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

Zayd S. Hammoudeh and Chris Pollett

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

Abstract. The jig swap puzzle is a variant of traditional jigsaw puzzles, 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 outputs' 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 [4], deleted files analysis [5], image editing [6], shredded document reconstruction [7], and DNA fragment reassembly [8].

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 off the level of difficulty [13]. The simplest variety is the *Type 1* puzzle, which fixes piece orientation by disallowing their rotation. Likewise, 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 increases the number of possible solutions by a factor of 4^n for puzzles of n pieces; the dimensions for this type of puzzle may be unknown. *Mixed-bag* puzzles contain pieces from multiple input

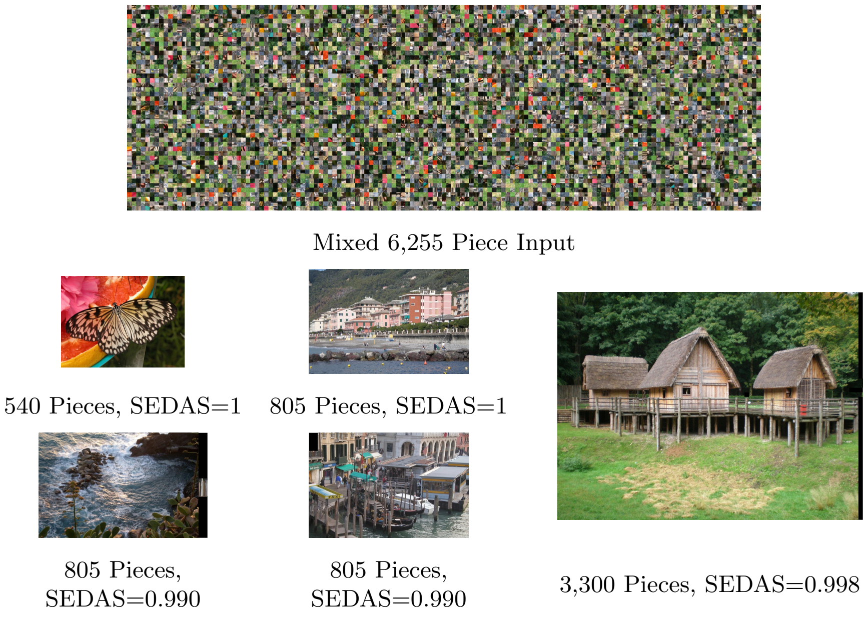

Fig. 1. Fully-Automated Mixed-Bag Puzzle Solving: Our solver generated these results without any external information, including the number of input puzzles. The average, weighted EDAS and ENAS scores were 0.997 and 0.993 respectively.

images as shown in Figure 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.

In 2011, Pomeranz *et al.* developed a greedy 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. They also defined multiple test datasets, some of which are used in this paper.

Paikin & Tal [14] 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.

Algorithm 1 The Mixed-Bag Solver

```
1: function MixedBagSolver(pieces)
2:     segments ← Segmentation(pieces)
3:     overlap_matrix ← Stitch(segments, pieces)
4:     clusters ← Cluster(segments, overlap_matrix)
5:     seeds ← SelectSeeds(clusters)
6:     solved_puzzles ← FinalAssembly(seeds, pieces)
7:     return solved_puzzles
```

In addition, metrics previously proposed for comparing the performance of single puzzle solvers [15] are inadequate when there are multiple simultaneous inputs. As such, we introduce the first quality metrics tailored for mixed-bag puzzles. We also enhance the existing single puzzle 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 highly-distinct seed piece is selected from each cluster for use in the final assembly stage, which generates the reconstructed puzzles 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 & Tal [14] 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 Figure 2, which

