A FULLY AUTOMATED SOLVER FOR MULTIPLE SQUARE JIGSAW PUZZLES USING HIERARCHICAL CLUSTERING

A Thesis

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Zayd Hammoudeh

December 2016

© 2016

Zayd Hammoudeh

ALL RIGHTS RESERVED

The Designated Thesis Committee Approves the Thesis Titled

A FULLY AUTOMATED SOLVER FOR MULTIPLE SQUARE JIGSAW PUZZLES USING HIERARCHICAL CLUSTERING

by

Zayd Hammoudeh

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

December 2016

Dr. Chris Pollett	Department of Computer Science
Dr. Thomas Austin	Department of Computer Science
Dr. Teng Moh	Department of Computer Science

ABSTRACT

A Fully Automated Solver for Multiple Square Jigsaw Puzzles Using Hierarchical Clustering

by Zayd Hammoudeh

The square jigsaw puzzle is a variant of traditional jigsaw puzzles, wherein all pieces are equal-sized squares; these pieces must be placed adjacent to one another to reconstruct an original image. This thesis proposes an agglomerative hierarchical clustering based solver that can simultaneously reconstruct multiple square jigsaw puzzles. This solver requires no additional information beyond an input bag of puzzle pieces and significantly outperforms the current state of the art in terms of both the quality of the reconstructed outputs as well the number of input puzzles it supports. In addition, this thesis defines Enhanced Direct Accuracy Score (EDAS), Shiftable Enhanced Direct Accuracy Score (SEDAS), and Enhanced Neighbor Accuracy Score (ENAS), which are the first quality metrics specifically tailored for multi-puzzle solvers. This thesis also outlines the first standards for visualizing best buddies and the quality of solver solutions.

DEDICATION

To my mother.

TABLE OF CONTENTS

CHAPTER

APPENDIX

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

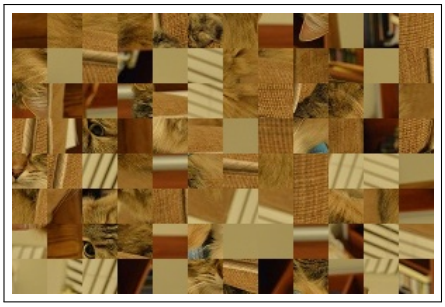
Jigsaw puzzles were first introduced in the 1760s when they were made from wood. The name ‘‘jigsaw’’ derives from the jigsaws that were used to carve the wooden pieces. The 1930s saw the introduction of the modern jigsaw puzzle where an image was printed on a cardboard sheet that was cut into a set of interlocking pieces [1, 2]. Although jigsaw puzzles had been solved by children for more than two centuries, it was not until 1964 that the first automated jigsaw puzzle solver was proposed by Freeman & Gardner [3]. While an automated jigsaw puzzle solver may seem trivial, the problem has been shown by Altman [4] and Demaine & Demaine [5] to be strongly NP-complete when inter-piece compatibility is not a reliable metric for determining adjacency.

In recent years, most research into automated jigsaw puzzle solving has focused on jig swap puzzles, where all pieces are equal-sized, non-overlapping squares.[1] An example of a jig swap puzzle is shown in Figure 1. Since all pieces are squares, shape cannot be considered when determining piece adjacency. Moreover, in this specific variant of the problem, the original, also known as ‘‘ground-truth,’’ input is not provided to the solver. These two factors significantly increase the problem’s difficulty as the complete solution’s structure must be determined using only the image information on the individual pieces.

There are clear parallels between jig swap puzzle solving and other domains where an unknown object must be reconstructed from a set of component pieces. As such, strategies developed for use with jigsaw puzzles can often be generalized to many practical problems. Some example applications where such techniques have

[1] Unless otherwise noted, the phrase ‘‘jigsaw puzzle’’ is used in this thesis to refer to specifically jig swap puzzles.

(a) Ground-truth image



(b) Randomized jig swap puzzle

Figure 1: Jig swap puzzle example

been applied include: reassembly of archaeological artifacts [6, 7], forensic analysis of deleted files [8], image editing [9], shredded document reconstruction [10], DNA fragment reassembly [11], and speech descrambling [12].

This thesis presents a fully automated solver for the simultaneous assembly of multiple jigsaw puzzles, with an overview of the architecture provided in Chapter 3. Chapter 4 introduces a set of new metrics specifically tailored for quantifying the quality of outputs of multiple puzzle solvers; the chapter also outlines a set of standards for visualizing the characteristics of solver outputs. Lastly, Chapter 5 compares the performance of this new solver with the current state of the art.

CHAPTER 2

Previous Work

Computational jigsaw puzzle solvers have been studied since the 1960s when Freeman & Gardner [3] proposed an algorithm that could solve puzzles of up to nine pieces using only piece shape. Since then, the focus of research has gradually shifted from traditional jigsaw puzzles to jig swap puzzles.

In 2010, Cho *et al.* [13] proposed one of the first modern, computational jig swap puzzle solvers; their approach relied on a graphical model built around a set of one or more ''anchor piece(s),'' whose position is fixed in the correct location before placement of other pieces begins. Future solvers would improve on Cho *et al.*'s results while simultaneously reducing the amount of information (i.e., beyond the set of pieces) passed to the solver.

A significant contribution of Cho *et al.* is that they were first to use the LAB (Lightness and the A/B opponent color dimensions) colorspace to encode image pixels. LAB was selected due to its property of normalizing the lightness and color variation across all three pixel dimensions. Cho *et al.* also proposed a measure for quantifying the pairwise distance between two puzzle pieces that became the basis of most future work.

Pomeranz *et al.* [14] published an iterative, greedy, jig swap puzzle solver in 2011. Their approach did not rely on anchor pieces, and the only information passed to the solver were the pieces, their orientation, and the puzzle dimensions. In addition, Pomeranz *et al.* introduced the concept of ''best buddies,'' which is any pair of pieces that are more compatible with each other on their respective sides than they are to any other piece. This is formally defined in Equation (1) for side s_x (e.g., top, left, right, bottom) of puzzle piece, p_i, and side, s_y, of piece p_j. $C(p_i, s_x, p_j, s_y)$ represents the compatibility between the two pieces' respective sides.

$$\forall p_k \forall s_z, C(p_i, s_x, p_j, s_y) \geq C(p_i, s_x, p_k, s_z)$$

$$\text{and} \tag{1}$$

$$\forall p_k \forall s_z, C(p_j, s_y, p_i, s_x) \geq C(p_j, s_y, p_k, s_z)$$

The best buddies relationship has served as both a metric for estimating the quality of a solver output [15] as well as the foundation of some solvers' assemblers [16]. Best buddies are discussed extensively in Sections 3.2.2, 4.3, and 5.1.1 of this thesis.

An additional key contribution of Pomeranz *et al.* is the creation of three image benchmarks. The first benchmark is comprised of twenty puzzles with 805 pieces each; this benchmark is used as the test set for the experiments described in Chapter 5. There are three images in each of the other two benchmarks, with images in the first data set having 2,360 pieces while those in the other benchmark have 3,300 pieces.

In 2012, Gallagher [17] formally categorized jig swap puzzle problems into four primary types. The following is Gallagher's proposed terminology; his nomenclature is used throughout this thesis.

- **Type 1 Puzzle**: The dimensions of the puzzle (i.e., the width and height of the ground-truth image in number of pixels) are known. The orientation/rotation of each piece is also known, which means that there are exactly four pairwise relationships between any two pieces. In addition, the solver may be provided with the correct location of one or more ‘‘anchor’’ pieces. This type of puzzle is the focus of [13, 14].
- **Type 2 Puzzle**: This is an extension of the Type 1 puzzle, where pieces may be rotated in 90° increments (e.g., 0°, 90°, 180°, or 270°); in comparison to Type 1, this change alone increases the number of possible solutions by a factor of 4^n, where n is the number of puzzle pieces. Additionally, all piece locations

are unknown, which means there are no anchor pieces. Lastly, the dimensions of the ground-truth image may be unknown.

- **Type 3 Puzzle**: All puzzle piece locations are known, and only the rotation of the pieces is unknown. This is the least computationally complex of the puzzle variants and is generally considered the least interesting. Type 3 puzzles are not explored as part of this thesis.
- **Mixed-Bag Puzzle**: The input set of pieces are from multiple puzzles. The solver may output either a single, merged puzzle, or it may separate the puzzle pieces into disjoint sets that ideally align with the set of ground-truth input images. This type of puzzle is the primary focus of this thesis.

In 2013, Sholomon *et al.* [15] presented a genetic algorithm-based solver for Type 1 puzzles. By moving away from the greedy paradigm used by Pomeranz *et al.*, Sholomon *et al.*'s approach is more immune to suboptimal decisions early in the placement process. Sholomon *et al.*'s algorithm is able to solve puzzles of significantly larger size than other techniques (e.g., greater than 23,000 pieces). What is more, Sholomon *et al.* defined three new large image benchmarks; the specific puzzle sizes are 5,015, 10,375, and 22,834 pieces [18].

Paikin & Tal [16] introduced in 2015 a greedy solver that handles both Type 1 and Type 2 puzzles, even if those puzzles are missing pieces. What is more, their algorithm is one of the first to support Mixed-Bag Puzzles. While Paikin & Tal's algorithm represents the current state of the art, it has serious limitations. For example, similar to previous solvers, Paikin & Tal's algorithm must be told the number of input puzzles. In many practical applications, this information may not be known.

Another limitation arises from the fact that Paikin & Tal's algorithm places

pieces using a single-pass, kernel growing approach. As such, a single piece is used as the seed of each output puzzle, and all subsequent pieces are placed around the expanding kernel. If a seed is selected poorly, the quality of the solver output may be catastrophically degraded. Despite this, their algorithm only requires that a seed piece have best buddies on each of its sides and that the seed's best buddies also have best buddies on each of their sides. Therefore, the selection of the seed is based on essentially 13 pieces. What is more, the selection of the seed is performed greedily at run time. Through the combination of these two factors, it is common that the seeds of multiple output puzzles come from the same ground-truth image.

The limitations of Paikin & Tal's algorithm are addressed by this thesis' Mixed-Bag Solver, which is described in Chapter 3. Since Paikin & Tal's algorithm represents the current state of the art, it is used as this thesis' assembler. What is more, their algorithm is used as the baseline for all performance comparisons.

