved segment or when no segment in a given round exceeds the minimum savable size.

4 Identifying Related Segments

Traditional image stitching involves combining multiple overlapping photographs to form a single panoramic or higher resolution image. The Mixed-Bag Solver's *Stitching* stage uses a similar technique to identify segments that originate from the same ground-truth input.

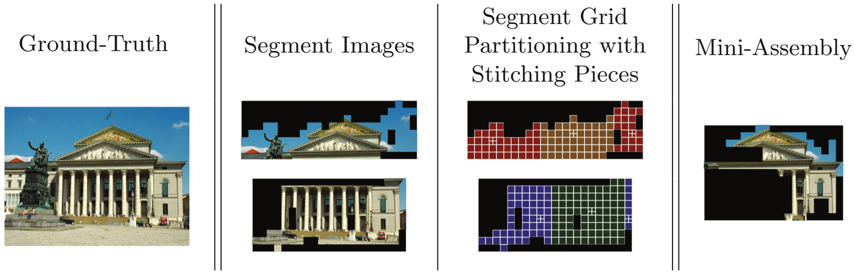


Fig. 3. An input image split into two disjoint segments that are sub-partitioned into a grid of (colored) cells. Stitching pieces are denoted with a white "+" mark. The mini-assembly, which uses a stitching piece from the upper segment, is composed of pieces from both segments (e.g., the building's roof and columns).

4.1 Stitching

Segmentation commonly partitions a single image into multiple disjoint segments. If a pair of such segments are adjacent in an original input, it is expected that they would eventually overlap if allowed to expand. A larger intersection between these two expanded segments (i.e., puzzle piece sets) indicates a stronger relationship. In contrast, if a ground-truth image consists of only a single, saved segment, then that segment generally resists growth. Since inter-segment spatial relationships, if any, are unknown by the solver, segment growth must be allowed, but never forced, to proceed in all directions.

Rather than attempt to grow a segment in its complete form, the Mixed-Bag Solver tests for localized expansion through the use of *grid cells*, which are non-overlapping subregions of a segment. These grid cells are defined by placing a bounding rectangle around the entire segment. Then starting from the upper left corner, this rectangle is partitioned into a grid of a target width (e.g., the equivalent of 10 puzzle pieces wide as used in this paper). This process is shown in Fig. 3 where an image split into two segments, both of which are further subdivided into three grid cells. If a segment's dimensions are not evenly divisible by the target width, then any grid cells along the segment's bottom and rightmost boundaries will be narrower than this ideal target.

Intuitively, it is obvious that expansion can only occur along a segment's edges. This is done by focusing on those grid cells that contain at least one piece next to an *open location*, which is any puzzle slot not occupied by a member of the segment including both the segment's external perimeter and any internal voids. For each such grid cell, localized expansion is done via a *mini-assembly* (MA). Unlike traditional placement, the MA places only a fixed number of pieces (e.g., 100 for all experiments in this paper). This placement size partially dictates the solver's inter-segment relationship sensitivity.

The MA's placement seed is referred to as a *stitching piece* and must be a member of the candidate grid cell. The selection of an appropriate stitching piece is critical; for example, if a piece too close to a boundary is selected, erroneous coupling with unrelated segments may occur. As such, the algorithm finds the set of pieces, if any, within the candidate grid cell whose distance to the nearest open location equals a predefined target (we used a distance of 3 for our experiments). If no pieces satisfy that distance criteria, the target value is decremented until at least one satisfying piece is identified. Then from this pool of possible stitching pieces, the one closest to the grid cell's center is used for stitching.

By selecting the stitching piece closest to the grid cell's center, the solver is able to enforce an approximate maximum inter-stitching piece spacing. This ensures that stitching pieces are not too far apart, which would hinder the detection of subtle inter-segment relationships. It also prevents multiple near-identical mini-assemblies, caused by stitching pieces being too close together, that contribute little added value.