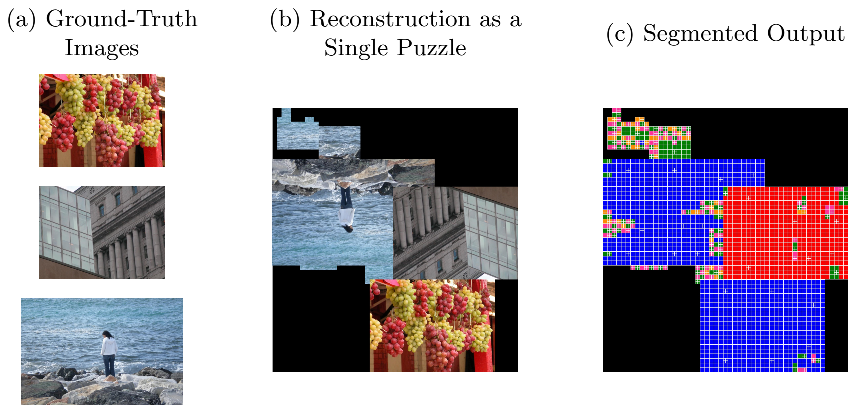

Fig. 2. Segmentation Example: Three ground-truth inputs of two different sizes are shown in (a). All pieces are placed in the single, reconstructed output puzzle in (b). Segmented output in (c) is shown with any contiguous group of matching colored pieces belonging to the same segment. Stitching pieces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

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 described in Algorithm 2, which is partially based on the approach originally proposed by Pomeranz *et al.* in [17].

Segments in the single, reconstructed output are found iteratively, with all pieces eventually assigned to a segment. Each segment's growth starts by adding a single *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 also its best buddy, then that neighbor is also added to the queue. A segment's growth terminates once no pieces remain in the queue to be popped.

As mentioned previously, *best buddies* are any two puzzle pieces, p_i and p_j, whose respective sides, s_x and s_y, are mutually more similar than they are to any other piece's side. This is defined formally 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 definition differs slightly from that of [14, 17, 18] as we limit best buddies to between exclusively two piece sides. This change is required because images with very low variation often have large numbers of "best buddy cliques" that can 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

Algorithm 2 Pseudocode for segmenting the single, reconstructed puzzle

```
 1: function SEGMENT($puzzle$)
 2:     $puzzle\_segments \leftarrow \{\}$
 3:     $unassigned \leftarrow$ {all pieces in $puzzle$}
 4:     while $|unassigned| > 0$ do
 5:         $segment \leftarrow$ new empty segment
 6:         $seed \leftarrow$ next piece in $unassigned$
 7:         $queue \leftarrow [seed]$
 8:         while $|queue| > 0$ do
 9:             $piece \leftarrow$ next piece in $queue$
10:             add $piece$ to $segment$
11:             for each $neighbor$ in NEIGHBORS($puzzle, piece$) do
12:                 if ISBESTBUDDY($neighbor, piece$) then
13:                     add $neighbor$ to $queue$
14:                     remove $neighbor$ from $unassigned$
15:         remove segment articulation pieces
16:         remove segment pieces disconnected from $seed$
17:         add removed pieces back to $unassigned$
18:         add $segment$ to $puzzle\_segments$
19:     return $puzzle\_segments$
```

whose removal increases the number of connected segment components. Also removed are any pieces disconnected from the segment's seed after articulation 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 7 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.

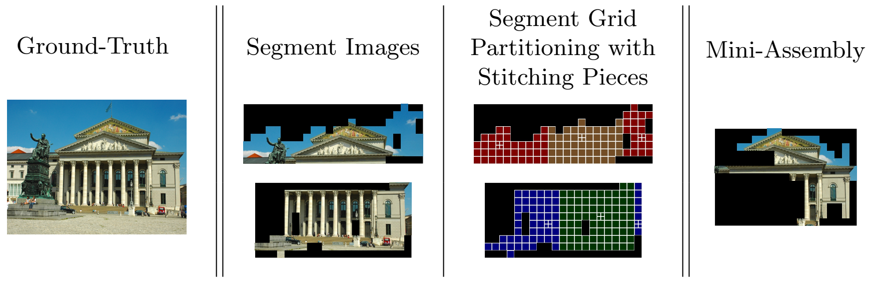

Fig. 3. An input image split into two disjoint segments that are sub-partitioned into a grid of (colored) cells. Stitching pieces are denoted with a white "+" mark. The mini-assembly, which uses a stitching piece from the upper segment, is composed of pieces from both segments (e.g., the building's roof 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 performs localized expansion. Figure 3 shows an image that was partitioned into two segments, both of which are subdivided into multiple, non-overlapping square *grid cells*, via a bounding rectangle starting from the segment's upper left corner. If a segment's dimensions are not evenly divisible by the grid cell target width (e.g., 10 as used in this paper), grid cells along the segment's bottom and right boundaries will be narrower than the specified target. Each grid cell represents a potential candidate for testing segment expansion.