CHAPTER 3

Mixed-Bag Solver Overview

When humans solve jigsaw puzzles, it is common that they first correctly assemble small regions of the puzzle and then merge those smaller assemblies to form larger ones. This strategy forms the basis of the Mixed-Bag Solver presented in this thesis. There are five distinct solver stages, namely: segmentation, stitching, hierarchical segment clustering, seed piece selection, and final assembly. The relationship between each stage is shown in Figure 2. Pseudocode, including the input(s) and output of each stage, is included in Algorithm 1.

The following subsections describe each of the Mixed-Bag Solver's stages. An additional associated component referred to as the "assembler" (not shown in Figure 2) is also discussed.

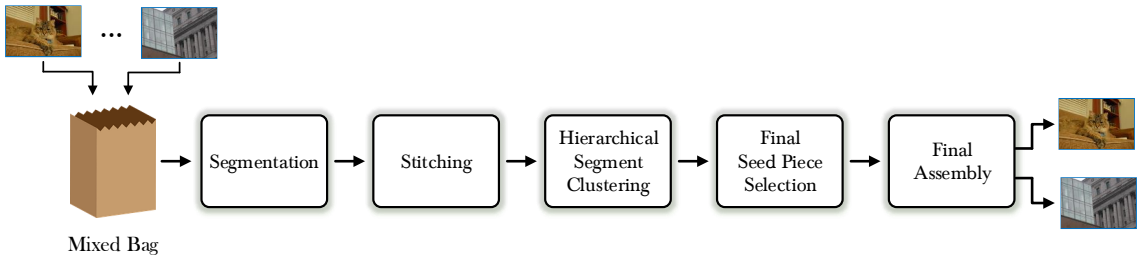


Figure 2: Relationship between the Mixed-Bag Solver's components

3.1 Assembly

The assembler places the individual pieces in the solved puzzle. The Mixed-Bag Solver's architecture is largely independent of the particular assembler used. Hence, any improvements made to the assembler will also improve the Mixed-Bag Solver's performance. Likewise, assemblers can be interchanged if particular ones perform better in specific applications.

This thesis uses the assembly algorithm proposed by Paikin & Tal [16] as it is the current state of the art and because it is one of the few assemblers that natively

Algorithm 1 Pseudocode for the Mixed-Bag Solver

```
1: function MixedBagSolver(puzzle_pieces)
2:     saved_segments ← Segmentation(puzzle_pieces)
3:     overlap_matrix ← Stitching(saved_segments, puzzle_pieces)
4:     clusters ← HierarchicalClustering(saved_segments, overlap_matrix)
5:     seed_pieces ← FindSeedPieces(clusters)
6:     solved_puzzles ← RunFinalAssembly(seed_pieces, puzzle_pieces)
7:     return solved_puzzles
```

supports Mixed-Bag puzzles. The following two subsections describe this thesis' implementation of their assembler as well as the time complexity of assembly.

3.1.1 Assembler Implementation

Paikin & Tal wrote their algorithm in Java, and as of this publication, the source code has not been released. Hence, their algorithm was fully reimplemented as part of this thesis using the description in [16]. This thesis' implementation is written in the Python programming language and is fully open source. While the results generated by the two algorithms should be very similar, it is expected that the underlying software architecture is significantly different.

No execution time comparisons between Paikin & Tal's algorithm and the Mixed-Bag Solver are included with this thesis since Java is generally significantly faster than Python [19]. Instead, the next subsection reviews the expected time complexity of both their algorithm and the Mixed-Bag Solver.

3.1.2 Assembler Time Complexity

Paikin & Tal's assembler relies on a set of inter-puzzle piece similarity metrics. As with all other jig swap solvers, these distances are calculated between all pairs of pieces, making the time required to calculate inter-piece similarity $O(n^2)$, where n is the number of puzzle pieces. If an input image has sufficient inter-piece variation, then the time complexity to place all pieces is $\Theta(n\lg(n))$, since a heap is used in this

thesis' implementation to determine the piece placement order. However, if most pieces are sufficiently similar that there are relatively few best buddies, then piece placement can be as slow as $O(n^3)$ since the inter-piece similarity may need to be recalculated after each piece is placed.

During the segmentation stage, the Mixed-Bag Solver performs assembly at least once, but usually more times. Placement is performed another time during the final assembly stage. Hence, while the execution time for the Mixed-Bag solver is necessarily longer than that of any assembler that may be used, they both generally share the same time complexity since in most cases the number of times placement is performed is not directly related to the number of puzzle pieces.

3.2 Segmentation

Segmentation provides basic structure to a bag of puzzle pieces by partitioning them into disjoint sets, referred to here as segments. These segments are partial puzzle assemblies where there is a high degree of confidence that the pieces are placed correctly. As detailed in Algorithm 1, the only input to segmentation is a bag of puzzle pieces; the solver takes no other inputs. It is expected that pieces from the same ground-truth input may be assigned to multiple segments. Section 3.4 describes how such segments are merged using hierarchical clustering.

Appendix A shows an example segmentation round with two input images. It is included as a visual reference of the segmentation process.

3.2.1 Overview of the Segmentation Procedure

Algorithm 2 outlines the basic segmentation framework; the implementation is iterative and will have one or more rounds. In each round, all pieces not yet assigned to a saved segment are assembled as if they all belong to a single ground-truth image. This strategy eliminates the need to make any assumptions at this early stage

Algorithm 2 Pseudocode for the complete segmentation algorithm

```
1: function SEGMENTATION(puzzle_pieces)
2:     saved_segments ← {}
3:     unassigned_pieces ← {puzzle_pieces}
4:     loop
5:         solved_puzzle ← RUNSINGLEPUZZLEASSEMBLY(unassigned_pieces)
6:         puzzle_segments ← SEGMENTPUZZLE(solved_puzzle)
7:         max_segment_size ← maximum size of segment in puzzle_segments
8:         if max_segment_size < smallest_allowed then
9:             return saved_segments
10:        for each segment ∈ puzzle_segments do
11:            if |segment| > max(α·max_segment_size, smallest_allowed) then
12:                add segment to saved_segments
13:                remove pieces in segment from unassigned_pieces
```

regarding the number of input puzzles.

Section 3.2.2 describes the procedure used to create the individual segments. The largest segment in each round is passed to the Stitching stage described in Section 3.3.[1] Similarly, the multiplicative scalar term "α" in Algorithm 2 dictates which other segments are also passed to stitching. In this thesis, α was set to 0.5, meaning that all segments that were at least half the size of the largest one were also saved. This approach provided sufficient balance between finding the largest possible segments while limiting overall execution time.

Any piece that is assigned to a saved segment is removed from the set of unassigned pieces. Hence, they will not be placed in future segmentation rounds. Segmentation continues until all pieces have been assigned to sufficiently large segments or until no segment exceeds the minimum allowed size.

[1] All saved segments must exceed a minimum size. For this thesis, it was observed that a minimum segment size of 7 provided the best balance between solution quality and algorithm execution time.

Algorithm 3 Pseudocode for segmenting a solved puzzle

```
1: function SegmentPuzzle(solved_puzzle)
2:     puzzle_segments ← {}
3:     unassigned_pieces ← {all pieces in solved_puzzle}
4:     while |unassigned_pieces| > 0 do
5:         segment ← new empty segment
6:         seed_piece ← next piece in unassigned_pieces
7:         queue ← [seed_piece]
8:         while |queue| > 0 do
9:             piece ← next piece in queue
10:            add piece to segment
11:            for each neighbor of piece do
12:                if IsBestBuddies(neighbor, piece) then
13:                    add neighbor to queue
14:                    remove neighbor_piece from unassigned_pieces
15:        articulation_points ← FindArticulationPoints(segment)
16:        remove articulation_points from segment
17:        disconnected_pieces ← FindDisconnectedPieces(segment)
18:        remove disconnected_points from segment
19:        add articulation_points and disconnected_pieces to unassigned_pieces
20:        add segment to puzzle_segments
21:    return puzzle_segments
```

3.2.2 Partitioning a Puzzle into Segments

The function "SegmentPuzzle" in Algorithm 3 partitions a solved puzzle into disjoints segments; this procedure is adapted from the region growing segmentation algorithm proposed by Pomeranz *et al.*, where it was shown to have greater than 99.7% accuracy identifying genuine neighbors [14].

Whenever a piece is added to a segment, the algorithm examines all of that piece's neighbors. Those adjacent pieces are also added to the segment if they are in the unassigned pieces pool and if their neighbor inside the segment is a "best buddy" (as checked by the predicate "IsBestBuddies"). Segment growth terminates once there are no neighboring pieces that satisfy both of these two criteria.

In Pomeranz *et al.*'s segmentation algorithm, no changes were made to a

segment after it reached its maximum size. Their approach is sufficient when solving only a single puzzle at a time. However, in Mixed-Bag puzzles, it is common that correctly assembled regions from different ground-truth inputs are joined into a single segment; this is usually through a very tenuous linking in the form of a narrow bridge no wider than a single piece. Section 3.2.3 describes how the Mixed-Bag Solver post-processes each segment to prevent this erroneous segment merging.

3.2.3 Articulation Points

A segment can be modeled as a graph with a single connected component. The individual puzzle pieces represent vertices while edges are the best buddy relationships between adjacent pieces. An articulation point is any vertex (i.e., puzzle piece) whose removal increases the number of connected components. The Mixed-Bag Solver identifies the articulation points using the algorithm proposed by [20]; any articulation pieces are then removed from the segment and returned to the set of unassigned pieces. This step necessarily causes other piece(s) to become disconnected from the segment's seed. Those pieces are also removed from the segment and marked as unassigned. Once this has been completed, the segment is in its final form.

3.3 Stitching

As discussed previously, a segment represents an ordering of pieces where there is a particularly high degree of confidence that placement is correct. A single ground-truth image is commonly partitioned into multiple segments. Since the Mixed-Bag Solver is not supplied with the number of input images, it must quantify the extent to which any pairs of segments are related to ensure it can accurately estimate the number of ground-truth inputs.