4.2 Quantifying Inter-Segment Relationships

A mini-assembly is performed for each stitching piece, ζ_x, in segment, $\mathit{\Phi}_i$, where $\zeta_x \in \mathit{\Phi}_i$. If the mini-assembly output, MA_{ζ_x}, is composed of pieces from multiple segments, there is a significantly increased likelihood that those segments come from the same ground-truth input.

Equation (2) defines the overlap score between a segment, $\mathit{\Phi}_i$, and any other segment, $\mathit{\Phi}_j$. The intersection between mini-assembly output, MA_{ζ_x}, and segment $\mathit{\Phi}_j$ is normalized with respect to the size of both, since the smaller of the two dictates the maximum possible overlap. Also, this score must use the maximum intersection across all of the segment's mini-assemblies as two segments may only be adjacent along a small portion of their boundaries.

$$Overlap_{\mathit{\Phi}_i,\mathit{\Phi}_j} = \max_{\zeta_x \in \mathit{\Phi}_i} \frac{|MA_{\zeta_x} \bigcap \mathit{\Phi}_j|}{\min(|MA_{\zeta_x}|, |\mathit{\Phi}_j|)} \tag{2}$$

Each segment generally has different mini-assembly outputs, meaning the overlap scores for each permutation of segment pairs is usually asymmetric. All overlap scores are combined into the m by m, square *Segment Overlap Matrix*, whose order, m, is the total number of saved segments.

5 Segment Clustering and Final Assembly

After stitching, the solver performs agglomerative hierarchical clustering of the saved segments to determine the number of ground-truth inputs. This necessitates that the overlap matrix be triangularized into the *Cluster Similarity Matrix*. Each element, $\omega_{i,j}$, in this new matrix represents the similarity (bounded between 0 and 1 inclusive) of segments, $\mathit{\Phi}_i$ and $\mathit{\Phi}_j$; it is calculated via:

$$\omega_{i,j} = \frac{Overlap_{\mathit{\Phi}_i,\mathit{\Phi}_j} + Overlap_{\mathit{\Phi}_j,\mathit{\Phi}_i}}{2}. \tag{3}$$

In each clustering round, the two most similar segment clusters, $\mathit{\Sigma}_x$ and $\mathit{\Sigma}_y$, are merged if their similarity exceeds a specified threshold. Based on a dozen random samplings of between two to five images from the dataset in [12], we observed a minimum similarity of 0.1 provided the best clustering accuracy.

Inter-cluster similarity, $r_{x \cup y,z}$, with respect to any other remaining segment cluster, Σ_z, is updated according to the single-linkage paradigm as shown in Eq. (4), wherein the similarity between any pair of clusters equals the similarity of their two most similar members. Solely the maximum similarities are considered as two clusters may only be adjacent along two of their member segments. The number of segment clusters remaining at the end of hierarchical clustering is the Mixed-Bag Solver's estimate of the ground-truth input count.

$$r_{x \cup y,z} = \max_{\Phi_i \in (\Sigma_x \cup \Sigma_y)} \left(\max_{\Phi_j \in \Sigma_z} \omega_{i,j} \right). \tag{4}$$

Some modern jigsaw puzzle placers including [11,13,14] use a kernel-growing technique. If the placer used by the Mixed-Bag Solver requires this additional step, we select a single seed from each segment cluster. This approach leads to better seed selection since most other placers make their seed decisions either randomly or greedily at runtime. Once this is completed, final piece placement begins simultaneously across all puzzle seeds. The resulting fully reconstructed puzzles, with all pieces placed, are the Mixed-Bag Solver's final outputs.

6 Quality Metrics for Mixed-Bag Puzzles

The direct and neighbor accuracy metrics for quantifying the quality of single puzzle reconstructions were defined in [2] and used by [4,11,13,14]. However, both measures are unusable for mixed-bag puzzles since neither account for two complications unique to this problem, specifically that pieces from multiple ground-truth inputs may be placed in the same generated output and that pieces from a single input image can be spread across different outputs [7].

6.1 Enhanced and Shiftable Direct Accuracy

Puzzle solving involves generating a set of output puzzles, S, from a set of inputs, P. Each $P_i \in P$ is composed of n_i pieces. $c_{i,j}$ is the number of pieces in the same location in both P_i and output, $S_j \in S$. In contrast, $m_{i,j}$ is the total number of pieces from P_i in S_j, making $0 \leq c_{i,j} \leq m_{i,j} \leq n_i$.