Intuitively, it is obvious that expansion can only occur along a segment's boundaries. 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, a 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, from the candidate 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 selected for stitching.

By using the grid cell's center as the target stitching piece location, 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 that contribute little added value caused by stitching pieces being too close together.

4.2 Quantifying Inter-Segment Relationships

A mini-assembly is performed for each stitching piece, ζ_x, in segment, Φ_i, where $\zeta_x \in \Phi_i$. 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 segment, Φ_i, and any other segment, Φ_j. The intersection between mini-assembly output, MA_{ζ_x} and segment, Φ_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.

$$Overlap_{\Phi_i,\Phi_j} = \max_{\zeta_x \in \Phi_i} \frac{|MA_{\zeta_x} \bigcap \Phi_j|}{\min(|MA_{\zeta_x}|, |\Phi_j|)} \tag{2}$$

Each segment generally has different mini-assembly outputs, meaning the overlap scores for each permutation of segment pairs is often asymmetric. All overlap scores are combined into the m by m, square *Segment Overlap Matrix*, whose the order, m, is the 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 (bound between 0 and 1) of segments, Φ_i and Φ_j; it is calculated via:

$$\omega_{i,j} = \frac{Overlap_{\Phi_i,\Phi_j} + Overlap_{\Phi_j,\Phi_i}}{2} \tag{3}$$

In each hierarchical clustering round, the two most similar segment clusters are combined if their mutual similarity exceeds a minimum threshold. In our experiments, a minimum similarity of 0.1 provided sufficient clustering accuracy, without merging unrelated segments.

When two segment clusters, Σ_x and Σ_y, are merged, the similarity matrix is updated via the single-linkage paradigm, wherein the similarity between any pair of clusters is equal to the similarity of their two most similar members. This is defined with respect to any other remaining segment cluster, Σ_z as:

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

Only 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.

Some modern-jigsaw puzzle placers including [14,17,18] 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 Mixed-Bag Quality Metrics

The direct and neighbor accuracy metrics for quantifying the quality of single puzzle reconstruction were defined in [15], and used by [13,14,17,18]. However, both measures are insufficient for mixed-bag puzzles since neither account for pieces from multiple ground-truth inputs being present in a single generated output nor for pieces from a single input image being spread across many generated outputs.

6.1 Enhanced and Shiftable Direct Accuracy

Direct accuracy is the ratio of pieces, c, placed in the same location in both the ground-truth and solved images with respect to the total number of pieces, n. It is formally defined as:

$$DA = \frac{c}{n} \tag{5}$$

A solved image is *perfectly reconstructed* if the location of all pieces exactly match the original image (i.e., $DA = 1$) [13].

Enhanced Direct Accuracy Score (EDAS) addresses standard direct accuracy's limitations for mixed-bag puzzles. For a puzzle, P_i, in the set of input puzzles P, EDAS is defined as:

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

Since pieces from P_i may be in multiple reconstructed puzzles, EDAS is calculated as the maximum value across all solved puzzles, S, so as to focus on the best overall reconstruction for input P_i. $c_{i,j}$ is the number of pieces from P_i correctly placed in S_j. Dividing by n_i, the number of pieces in P_i, marks as incorrect any piece placed in a puzzle other than S_j. Similarly, the summation of all $m_{k,j}$ penalizes for the placement of any pieces not from P_i that are in S_j.