If two segments are adjacent in a ground-truth image, it is expected that they would eventually merge if one segment were allowed to expand. Since it is not known

in which relative direction (if any) an adjacent segment may be located, a segment should be allowed to grow in all directions; however, it should not be forced to expand in a certain direction as this may lead to the formation of erroneous inter-segment coupling. This strategy forms the foundation of segment stitching, which is described in the following subsections.

3.3.1 Mini-Assemblies and Stitching Pieces

As mentioned previously, a segment should be allowed, but not forced, to expand in all directions to identify related segments. To achieve this, the Mixed-Bag Solver introduces the concept of a ‘‘mini-assembly,’’ which is similar to the standard assembly process described in Section 3.1, with the expectation that only a limited number of pieces are placed.[2] The seeds for each of these mini-assemblies is referred to as a ‘‘stitching piece’’ since they serve the role of ‘‘stitching’’ together associated segments.

3.3.2 Stitching Piece Selection

If stitching pieces are poorly selected, two divergent, yet deleterious outcomes may occur. First, placing the stitching pieces too close to one another can add significant overhead without creating much tangible value. In contrast, if stitching pieces are too far apart, the solver may not be able to detect subtle inter-segment relationships. Algorithm 4 details the procedure used by the Mixed-Bag Solver to select stitching pieces that balances these two concerns. The implementation of this algorithm is described in detail in the following two subsections.

3.3.2.1 Spacing Stitching Pieces from Open Locations

It is not sufficient for stitching pieces to be placed solely around the external perimeter of a segment as it is common for segments to have internal voids, where no pieces are present. As such, stitching pieces are placed near ‘‘open locations,’’ which

[2] In this thesis, a mini-assembly places exactly 100 pieces.

Algorithm 4 Pseudocode for selecting a segment's stitching pieces

```
1: procedure FindStitchingPieces(segment_pieces)
2:     FindPieceDistanceToOpen(segment_pieces)
3:     segment_stitching_pieces ← {}
4:     segment_grid_cells ← PartitionIntoGrid(segment_pieces)
5:     for each grid_cell ∈ segment_grid_cells do
6:         if HasPieceAdjacentToOpen(grid_cell) then
7:             candidates ← {pieces} ∈ grid_cell closest to target_distance_to_open
8:             stitching_piece ← piece ∈ candidates closest to center of grid_cell
9:             add stitching_piece to segment_stitching_pieces
10:    return segment_stitching_pieces
```

have either a piece from a different segment or no piece at all. If a stitching piece is too close to one of these open locations, erroneous coupling between unrelated segments may occur. Algorithm 4 invokes the function FindPieceDistanceToOpen to determine the distance of each segment piece to the nearest open location; the implementation of this function is shown in Algorithm 5.

FindPieceDistanceToOpen follows an iterative boundary tracing technique; hence, during each iteration of the **while** loop on line 5, the algorithm explores all segment pieces whose distance to the nearest open location is equal to *distance_to_open*. Therefore, any pieces explored in the first iteration of the **while** loop have a distance of 1 to the nearest open while those explored in the second iteration have distance 2, etc. This approach is robust enough to handle internal voids as well as potential segment necking, where two larger segment components are joined by a narrower bridge. In addition, since each piece is explored only once, the execution time of this algorithm is $O(n)$, where n is the number of pieces in the segment.

Algorithm 5 Pseudocode for determining the Manhattan distance between each segment piece and the nearest open location

```
1: procedure FindPieceDistanceToOpen(segment_pieces)
2:     explored_pieces ← {}
3:     locations_at_prev_dist ← {open locations adjacent to segment_pieces}
4:     distance_to_open ← 1
5:     while |explored_pieces| > 0 do
6:         locations_at_current_dist ← {}
7:         for each prev_dist_loc ∈ locations_at_prev_dist do
8:             for each adjacent_loc of prev_dist_loc do
9:                 if ∃ piece at adjacent_loc and piece ∉ explored_pieces then
10:                    set distance_to_open for piece
11:                    add piece to explored_pieces
12:                    add adjacent_loc to locations_at_current_dist
13:        locations_at_prev_dist ← locations_at_current_dist
14:        distance_to_open ← distance_to_open + 1
```

3.3.2.2 Inter-Stitching Piece Spacing

If stitching pieces are too close together, the outputs from several mini-assemblies will be almost identical, which means that the additional stitching pieces contribute little value. To address inter-stitching piece spacing, Algorithm 4 sub-partitions each segment into a grid of adjacent cells; this allows the algorithm to easily space out the stitching pieces at some maximum spacing. The grid spans the entire segment starting from upper left corner. For this thesis, the grid cell width was set to 10 pieces.[3]

Stitching pieces will only be selected from those grid cells that have at least one puzzle piece adjacent to an open location. For such grid cells, the algorithm finds the set of pieces (if any) whose distance to the nearest open location equals a predefined target.[4] If no pieces satisfy that criteria, then the target value is decremented until at least one piece is identified. From among the set of candidates that satisfy the

[3]If the segment dimensions are not evenly divisible by the target grid cell width, those cells along the bottom and right boundaries of the segment will be narrower than the specified target.

[4]For this thesis, the target distance to the nearest open location was set to 3.

distance-to-the-nearest-open-location criteria, the piece closest to the grid cell center is selected for stitching.

3.3.3 Quantifying Inter-Segment Relationships

As mentioned previously, 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}, includes pieces from multiple segments, there is a significantly increased likelihood that the segments came from the same ground-truth input.

Equation (2) defines the overlap coefficients 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 mini-assembly's size as well as potentially the size of segment Φ_j, since the latter will dictate the maximum overlap if $|\Phi_j| < |MA_{\zeta_x}|$.

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

The outputs of the mini-assemblies will vary between segments based on their respective stitching pieces as well as potentially the segment sizes. Hence, in most cases, the overlap coefficient is asymmetric, meaning: $Overlap_{\Phi_i,\Phi_j} \neq Overlap_{\Phi_j,\Phi_i}$. All of these asymmetric, inter-segment, overlap coefficients are combined into an m by m matrix, where m is the number of saved segments. Section 3.4.1 defines how this ‘‘Segment Overlap Matrix’’ is normalized to quantify inter-segment similarity.

3.4 Hierarchical Clustering of Segments

Agglomerative hierarchical clustering is a bottom-up clustering algorithm where in each round, two clusters are merged. Algorithm 6 shows the basic hierarchical clustering procedure used by the Mixed-Bag Solver; it is adapted from [21]. The only inputs are the saved segments and the overlap matrix calculated during stitching.

Algorithm 6 Pseudocode for hierarchical segment clustering

```
1: function HierarchicalClustering(saved_segments, overlap_matrix)
2:     segment_clusters = {}
3:     for each segment Φ_i ∈ saved_segments do
4:         add new segment cluster Σ_i containing Φ_i to segment_clusters
5:     Compute the similarity matrix Γ from overlap_matrix
6:     while maximum similarity in Γ > min_cluster_similarity do
7:         Merge the two most similar clusters Σ_i and Σ_j in segment_clusters
8:         Update the similarity matrix Γ for the merged clusters
9:     return cluster_segments
```

3.4.1 Building the Initial Similarity Matrix

All elements in the Segment Overlap Matrix, except those along the diagonal, are populated with meaningful values. In contrast, hierarchical clustering requires a triangular, similarity matrix. Equation (3) defines how the inter-segment similarity, $\omega_{i,j}$, for segments Φ_i and Φ_j is calculated from their respective asymmetric, overlap coefficients.

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

Like the overlap matrix, the initial segment similarity matrix, Γ, is size m by m, where m is the number of saved segments. Each element in Γ is defined by Equation (4). Both i and j are integers bounded between 1 and m (inclusive). What is more, all elements in Γ are bounded between 0 and 1, also inclusive.

$$\Gamma = \begin{cases} 0 & j \geq i \\ \omega_{i,j} & i < j \end{cases} \tag{4}$$

3.4.2 Updating the Similarity Matrix via Single Linking

Whenever two clusters merge, the similarity matrix is updated using the Single Link paradigm, which means that the similarity between any pair of clusters is equal to the similarity of the two most similar segments from each cluster. This approach is

required because two segment clusters may only be adjacent along the border of two of the composite segments.

If segment clusters Σ_x and Σ_y are merged, then Equation (5) defines the similarity between this new merged cluster and any other segment cluster Σ_z. Note that segment, Φ_i, is a member of the union of segment clusters Σ_x and Σ_y, while segment, Φ_j, is a member of segment cluster Σ_z.

$$\omega_{x\cup y,z} = \max_{\Phi_i \in (\Sigma_x \cup \Sigma_y)} \left(\max_{\Phi_j \in \Sigma_z} \omega_{i,j} \right) \tag{5}$$

3.4.3 Terminating Hierarchical Clustering

Unlike traditional hierarchical clustering, the Mixed-Bag Solver does not necessarily continue merging the segment clusters until only a single cluster remains. Rather, the solver continues clustering until the maximum similarity between any of the remaining clusters drops below a predefined threshold. In this thesis, a minimum inter-cluster similarity of 0.1 provided sufficient clustering accuracy, without merging unrelated segments.

The number of segment clusters remaining at the end of hierarchical clustering represents the Mixed-Bag Solver's estimate of the number of ground-truth inputs. The segment clusters are passed to the next stage to determine the seed pieces for the final output puzzles.

3.5 Final Seed Piece Selection

Most modern jigsaw puzzle solvers [14, 15, 16] rely on a kernel growing model, where a kernel is a partial assembly of one or more pieces. In Chapter 2, it was explained that Paikin & Tal select the puzzle seeds using a greedy condition at run time. Hence, their algorithm often picks suboptimal seeds (e.g., pieces from the same input puzzle are selected as seeds for multiple output puzzles).

In contrast, through the combination of segmentation and hierarchical

clustering, the Mixed-Bag Solver partitions the input pieces into disjoint segment clusters, with each one roughly approximating a single input puzzle. As such, the Mixed-Bag Solver selects a single piece from each segment cluster to be used as the seed of a puzzle during final segment. Within a given segment cluster, the Mixed-Bag Solver uses the same approach proposed by Paikin & Tal wherein the selected seed must have best buddies on each of its sides and each of its best buddies must also best buddies on each of their sides. This approach of selecting seeds using segment clusters provides vastly superior results as shown in Section 5.2.