Standard direct accuracy (where $|P| = |S| = 1$ and $n_1 = m_{1,1}$) is the fraction of pieces that are correctly placed in the reconstructed output. It is defined as:

$$DA = \frac{c_{1,1}}{n_1}. \tag{5}$$

A solved image is *perfectly reconstructed* if the location of all pieces exactly match the original image (i.e., $DA = 1$) [4].

Our *Enhanced Direct Accuracy Score* (EDAS) in Eq. (6) addresses standard direct accuracy's deficiencies for mixed-bag puzzles in three primary ways. First, since pieces from P_i may be in multiple reconstructed puzzles, EDAS uses the maximum score across all of S so as to focus on the best overall reconstruction of P_i. Second, dividing by n_i marks as incorrect any piece from P_i that is not in S_j. Lastly, the summation of all $m_{k,j}$ penalizes for the placement of any pieces from inputs other than P_i.

$$EDAS_{P_i} = \max_{S_j \in S} \frac{c_{i,j}}{n_i + \sum_{k \neq i}(m_{k,j})} \tag{6}$$

Both standard and enhanced direct accuracy can be misleadingly punitive for shifts in the output, in particular when the solved puzzle's boundaries are not fixed/known. As noted in [7], even a single misplaced piece can cause these metrics to drop to zero. Direct accuracy more meaningfully quantifies output quality if the comparison reference location, l, is allowed to shift within a fixed set of possible locations, L_j, in S_j. As such, our *Shiftable Enhanced Direct Accuracy Score* (SEDAS) in Eq. (7) updates the term $c_{i,j}$ to $c_{i,j,l}$ to denote the use of this variable reference when determining the correctly placed piece count.

$$SEDAS_{P_i} = \max_{S_j \in S} \left(\max_{l \in L_j} \frac{c_{i,j,l}}{n_i + \sum_{k \neq i}(m_{k,j})} \right) \tag{7}$$

For this paper, L_j was the set of all puzzle locations within the radius defined by the Manhattan distance between the upper left corner of S_j and the nearest puzzle piece, inclusive. An alternative approach is for L_j to be the set of all locations in S_j, but that can be computationally prohibitive for large puzzles.

6.2 Enhanced Neighbor Accuracy

Standard neighbor accuracy (where $|P| = |S| = 1$) is the fraction of puzzle piece sides with the same neighbors in both the input and output puzzles. If $a_{i,j}$ is the number of puzzle piece sides with matching neighbors in both P_i and S_j, then for square pieces, this single-puzzle metric is formally defined as:

$$NA = \frac{a_{1,1}}{4n_1}. \tag{8}$$

Similar to the reasons described for EDAS, our *Enhanced Neighbor Accuracy Score* (ENAS), which is defined as:

$$ENAS_{P_i} = \max_{S_j \in S} \frac{a_{i,j}}{4(n_i + \sum_{k \neq i}(m_{k,j}))} \tag{9}$$

addresses standard neighbor accuracy's limitations for mixed-bag puzzles. Neighbor accuracy is immune to shifts [2] making a shiftable version of it unnecessary.

Table 1. Number of solver experiments for each puzzle input count

#Puzzles	2	3	4	5
#Iterations	55	25	8	5

7 Experimental Results

Our experiments followed the standard puzzle parameters established by previous work including [2,4,11,13,14]. All of the square puzzle pieces were 28 pixels wide. We also used the three, 20 image datasets of sizes 432, 540, and 805 pieces from [2,10,12]. Only the more challenging Type 2 mixed-bag puzzles were investigated, meaning piece rotation and puzzle(s) dimensions were unknown.

The current state of the art, Paikin and Tal's algorithm, was used as the comparative performance baseline. In each test, two to five images were randomly selected, without replacement, from the 805 piece dataset [12] and input into the two solvers. Table 1 shows the number of tests performed for each input count.