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. For example, depending on the location of only a single misplaced piece, the accuracy can range anywhere from zero to essentially unchanged. Direct accuracy more meaningfully quantifies solver output quality if the comparison reference point, l, is allowed to shift within a fixed set of possible locations, L. As such, we define the *Shiftable Enhanced Direct Accuracy Score* (SEDAS) as:

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

EDAS' term, $c_{i,j}$ has been changed to $c_{i,j,l}$ to denote the use of l as a custom reference point when determining the number of correctly placed pieces. L_j is the set of all locations in output puzzle S_j from which l can be selected. For this paper, L_j is 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 to be the set of all location in S_j, but that is computationally prohibitive for large puzzles.

6.2 Enhanced Neighbor Accuracy

For a jig swap puzzle composed of n square pieces, neighbor accuracy is the ratio of puzzle piece sides, a, whose neighbor in the original and solved images are the same normalized by the total number of puzzle piece sides via:

$$NA = \frac{a}{4n} \tag{8}$$

Similar to the techniques 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.

7 Experimental Results

Our experiments followed the standard parameters established by previous work including [13–15, 17, 18]. 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

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

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

from [15, 16, 20]. 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 & 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 [20] and input into the two solvers. Table I shows the number of tests performed for each input count.

7.1 Determining the Number of Input Puzzles

Most previous solvers including [14, 15, 17, 18] 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 [15], 540 [16], and 805 piece [20] 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 6 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 resulting in clusters that are too isolated to merge with others.

7.2 Comparison of Solver Output Quality for Multiple Input Images

Table II contains the comparative results for 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

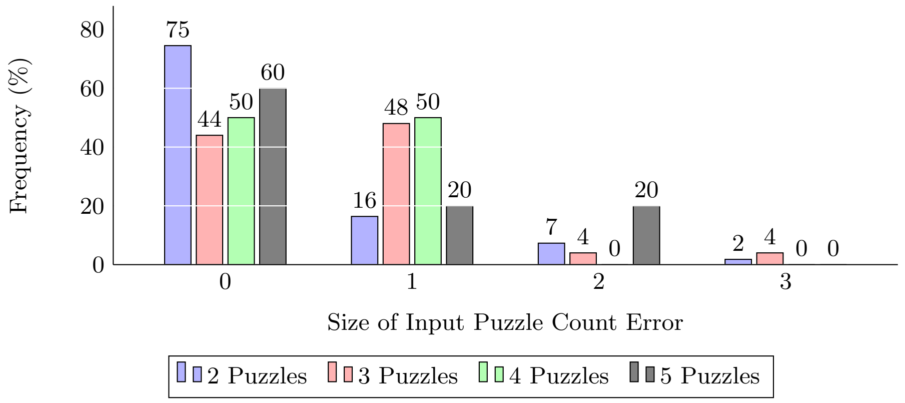

Fig. 4. Multiple Input Puzzle Clustering Accuracy: A correct estimation of the input puzzle count is an error of "0." An overestimation of a single puzzle is an error of "1."

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 & Tal's algorithm.

Despite receiving less information, the quality of our results exceeded that of Paikin & 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 & Tal our algorithm showed no significant degradation in performance as the number of input puzzles increased.

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 & Tal. In contrast, our solver reconstructed the 10 image dataset in [21], with a SEDAS and ENAS greater than 0.9 for all images. In contrast, despite being provided the input puzzle count, Paikin & Tal's algorithm only had a SEDAS and ENAS of 0.9 for a single image as their solver struggled to select quality seeds for each output puzzle.

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	Average SEDAS			Average ENAS			Perfect Reconstruction		
Count	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%

8 Conclusion and Future Work

We presented an algorithm for simultaneous reassembly of multiple jig swap puzzles without any prior knowledge. Despite the fact that the current state of the art requires specification of the input puzzle count, our algorithm still outperforms it in terms of both the generated outputs' quality and the supportable number of input puzzles.

Potential improvements to our solver remain that merit further investigation. First, rather 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 the use of well-studied graph partition techniques. In addition, our approach required that stitching pieces be members of a saved segment. Superior results may be achieved if pieces not assigned to a segment were also used as they may better bridge inter-segment gaps.

References

[1] T. Altman, "Solving the jigsaw puzzle problem in linear time," *Applied Artificial Intelligence*, vol. 3, no. 4, pp. 453–462, Jan. 1990.