3.6 Final Assembly

Once the seed pieces have been selected from the segment clusters, they are used as the initial kernels for the solver outputs. Assembly then proceeds simultaneously across all boards normally. The fully-assembled boards, with all pieces placed, are the Mixed-Bag Solver's final output.

CHAPTER 4

Quantifying and Visualizing the Quality of a Mixed-Bag Solver Output

Modern jig swap puzzle solvers are not able to perfectly reconstruct the ground-truth input(s) in many cases. As such, quantifiable metrics are required to objectively compare the quality of outputs from different solvers. Cho *et al.* [13] defined two such metrics, namely direct accuracy and neighbor accuracy. These metrics have been used by others including [15, 14, 16, 22, 17]. This chapter discusses the existing quality metrics and outlines a set of enhancements to make these metrics more applicable to Mixed-Bag puzzles. This thesis also proposes advanced metrics for quantifying the best buddy attributes of an image. The final two sections of this chapter outline new standards to visualize the quality of solver accuracy as well as the best buddy profile of images.

4.1 Direct Accuracy

Direct accuracy is a relatively naïve quality metric; it is defined as the fraction of pieces placed in the same location in both the ground-truth (i.e., original) and solved images with respect to the total number of pieces. Equation (6) shows the formal definition of direct accuracy (DA), where n is the number of pieces and c is the number of pieces in the solved image that are placed in their original (i.e., correct) location. A solved image is referred to as ''perfectly reconstructed'' if the location (and rotation, if applicable) of all pieces match the original image (i.e., $DA = 1$).

$$DA = \frac{c}{n} \tag{6}$$

This thesis proposes two new direct accuracy metrics, Enhanced Direct Accuracy Score (EDAS) and Shiftable Enhanced Direct Accuracy Score (SEDAS), which are specifically tailored to address Mixed-Bag puzzles. The metrics are described in the following two subsections; the complementary relationship between

EDAS and SEDAS is described in the third subsection.

4.1.1 Enhanced Direct Accuracy Score

The standard direct accuracy metric does not account for the possibility that there may be pieces from multiple input puzzles in the same solver output image. For a given puzzle, P_i, in the set of input puzzles P (i.e., $P_i \in P$) and a set of solved puzzles S where $S_j \in S$, EDAS is defined as shown in Equation (7). $c_{i,j}$ is the number of pieces from input puzzle P_i correctly placed (with no rotation for Type 2 puzzles) in solved puzzle S_j while n_i is the number of pieces in puzzle P_i. $m_{k,j}$ is the number of pieces from an input puzzle P_k (where $k \neq i$) that are also in S_j.

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

Standard direct accuracy (see Equation (6)) and EDAS are equivalent when solving a single puzzle. Moreover, like standard direct accuracy, a perfectly reconstructed puzzle will always have an EDAS of 1.

For Mixed-Bag puzzles, EDAS marks as incorrect any pieces from P_i that are not in S_j by dividing by n_i. Moreover, since pieces from P_i may have been placed in more than one output puzzle, EDAS is calculated as the maximum value across all solved puzzles, S. In addition, the summation of term $m_{k,j}$ penalizes for any puzzle pieces in S_j that are not from P_i. It is through the combination of these three techniques that EDAS takes into account both extra and missing pieces in the solver output.

It is important to note that EDAS is a score and not a measure of accuracy. While its value is bounded between 0 and 1 (inclusive), it is not specifically defined as the number of correct placements divided by the total number of placements since the denominator of Equation (7) is greater than or equal to the number of pieces in both P_i and S_j.

(a) Ground-truth image (b) Solver output

Figure 3: Solver output where a single misplaced piece catastrophically affects the direct accuracy

4.1.2 Shiftable Enhanced Direct Accuracy Score

Standard direct accuracy is vulnerable to shifts in the solved image where even very minor placement errors can cause the reported accuracy to drop to 0. Figure 3 shows a ground-truth image and an actual solver output when the puzzle boundaries were not fixed. Note that only a single piece is misplaced; this shifted all other pieces to the right one location causing the direct accuracy to drop to zero. Had this same piece been misplaced along either the right or bottom side of the image, the direct accuracy would have been largely unaffected. The fact that direct accuracy can give such vastly differing results for essentially the same error shows that direct accuracy has a significant flaw. This thesis proposes SEDAS to address the often misleadingly punitive nature of standard direct accuracy.

Equation (8) is the formal definition of SEDAS. d_{min} represents the Manhattan distance between the upper left corner of the solved image and the nearest puzzle piece. Similarly, L is the set of all puzzle locations within radius d_{min} (inclusive) of the upper left corner of the image. Given that l is a location in L, the term $c_{i,j}$ from Equation (7) has been changed to $c_{i,j,l}$ to denote that l is used as a custom reference point when determining the number of pieces correctly placed in the solved puzzle.

$$SEDAS_{P_i} = \max_{l \in L} \left(\max_{S_j \in S} \frac{c_{i,j,l}}{n_i + \sum_{k \neq i}(m_{k,j})} \right) \tag{8}$$

In the standard definition of direct accuracy proposed by Cho *et al.*, l is fixed at the upper left corner of the image. In contrast, SEDAS shifts this reference point within a radius of the upper left corner of the image in order to find a more meaningful value for direct accuracy.

Rather than defining SEDAS based on the distance d_{min}, an alternative approach is to use the location anywhere in the solved image, S_j, that maximizes Equation (8). However, that approach can take significantly longer to compute in particular when the solved puzzle has several thousand pieces. SEDAS balances the need for a meaningful direct accuracy score against computational efficiency.

4.1.3 Necessity of Using Both EDAS and SEDAS

While EDAS can be misleadingly punitive, it cannot be wholly replaced by SEDAS. Rather, EDAS and SEDAS serve complementary roles. First, EDAS must necessarily be calculated as part of SEDAS since the upper left corner location is inherently a member of the set L. (When the solved puzzle is not shifted, it is the only location in L.) Hence, there is no additional time required to calculate EDAS. What is more, by using EDAS along with SEDAS, some shifts in the solved image may be quantified (such as the one in Figure 3); this is not possible if only SEDAS is used.

4.2 Neighbor Accuracy

Cho *et al.* [13] defined neighbor accuracy as the ratio of puzzle pieces sides that are adjacent in both the original and solved images versus the total number of puzzle piece sides. Formally, let q be the number of sides each piece has (i.e., four in a jig swap puzzle) and n be the number of pieces. If a is the number of puzzle piece sides adjacent in both the ground-truth and solved images, then the neighbor accuracy,

NA, is defined as shown in Equation (9).

$$NA = \frac{a}{n \cdot q} \tag{9}$$

Unlike direct accuracy, neighbor accuracy is largely unaffected by shifts in the solved image since it considers only a piece's neighbors and not its absolute location. However, the standard definition of neighbor accuracy does not encompass cases where pieces from multiple inputs may be present in the same solver output.

4.2.1 Enhanced Neighbor Accuracy Score

Enhanced Neighbor Accuracy Score (ENAS) improves the neighbor accuracy metric by providing a framework to quantify the quality of Mixed-Bag puzzle outputs. Let n_i be the number of puzzle pieces in input puzzle P_i and $a_{i,j}$ be the number of puzzle piece sides that are adjacent in both P_i and solved output, S_j. If $m_{k,j}$ is the number of puzzle pieces in S_j from an input puzzle P_k (where $k \neq i$), then the ENAS for P_i is defined as shown in Equation (10).

$$ENAS_{P_i} = \max_{S_j \in S} \frac{a_{i,j}}{q(n_i + \sum_{k \neq i}(m_{k,j}))} \tag{10}$$

Similar to the technique described for EDAS in Section 4.1.1, ENAS divides by the number of pieces n_i in input puzzle P_i. By doing so, it effectively marks as incorrect any pieces from P_i that are not in S_j. What is more, by including in the denominator of Equation (10) a summation of all $m_{k,j}$, ENAS marks as incorrect any pieces not from P_i that are in S_j. The combination of these two factors allows ENAS to account for both extra and missing pieces.

4.3 Best Buddy Metrics

Chapter 2 explains that two puzzle pieces are best buddies on their respective sides if they are both more similar to each other than they are to any other piece. This thesis refers to a best buddy relationship as ‘‘adjacent’’ if the two pieces are

neighbors on their respective sides. In contrast, ''non-adjacent'' best buddies are not neighbors. Note that it is also possible that a piece has no best buddy at all on one or more sides.

Best buddy relationships have been used for segmentation [14], placement [16], and as an estimation metric [15]. The following subsections propose the first advanced best buddy metrics for both input and solved puzzles.

4.3.1 Interior and Exterior Non-Adjacent Best Buddies

If an image has fewer non-adjacent best buddies, then the best buddy relationships are a more accurate determiner of puzzle piece adjacency. It is expected that a pair of best buddies are more likely to be non-adjacent if they have no neighbor at all (i.e., the piece(s) is next to an open location). This is because those puzzle piece sides have no true neighbor, leaving them more inclined to couple with an unrelated piece, which is often another piece's side with no neighbor. This is illustrated by the example in Section 4.5.

This thesis subcategorizes non-adjacent best buddies depending on whether they are interior (i.e., the puzzle piece's side has an actual neighbor) or exterior (i.e., the puzzle piece's side has no neighbor). Interior non-adjacent best buddies are generally more deleterious since they are more likely to affect both placement and segmentation.

4.3.2 Best Buddy Density

As mentioned previously, some puzzle pieces may not have a best buddy; however, no metric exists that quantifies an image's best buddy profile. As such, this thesis proposes Best Buddy Density (BBD) as defined by Equation (11), where b is the number of puzzle piece sides that have a best buddy. By dividing by the number of puzzle pieces, n, each of which has q sides,[1] BBD normalizes for the size of the

[1] In a jig swap puzzle, q is equal to 4.

input images. This bounds BBD between 0 and 1 (inclusive), with a higher best buddy density indicating that the puzzle pieces are more differentiated from one another. This equation can be adjusted to a more localized metric by considering only a subset of the pieces.

$$BBD = \frac{b}{n \cdot q} \tag{11}$$

Ideally, all adjacent puzzle piece sides would be best buddies, and there would be no exterior best buddies. In such cases, the best buddy density would actually be less than 1; the extent to which it would be below 1 is dependent on the puzzle dimensions.