7.1 Determining the Number of Input Puzzles

Most previous solvers including [2,11,13,14] either assumed or were provided the number of input images. In contrast, the Mixed-Bag Solver determines this information via hierarchical clustering.

Clustering a Single Input Image: The solver's accuracy determining the number of inputs when passed only a single image represents its overall performance ceiling. For the 432 [2], 540 [10], and 805 piece [12] datasets, the solver's accuracy determining that the pieces came from a single puzzle was 100%, 80%, and 85% respectively. While there was a degradation in performance for larger puzzles, it was not significant. In all cases where an error was made, the solver reported that there were two input images.

Input puzzle count errors are more likely for images with large areas of little variation (e.g., a clear sky, smooth water, etc.). These incorrectly classified images have on average lower numbers of best buddies (by 8% and 12% for the 540 and 805 piece datasets respectively), which adversely affected segmentation.

Clustering Multiple Input Images: Figure 4 shows the Mixed-Bag Solver's performance identifying the number of input puzzles when randomly selecting, without replacement, multiple images from the 805 piece dataset. The number of input images was correctly determined in 65% of tests. Likewise, the solver overestimated the number of inputs by more than one in less than 8% tests, with a maximum overestimation of three. Across all experiments, it never underestimated the input puzzle count. This indicates the solver can over-reject cluster mergers due to clusters being too isolated to merge with others.

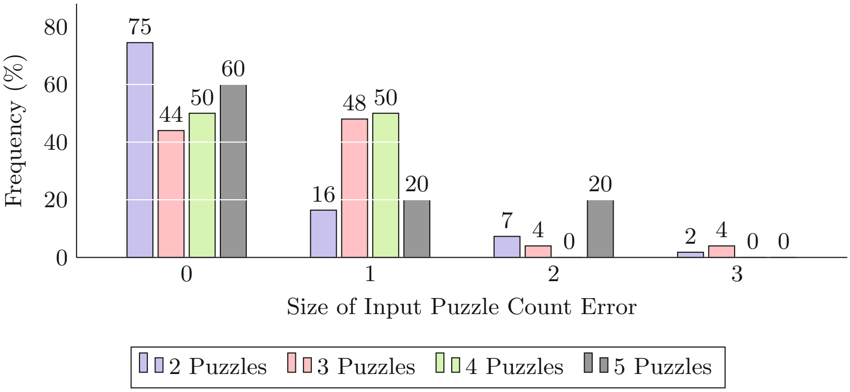


Fig. 4. Multiple input puzzle clustering Accuracy: A correct estimation of the input puzzle count is an error of "0." An overestimation of a single puzzle is an error of "1."

7.2 Comparison of Solver Output Quality for Multiple Input Images

Table 2 contains the comparative results when both solvers were supplied multiple input images. The values for each of the three metrics, namely SEDAS, ENAS, and percentage of puzzles reassembled perfectly, are averaged. The Mixed-Bag Solver (MBS) results are subdivided between when the number of input puzzles was correctly determined (denoted with a "†") versus all combined results ("‡"); the former value represents the performance ceiling had our solver been provided the input puzzle count like Paikin and Tal's algorithm.

Despite receiving less information, the quality of our results exceeded that of Paikin and Tal by between 2.5 to 8 times for SEDAS and up to four times ENAS. The Mixed-Bag Solver was also substantially more likely to perfectly reconstruct the images. Furthermore, unlike Paikin and Tal, our algorithm showed no significant performance degradation as the number of input puzzles increased.

Table 2. Solver performance comparison for multiple input puzzles. Results with "†" indicate the Mixed-Bag Solver (MBS) correctly estimated the input puzzle count while "‡" values include all MBS results.

Puzzle count	Average SEDAS			Average ENAS			Perfect reconstruction		
	MBS†	MBS‡	Paikin	MBS†	MBS‡	Paikin	MBS†	MBS‡	Paikin
2	0.850	0.757	0.321	0.933	0.874	0.462	29.3%	23.6%	5.5%
3	0.953	0.800	0.203	0.955	0.869	0.364	18.5%	18.8%	1.4%
4	0.881	0.778	0.109	0.920	0.862	0.260	25.0%	15.6%	0%
5	0.793	0.828	0.099	0.868	0.877	0.204	20.0%	24%	0%

It should also be noted the Mixed-Bag Solver's performance scores are similar irrespective of whether the input puzzle count estimation was correct. This indicates that any extra puzzles generated were relatively insignificant in size.