[2] E. D. Demaine and M. L. Demaine, "Jigsaw puzzles, edge matching, and polyomino packing: Connections and complexity," *Graphs and Combinatorics*, vol. 23 (Supplement), pp. 195–208, June 2007.

[3] 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*, vol. 27, no. 3, Aug. 2008.

[4] M. L. David Koller, "Computer-aided reconstruction and new matches in the Forma Urbis Romae," *Bullettino Della Commissione Archeologica Comunale di Roma*, vol. 2, pp. 103–125, 2006.

[5] S. L. Garfinkel, "Digital forensics research: The next 10 years," *Digital Investigation*, vol. 7, Aug. 2010.

[6] T. S. Cho, M. Butman, S. Avidan, and W. T. Freeman, "The patch transform and its applications to image editing," *Proceedings of the 2008 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[7] L. Zhu, Z. Zhou, and D. Hu, "Globally consistent reconstruction of ripped-up documents," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, pp. 1–13, 2008.

[8] W. Marande and G. Burger, "Mitochondrial DNA as a genomic jigsaw puzzle," *Science*, vol. 318, no. 5849, pp. 415–415, 2007.

[9] Y.-X. Zhao, M.-C. Su, Z.-L. Chou, and J. Lee, "A puzzle solver and its application in speech descrambling," in *Proceedings of the 2007 International Conference on Computer Engineering and Applications.* World Scientific and Engineering Academy and Society, 2007, pp. 171–176.

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

[11] D. Goldberg, C. Malon, and M. Bern, "A global approach to automatic solution of jigsaw puzzles," in *Proceedings of the Eighteenth Annual Symposium on Computational Geometry*, ser. SCG '02, 2002, pp. 82–87.

[12] T. R. Nielsen, P. Drewsen, and K. Hansen, "Solving jigsaw puzzles using image features," *Pattern Recogn. Lett.*, vol. 29, no. 14, pp. 1924–1933, Oct. 2008. [Online]. Available: http://dx.doi.org/10.1016/j.patrec.2008.05.027

[13] A. C. Gallagher, "Jigsaw puzzles with pieces of unknown orientation," in *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, ser. CVPR '12. IEEE Computer Society, 2012, pp. 382–389.

[14] G. Paikin and A. Tal, "Solving multiple square jigsaw puzzles with missing pieces," in *Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, ser. CVPR '15. IEEE Computer Society, 2015.

[15] T. S. Cho, S. Avidan, and W. T. Freeman, "A probabilistic image jigsaw puzzle solver," in *Proceedings of the 2010 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, ser. CVPR '10. IEEE Computer Society, 2010, pp. 183–190.

[16] A. Olmos and F. A. A. Kingdom, "McGill calibrated colour image database," http://tabby.vision.mcgill.ca/, 2005, (Accessed on 05/01/2016).

[17] D. Pomeranz, M. Shemesh, and O. Ben-Shahar, "A fully automated greedy square jigsaw puzzle solver," in *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, ser. CVPR '11. IEEE Computer Society, 2011, pp. 9–16.

[18] D. Sholomon, O. David, and N. S. Netanyahu, "A genetic algorithm-based solver for very large jigsaw puzzles," in *Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, ser. CVPR '13. IEEE Computer Society, 2013, pp. 1767–1774.

[19] K. Son, J. Hays, and D. B. Cooper, "Solving square jigsaw puzzles with loop constraints," in *Proceedings of the 2014 European Conference on Computer Vision (ECCV)*. Springer, 2014, pp. 32–46.

[20] D. Pomeranz, M. Shemesh, and O. Ben-Shahar, "Computational jigsaw puzzle solving," https://www.cs.bgu.ac.il/~icvl/icvl_projects/automatic-jigsaw-puzzle-solving/, 2011, (Accessed on 05/01/2016).

[21] Z. S. Hammoudeh, "Ten puzzle dataset." [Online]. Available: http://www.cs.sjsu.edu/faculty/pollett/masters/Semesters/Spring16/zayd/?10_puzzles.html