4.4 Visualizing the Quality of Solver Outputs

In images with thousands of pieces, it is often difficult to visually determine the location of individual pieces that are incorrectly placed. The following two subsections describe the standards developed as part of this thesis for visualizing direct and neighbor accuracy.

4.4.1 Visualizing EDAS and SEDAS

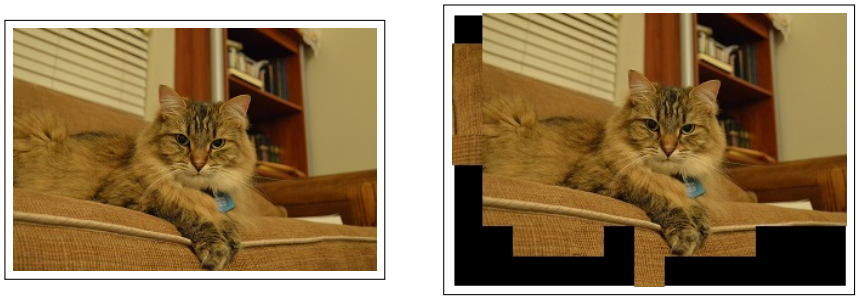
In standard direct accuracy, EDAS, and SEDAS, each puzzle piece is assigned a single value (i.e., correctly or incorrectly placed). As such, the direct accuracy visualization represents each puzzle by a square filled with one solid color. A refinement used in this thesis is to subdivide the ‘‘incorrect’’ placements into a set of subcategories, namely in order of precedence: wrong puzzle, wrong location, and wrong rotation. Note that the ‘‘wrong puzzle’’ classification applies only to Mixed-Bag puzzles and occurs when a piece in the solver output is not from the puzzle of interest, P_i. Table 1 shows the colors assigned to puzzle pieces depending on their direct accuracy classification. Assuming no missing pieces in the ground-truth image, the ideal EDAS and SEDAS visualizations would have the same

Table 1: Color scheme for puzzles pieces in direct accuracy visualizations

Wrong Puzzle	Wrong Location	Wrong Rotation	Correct Location	No Piece Present

dimensions as the ground-truth input and only green squares.

Figure 4 shows a Type 2 solver output as well as its associated EDAS and SEDAS visualizations. Since four puzzle pieces were erroneously placed on the left of the image, almost all pieces had the wrong location according to EDAS; the only exception is a single piece that had the right location but wrong rotation. In contrast, almost all pieces have the correct location in the SEDAS representation; note that the piece in the correct location but wrong rotation in EDAS has the wrong location



(a) Ground-truth image (b) Type 2 solver output

(c) EDAS visualization (d) SEDAS visualization

Figure 4: Example solver output visualizations for EDAS and SEDAS

Table 2: Color scheme for puzzles piece sides in neighbor accuracy visualizations

Wrong Puzzle	Wrong Neighbor	Correct Neighbor	No Piece Present

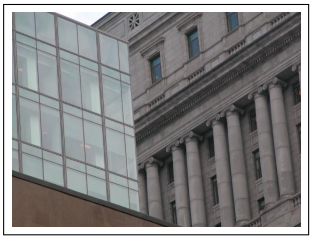
in SEDAS since the reference is shifted.

4.4.2 Visualizing ENAS

Jig swap puzzle pieces have four sides. As such, each piece in the ENAS visualization is divided into four isosceles triangles; the base of each triangle is along the puzzle piece's side whose neighbor accuracy is represented. The four isosceles triangles all share a common, non-base vertex at the piece's center. Table 2 defines the color assigned to each triangle depending on whether a piece's neighbors match in the ground-truth input and the solver output.

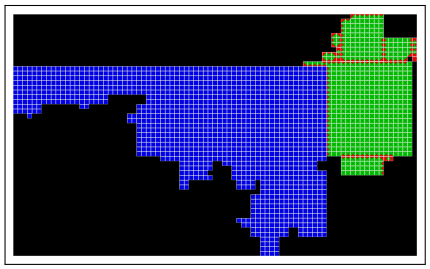
Figure 5 shows an actual output when solving a Mixed-Bag puzzle with two images. In this example, the puzzle of interest, P_i, is the glass and stone building while the other puzzle, P_k, is the rainforest house. All pieces that came from the rainforest house image are blue, despite being assembled correctly; this is because they are not from the puzzle of interest. In contrast, all pieces from the glass and stone building image that are placed next to their original neighbor are represented by green triangles while all incorrect neighbors, such as those bordering the rainforest house image, are represented by red triangles.

(a) Input image # 1 – rainforest house [23] Reproduced with permission



(b) Input image # 2 – building exterior [25]

(c) Solver output



(d) ENAS visualization

Figure 5: Example solver output visualization for ENAS

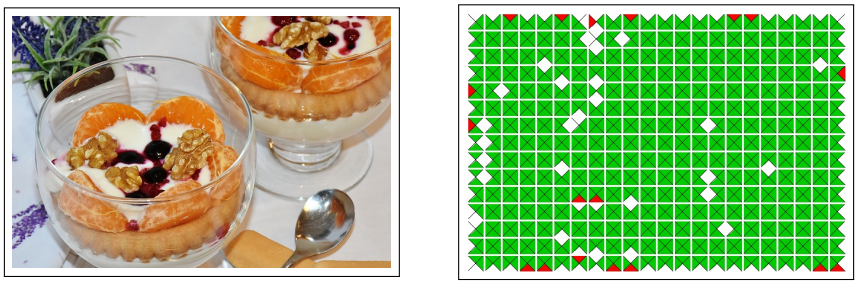
4.5 Visualizing Best Buddies

The visualization for best buddies is similar to that of neighbor accuracy where each piece is divided into four isosceles triangles with each triangle representing the piece's best buddy relationship with its neighbor. Table 3 defines the color scheme used to denote the three best buddy relationships outlined in Section 4.3.

Figure 6 shows an example image and its associated best buddy visualization. Despite having 16 times as many interior sides, the image in this figure still has 3 times more exterior, non-adjacent best buddies than interior ones.

Table 3: Color scheme for puzzles piece sides in best buddy visualizations

No Best Buddy	Non-Adjacent Best Buddy	Adjacent Best Buddy	No Piece Present



(a) Original image [24] (b) Best buddy visualization

Figure 6: Visualization of best buddies in an example image

CHAPTER 5

Experimental Results

A set of experiments were performed to compare the performance of the Mixed-Bag Solver and Paikin & Tal's algorithm. These experiments followed the standard test conditions collectively used by [13, 14, 17, 15, 16]. For example, each square puzzle piece was 28 pixels wide. Likewise, all image information was represented using the LAB colorspace. What is more, only the more challenging Type 2 puzzles were investigated, meaning that piece location and rotation were unknown. Furthermore, the solvers were not provided any information concerning the dimensions of the ground-truth input(s).

The only difference in the two solvers' test conditions arises from the fact that Paikin & Tal's algorithm requires that the number of input puzzles be specified. In contrast, the Mixed-Bag Solver is not supplied any additional information beyond the puzzle pieces. This gives Paikin & Tal's algorithm a clear advantage.

To compare the performance of the Mixed-Bag Solver and Paikin & Tal's algorithm when provided multiple ground-truth inputs, this thesis used Pomeranz *et al.*'s benchmark containing twenty, 805-piece images [23]. In each test, a specified number of images (ranging from two to five) were randomly selected, without replacement, from the image pool. The two solvers' outputs were then compared. Table 4 shows the number of times each solver was run for a specific input puzzle count. As explained in Section 3.1.2, the execution time of Paikin & Tal's assembler can grow cubicly, especially if the best buddy density is low. As such, the solvers were run fewer times as the number of input puzzles increased.

5.1 Accuracy Determining the Number of Input Puzzles

For the Mixed-Bag Solver to provide meaningful outputs, it must be able to identify the number of ground-truth inputs. The first subsection discusses the solver's

Table 4: Number of solver iterations for each puzzle input count

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

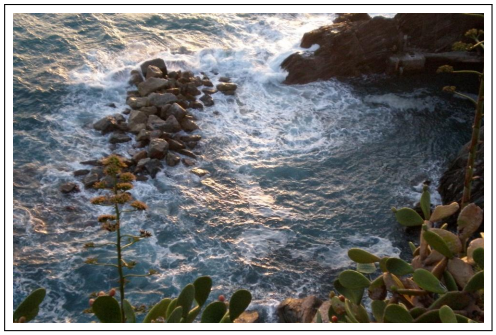
accuracy when provided only a single image. This is separated from the more general discussion as the algorithm's performance on a single image represents its accuracy ceiling. The algorithm's performance when solving two to five puzzles is discussed in a separate subsection.

5.1.1 Single Puzzle Solving

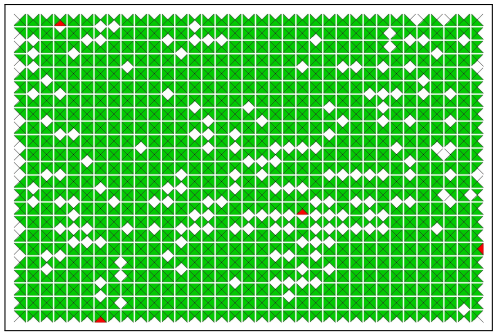
The Mixed-Bag Solver was able to correctly identify the single ground-truth input for 17 out of the 20 images (i.e., 85% accuracy) in the Pomeranz *et al.*'s data set. For the remaining three images, the Mixed-Bag Solver incorrectly found that the pieces came from two images, meaning that the error was at most only a single output puzzle.

Appendix B shows the three misclassified images and the associated Mixed-Bag Solver outputs. The figures in the appendix show that the solver struggles to correctly identify the number of input puzzles when an image has large areas with little variation (e.g., a clear sky, smooth water, etc.). Two example images from the Pomeranz *et al.* dataset are shown in Figure 7. The Mixed-Bag Solver was able to perfectly reconstruct image (a); in contrast, the Mixed-Bag Solver incorrectly determined that the pieces from image (b) came from two separate puzzles. The best buddy visualizations in Figure 7 shows that image (a) has a significantly higher best buddy density than image (b) as well as fewer interior, non-adjacent best buddies. It is these two factors that most contributed to the Mixed-Bag Solver being unable to determine the number of ground-truth inputs for the three misclassified images.