Ten Puzzle Solving: The previous maximum number of puzzles reconstructed simultaneously was five by Paikin and Tal. In contrast, our solver reconstructed the 10 image dataset in [6], with a SEDAS and ENAS greater than 0.9 for all images. Despite being provided the input puzzle count, Paikin and Tal's algorithm only had a SEDAS and ENAS greater than 0.9 for a single image as their solver struggled to select quality seeds for that many puzzles.

8 Conclusion and Future Work

We presented an algorithm for simultaneous reassembly of multiple jig swap puzzles without prior knowledge. Despite the current state of the art requiring specification of the input puzzle count, our approach still outperforms it in terms of both the output quality and the supportable number of input puzzles.

Potential improvements to our solver remain that merit further investigation. First, rather than performing segmentation through placement, it may be faster and yield better, larger segments if the entire set of puzzle pieces were treated as nodes in an undirected graph with edges being the best buddy relationships. This would enable segment identification through the use of well-studied graph partition techniques. In addition, our approach requires that stitching pieces be members of a saved segment. Superior results may be achieved if pieces not assigned to a segment are also used, as they may help bridge inter-segment gaps.

References

1. Altman, T.: Solving the jigsaw puzzle problem in linear time. Appl. Artif. Intell. **3**(4), 453–462 (1990)
2. Cho, T.S., Avidan, S., Freeman, W.T.: A probabilistic image jigsaw puzzle solver. In: CVPR, pp. 183–190 (2010)
3. Cho, T.S., Butman, M., Avidan, S., Freeman, W.T.: The patch transform and its applications to image editing. In: CVPR, pp. 1489–1501 (2008)
4. Gallagher, A.C.: Jigsaw puzzles with pieces of unknown orientation. In: CVPR, pp. 382–389 (2012)
5. Garfinkel, S.L.: Digital forensics research: The next 10 years. Digit. Invest. **7**, S64–S73 (2010)
6. Hammoudeh, Z.S.: Ten puzzle dataset. http://www.cs.sjsu.edu/faculty/pollett/masters/Semesters/Spring16/zayd/?10_puzzles.html
7. Hammoudeh, Z.S.: A Fully Automated Solver for Multiple Square Jigsaw Puzzles Using Hierarchical Clustering. Master's thesis, San José State University (2016)
8. Koller, D., Levoy, M.: Computer-aided reconstruction and new matches in the forma urbis romae. Bullettino Della Commissione Archeologica Comunale di Roma **2**, 103–125 (2006)
9. Marande, W., Burger, G.: Mitochondrial DNA as a genomic jigsaw puzzle. Science **318**(5849), 415 (2007)

10. Olmos, A., Kingdom, F.A.A.: McGill calibrated colour image database. http://tabby.vision.mcgill.ca/
11. Paikin, G., Tal, A.: Solving multiple square jigsaw puzzles with missing pieces. In: CVPR, pp. 4832–4839 (2015)
12. Pomeranz, D., Shemesh, M., Ben-Shahar, O.: Computational jig-saw puzzle solving. https://www.cs.bgu.ac.il/~icvl/icvl_projects/automatic-jigsaw-puzzle-solving/
13. Pomeranz, D., Shemesh, M., Ben-Shahar, O.: A fully automated greedy square jigsaw puzzle solver. In: CVPR, pp. 9–16 (2011)
14. Sholomon, D., David, O., Netanyahu, N.S.: A genetic algorithm-based solver for very large jigsaw puzzles. In: CVPR, pp. 1767–1774 (2013)
15. Zhu, L., Zhou, Z., Hu, D.: Globally consistent reconstruction of ripped-up documents. Trans. Pattern Anal. Mach. Intell. **30**, 1–13 (2008)