It is important to note that the Mixed-Bag Solver's difficulty reconstructing

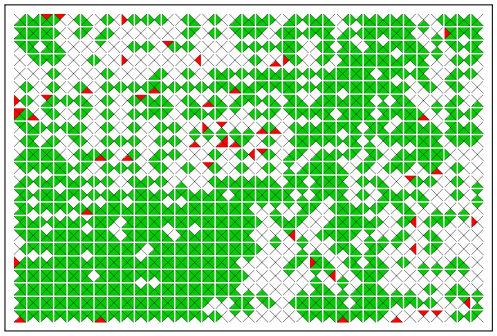


Ground-truth image (a) [23]
Reproduced with permission



Best buddy visualization of image (a)

Ground-truth image (b) [23]
Reproduced with permission



Best buddy visualization of image (b)

Figure 7: Comparison of best buddy density and interior non-adjacent best buddies for two images from the Pomeranz *et al.* 805 piece data set.

images with low best buddy density is actually an artifact of the assembler. Paikin & Tal mentioned in [16] that their algorithm may yield ‘‘unsatisfactory results’’ on such images.

5.1.2 Multiple Puzzle Solving

As mentioned previously, the Mixed-Bag Solver was tested by randomly selecting a specified number of images, without replacement, from Pomeranz *et al.*'s 805 piece data set. Figure 8 illustrates the Mixed-Bag Solver's performance in identifying the number of input puzzles when passed multiple images. A correct

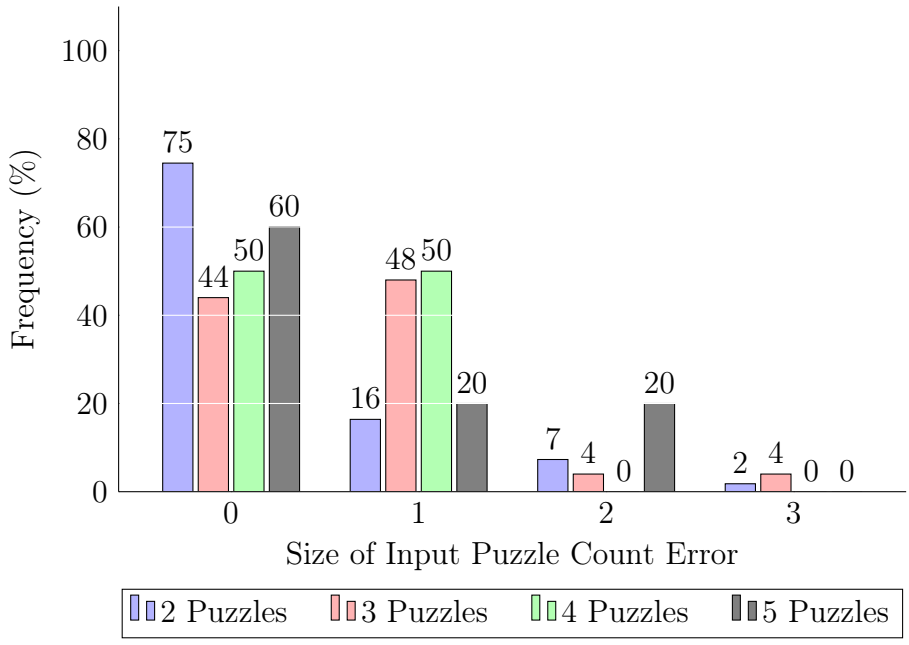


Figure 8: Mixed-Bag Solver's input puzzle count error frequency

estimation of the number of puzzles would represent an error of "0" in the figure. Similarly, an overestimation of a single puzzle (e.g., the solver identified four puzzles when only three were provided as an input) would represent an error of "1." Across all experiments, the Mixed-Bag Solver never underestimated the number of input puzzles; what is more, it never overestimated the number of input puzzles by more than 3.

In this set of experiments, the Mixed-Bag solver correctly determined the number of input puzzles in 65% of the tests. Likewise, the solver overestimated the number of input puzzles by more than one in less than 8% of tests. Since the solver never underestimated the input puzzle count, it is clear that it is over-rejecting cluster mergers and/or creating very small clusters that are too isolated to merge with others. It is expected that this aspect of the solver's performance would be improved by reducing the minimum clustering threshold (see Section 3.4) as well as

Table 5: Comparison of the Mixed-Bag and Paikin & Tal Solvers' performance on multiple input puzzles

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%

increasing the minimum segment size (see Section 3.2.2).

5.2 Comparison of Solver Output Quality

As mentioned at the beginning of this chapter, images were randomly selected from the Pomeranz *et al.* data set and passed to both the Mixed-Bag Solver and Paikin & Tal's algorithm. Table 5 and Figure 9 show the quantified quality of the outputs generated by both solvers for varying input puzzle counts. The three metrics used are the mean Shiftable Enhanced Direct Accuracy Score (SEDAS), mean Enhanced Neighbor Accuracy Score (ENAS), and the percentage of puzzles assembled perfectly (i.e., input and output puzzles are an identical match). The results for the Mixed-Bag Solver (MBS) are subdivided between the case when the number of input puzzles was correctly determined (denoted with a "†" in the table heading) versus all solver results (denoted with a "‡"). The reason for this distinction is that the former category represents the solver's performance ceiling if it were provided the input puzzle count.

Across all quality metrics and categories, the Mixed-Bag Solver significantly outperformed Paikin & Tal's algorithm. This is despite that only their algorithm was provided additional information concerning the number of input puzzles. Furthermore, unlike Paikin & Tal's algorithm, there was no significant decrease in

the Mixed-Bag Solver's performance as the number of input puzzles increased. In addition, there was not a substantial difference in SEDAS or ENAS if the Mixed-Bag Solver incorrectly estimated the number of input images; this indicates that the extra puzzles generated were relatively insignificant in size.

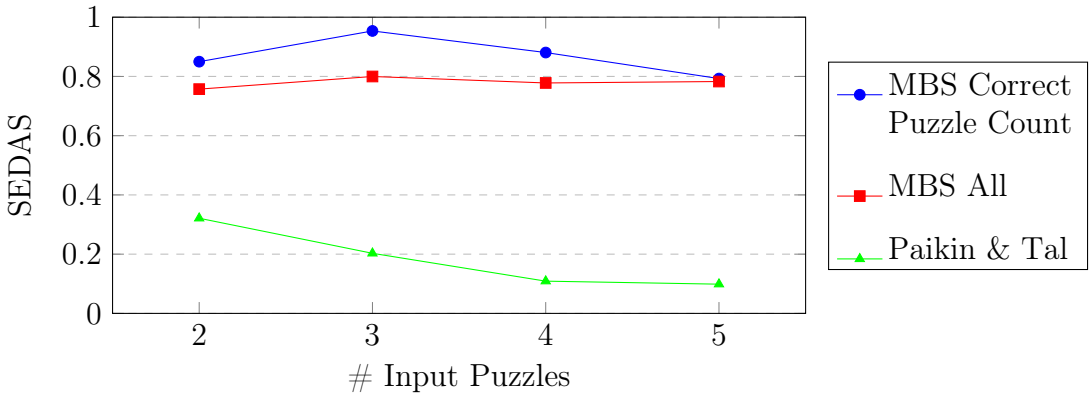
5.3 Ten Puzzle Solving

Paikin & Tal's algorithm was shown in [16] to be able to solve up to five images simultaneously; this represents the most in the current literature. In contrast, this thesis' solver has been shown to work on up to 10 puzzles simultaneously, which is double the current state of the art.

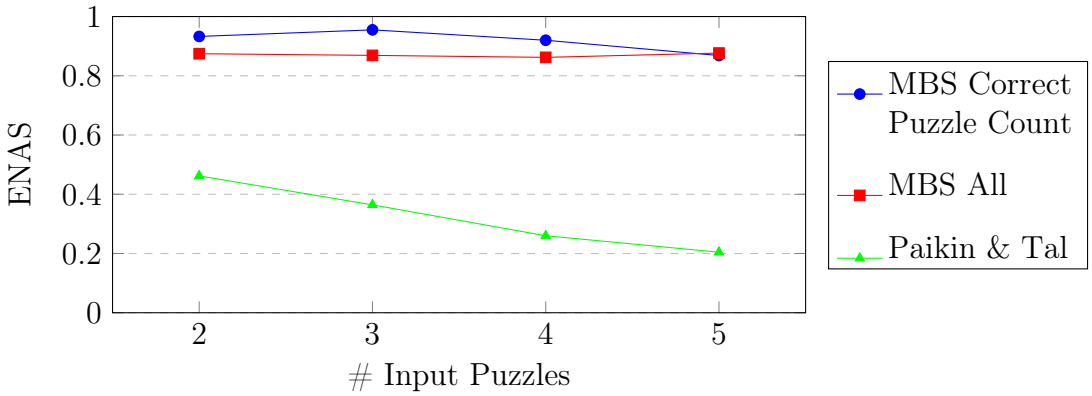
Appendix C contains the set of ten images that were input into both the Mixed-Bag Solver (MBS) and Paikin & Tal's algorithm. The comparison of their respective performance is shown in Table 6. Despite the Mixed-Bag Solver receiving less information, it scored greater than 0.9 for both Shiftable Enhanced Direct Accuracy Score (SEDAS) and the Enhanced Neighbor Accuracy Score (ENAS) on all puzzles. In contrast, Paikin & Tal's algorithm only exceeded a SEDAS and ENAS of 0.9 for image (f); their algorithm particularly struggled to select puzzle seeds with the starting pieces of nine of the output puzzles coming from just three of the input images. This experiment also shows that the Mixed-Bag Solver has greater immunity than Paikin & Tal's algorithm to potential shifts in the solved output since only four of the Mixed-Bag Solver's outputs showed a shift that would affect EDAS while seven of Paikin & Tal's outputs were shifted.

Table 6: Comparison of the image shifting, SEDAS, and ENAS results for the 10 puzzle data set

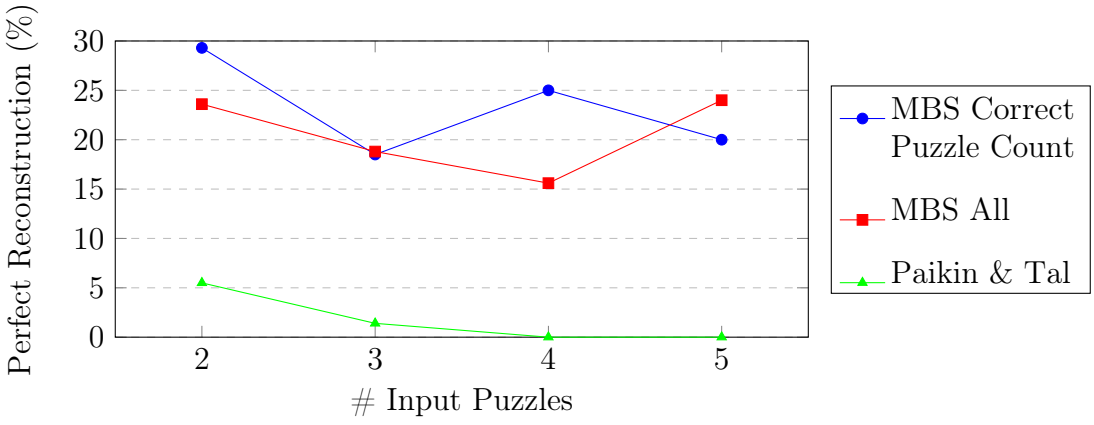
Image		Shifted		SEDAS		ENAS	
ID	# Pieces	MBS	Paikin	MBS	Paikin	MBS	Paikin
(a)	264	No	Yes	1.000	0.000	1.000	0.544
(b)	330	No	Yes	1.000	0.000	1.000	0.090
(c)	432	Yes	Yes	0.905	0.000	0.911	0.034
(d)	540	No	No	0.978	0.526	0.975	0.509
(e)	540	No	No	1.000	0.059	1.000	0.327
(f)	540	Yes	No	0.978	0.943	0.917	0.931
(g)	805	No	Yes	0.997	0.000	0.990	0.077
(h)	805	Yes	Yes	0.958	0.000	0.967	0.070
(i)	805	No	Yes	1.000	0.000	1.000	0.311
(j)	805	Yes	Yes	0.998	0.000	0.990	0.073



(a) Shiftable Enhanced Direct Accuracy Score (SEDAS)



(b) Enhanced Neighbor Accuracy Score (ENAS)



(c) Percentage of puzzles perfectly reconstructed

Figure 9: Performance of the Mixed-Bag and Paikin & Tal Solvers with multiple input puzzles

CHAPTER 6

Conclusions and Future Work

This thesis presented a fully automated solver for Mixed-Bag jigsaw puzzles. The solver outperforms the current state of the art both in terms of solution quality and also the maximum number of puzzles it can solve simultaneously. What is more, unlike the state of the art, it requires no externally supplied information beyond the set of puzzle pieces.

Opportunities exist to improve the Mixed-Bag Solver's performance. First, the assembler places a ceiling on the quality of the solver outputs. This solver is largely independent of the assembler used, meaning that the solver's performance will improve as better assemblers are proposed. As such, an improved assembler that uses multiple best buddies to prioritize placement is currently under development. What is more, this new assembler addresses some of the performance limitations of Paikin & Tal's algorithm for images with low best buddy density.

In addition, the threshold for hierarchical clustering is currently set to a fixed value. It is expected that a dynamic approach may improve the clustering overall and in turn the solver's performance.

Lastly, it was explained in Section 3.3.2 that stitching pieces are always members of saved segments. In some cases, the mini-assembly may not actually expand the segment, which would prevent segment clustering. As such, stitching may improve if pieces not assigned to a segment are also used since these pieces may be more likely to bridge inter-segment gaps.

LIST OF REFERENCES

[1] A. D. Williams, *Jigsaw Puzzles: An Illustrated History and Price Guide.* Wallace-Homestead Book Co., 1990.

[2] A. D. Williams, *The Jigsaw Puzzle: Piecing Together a History.* Berkley Books, 2004.

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

[4] T. Altman, ''Solving the jigsaw puzzle problem in linear time,'' *Applied Artificial Intelligence*, vol. 3, no. 4, pp. 453–462, Jan. 1990.

[5] 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.

[6] 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.

[7] 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.

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

[9] 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.

[10] 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.

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

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] D. Sholomon, O. David, and N. S. Netanyahu, ''Datasets of larger images and GA-based solver's results on these and other sets,'' http://u.cs.biu.ac.il/~nathan/Jigsaw/, 2013, (Accessed on 05/01/2016).

[19] P. S. Foundation, ''Comparing python to other languages | python.org,'' https://www.python.org/doc/essays/comparisons/, (Accessed on 10/09/2016).

[20] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*, 3rd ed. The MIT Press, 2009.

[21] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining, (First Edition)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2005.

[22] 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.

[23] 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).

[24] H. Braxmeier and S. Steinberger, ‘‘Pixabay,’’ https://pixabay.com/, (Accessed on 05/15/2016).

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

APPENDIX A

Example Outputs of a Single Segmentation Round

This appendix is provided as example to assist in visualizing the different outputs generated during segmentation. Figure A.10 shows the two ground-truth images that were input into the Mixed-Bag Solver for this example. As explained in Section 3.2, pieces from these images are assembled as if they had come from a single puzzle; Figure A.11 is the assembler output for the first round of segmentation. Figure A.12 shows the segments (they are colored to make them easier to identify) identified in the assembler output. For each segment, the pieces that are further away from an open location are lighter in color while those closer to a boundary are darker. Although not strictly a part of segmentation, the stitching piece(s) that would have resulted from each segment (assuming it exceeded the minimum size) are marked with white crosses as a reference.

Figure A.13 is the best buddy visualization of the assembler output. Note that the right and left sides of image (a) have stripes of best buddies that extend only in the horizontal direction. All of the pieces in these stripes are articulation points. As such, they were removed from the main segment in the center of the image as described in Section 3.2.3.

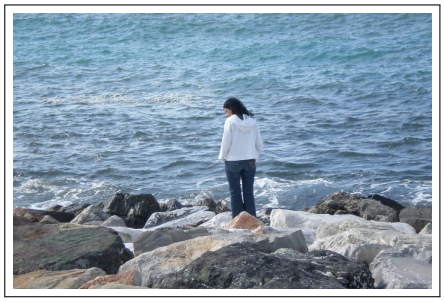


Image (a) – 805 pieces [23]
Reproduced with permission

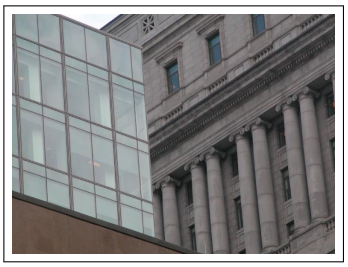


Image (b) – 540 pieces [25]

Figure A.10: Ground-truth images used in the segmentation example

Figure A.11: Assembler output of a single puzzle after the first segmentation round

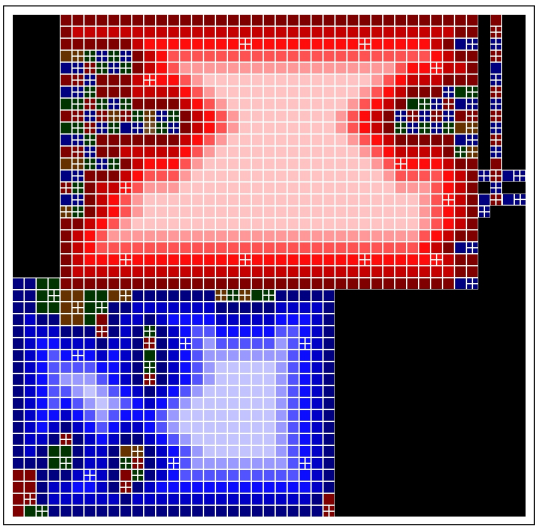


Figure A.12: Segmentation of the assembler output with marking of the articulation points and the lightness of piece coloring dependent on the distance to the nearest open location

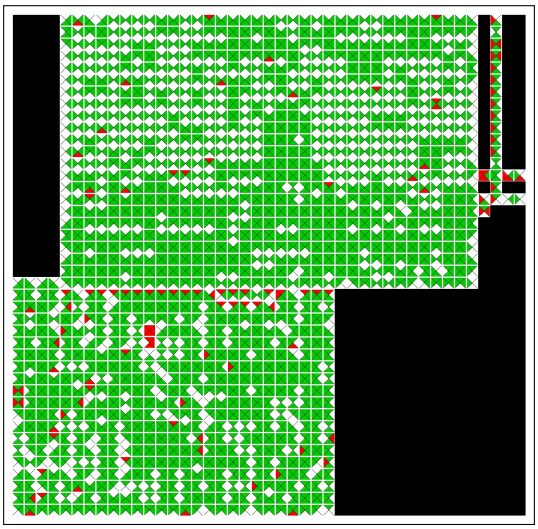


Figure A.13: Best buddy visualization of the assembler output

APPENDIX B

Incorrectly Classified Single Image Puzzles

To determine the Mixed-Bag Solver's performance ceiling, twenty images from Pomeranz *et al.*'s 805 piece dataset were individually input into the solver. The Mixed-Bag Solver correctly identified that there was only a single ground-truth input for 17 out of the 20 images. Figure B.14 shows the three misidentified images, and Figure B.15 contains the Mixed-Bag Solver's output for these images. All three images have large areas with little variation (e.g., a clear sky, smooth water). Paikin & Tal note in [16] that their assembler does not perform well on such images. Therefore, it is expected the Mixed-Bag Solver's performance on these images would improve if a different assembler is used.

Input image (a) [23]

Input image (b) [23]

Input image (c) [23]

Figure B.14: 805 piece images that were incorrectly identified by the Mixed-Bag Solver. Reproduced with permission from Pomeranz *et. al.*

Solver output (a–1)

Solver output (a–2)

Solver output (b–1)

Solver output (b–2)

Solver output (c–1)

Solver output (c–2)

Figure B.15: Mixed-Bag Solver outputs for the incorrectly identified images

APPENDIX C

Ten Puzzle Results

Figures C.16 and C.17 contain a set of 10 images of 5 different sizes that are made up of more than 5,800 total pieces. These images were input into both the Mixed-Bag and Paikin & Tal solvers; this experiment represents twice as many puzzles as Paikin & Tal solved in [16].

Figures C.18 and C.19 show the Mixed-Bag Solver outputs for this test set. Four of the images (e.g., (a), (b), (e), (i)) are perfectly reconstructions. The rest have only a small percentage of pieces out of place. This is shown in the SEDAS visualizations in Figures C.20 and C.21. The Mixed-Bag Solver's output quality for these images is comparable to that of Paikin & Tal's algorithm when it solves these images individually.

Image (a) – 264 pieces [24]

Image (b) – 330 pieces [24]

Image (c) – 432 pieces [25]

Image (d) – 540 pieces [25]

Image (e) – 540 pieces [25]

Image (f) – 540 pieces [25]

Figure C.16: First set of six images in the 10 image test set

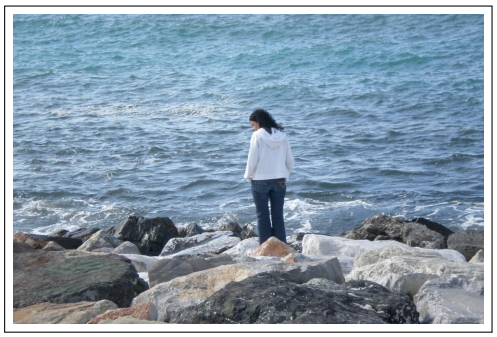


Image (g) – 805 pieces [23]

Image (h) – 805 pieces [23]

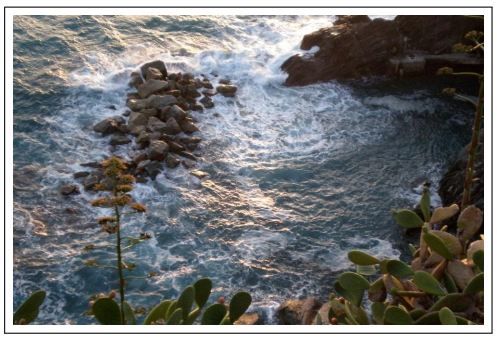


Image (i) – 805 pieces [23]

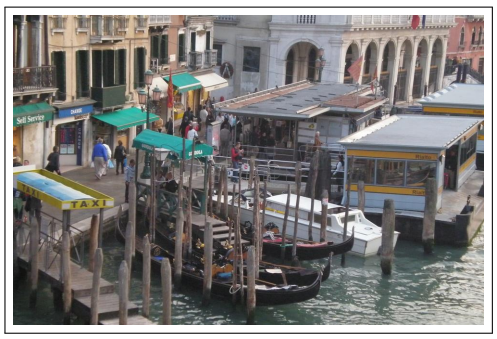


Image (j) – 805 pieces [23]

Figure C.17: Second set of four images in the 10 image test set. Reproduced with permission from Pomeranz *et. al.*

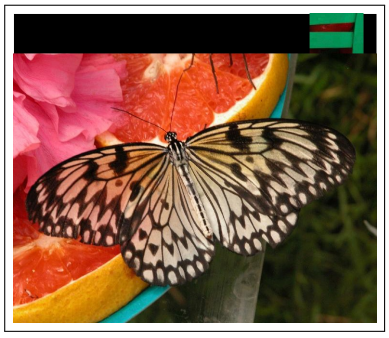
Reconstructed image (a)

Reconstructed image (b)

Reconstructed image (c)

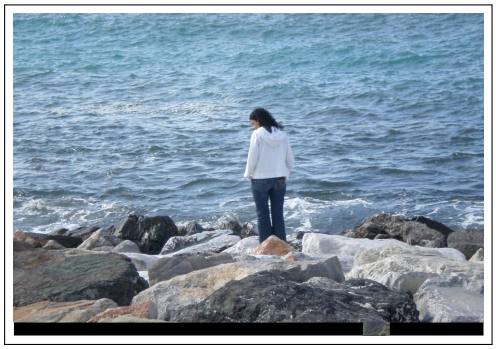
Reconstructed image (d)

Reconstructed image (e)



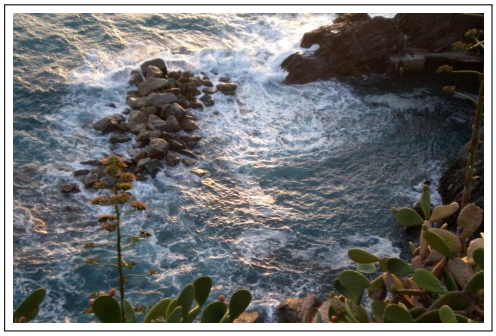
Reconstructed image (f)

Figure C.18: First set of six images output by the Mixed-Bag Solver for the 10 image test set

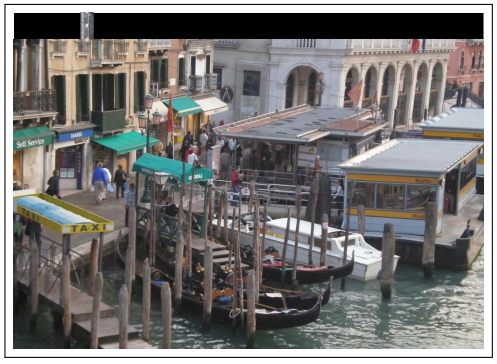


Reconstructed image (g)

Reconstructed image (h)

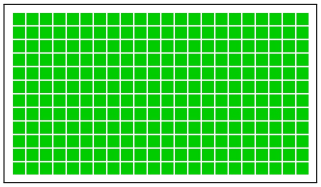


Reconstructed image (i)

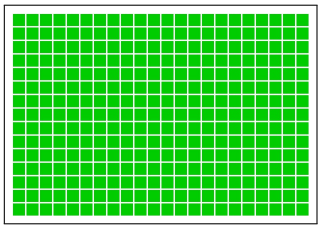


Reconstructed image (j)

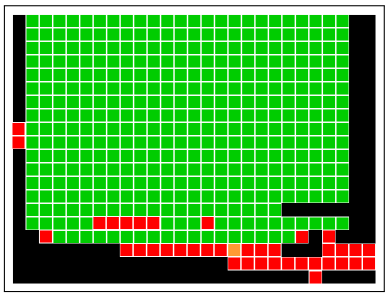
Figure C.19: Second set of four images output by the Mixed-Bag Solver for the 10 image test set



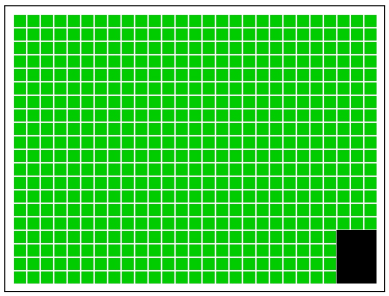
SEDAS visualization of image (a)



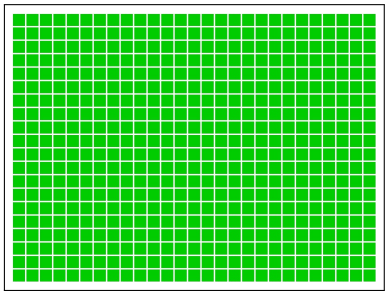
SEDAS visualization of image (b)



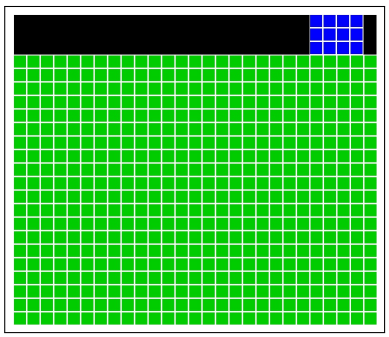
SEDAS visualization of image (c)



SEDAS visualization of image (d)

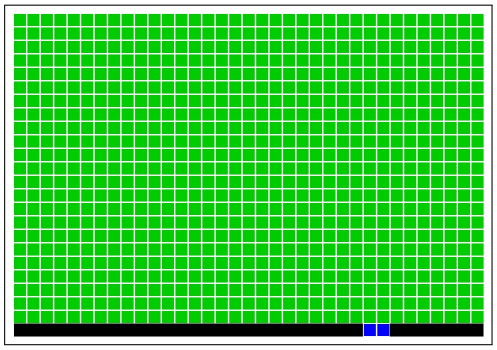


SEDAS visualization of image (e)

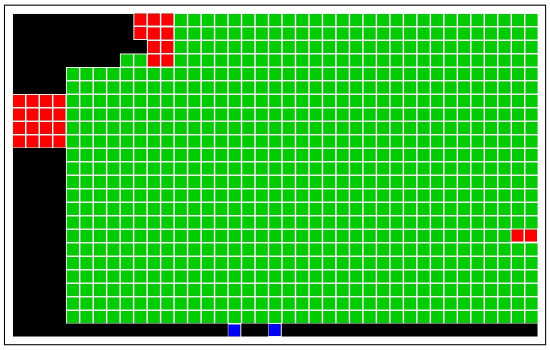


SEDAS visualization of image (f)

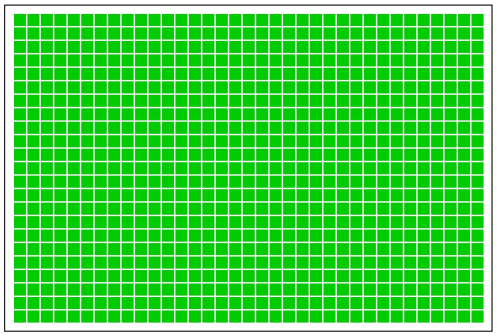
Figure C.20: First set of six SEDAS visualizations for the 10 image test set



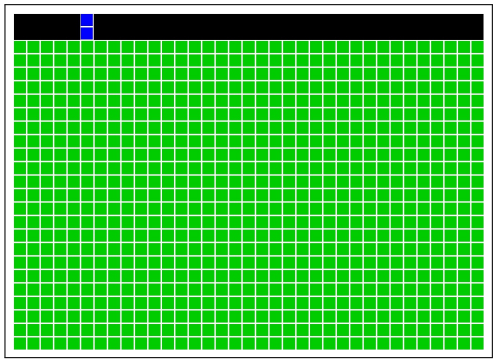
SEDAS visualization of image (g)



SEDAS visualization of image (h)



SEDAS visualization of image (i)



SEDAS visualization of image (j)

Figure C.21: Second set of four SEDAS visualizations for the 10 image test set