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

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

Square jigsaw puzzle solving is a variant of a standard jigsaw puzzles, wherein all pieces are equal-sized squares that must be placed adjacent to one another to match an original image. This thesis proposes the first algorithm that can solve multiple square jigsaw puzzles simultaneously without any information provided to it beyond the set of puzzle pieces. The algorithm has been able to assemble up to 10 images simultaneously with results comparable to the state-of-the-art solver working on each image individually.

This thesis also defines the first set of metrics specifically tailored for multiple puzzle solvers. It also outlines the first visualization standards for representing the quality of solver solutions.

DEDICATION

To my mother.

TABLE OF CONTENTS

CHAPTER

1 Introduction 1

2 Previous Work 3

3 Mixed-Bag Solver Overview 7

3.1 Assembler 8

3.1.1 Assembler Time Complexity 8

3.1.2 Assembler Implementation 9

3.2 Segmentation 9

3.2.1 The SegmentPuzzle Function 10

3.2.2 Articulation Points 11

3.3 Stitching 13

3.3.1 Stitching Pieces 13

3.3.2 Stitching Piece Selection 14

3.3.3 Quantifying Inter-Segment Relationships 17

3.4 Hierarchical Clustering of Segments 18

3.4.1 Calculating the Initial Similarity Matrix 18

3.4.2 Updating the Similarity Matrix via Single Linking 19

3.4.3 Terminating Hierarchical Clustering 19

3.5 Final Seed Piece Selection 20

3.6 Final Assembly 21

4 Quantifying and Visualizing the Quality of a Mixed-Bag Solver Output 22

4.1 Direct Accuracy 22
4.1.1 Enhanced Direct Accuracy Score 23
4.1.2 Shiftable Enhanced Direct Accuracy Score 24
4.1.3 The Necessity to Use Both EDAS and SEDAS 25
4.2 Neighbor Accuracy 26
4.2.1 Enhanced Neighbor Accuracy Score 26
4.3 Best Buddy Metrics 27
4.3.1 Interior and Exterior Best Buddies 28
4.3.2 Best Buddy Density 28
4.4 Visualizing Solver Output Quality and Best Buddies Relationships 29
4.4.1 Visualizing EDAS and SEDAS 29
4.4.2 Visualizing ENAS 31
4.4.3 Visualizing Best Buddies 32
5 Experimental Results 34
6 Conclusions and Future Work 35
LIST OF REFERENCES 37
APPENDIX
A Example Output of a Single Segmentation Round 40
B Ten Puzzle Solver Result 41

LIST OF TABLES

1 Color Scheme for Puzzles Pieces in Direct Accuracy Visualizations 30

2 Color Scheme for Puzzles Piece Sides in Neighbor Accuracy Visualizations 31

3 Color Scheme for Puzzles Piece Sides in Best Buddy Visualizations 32

LIST OF FIGURES

1	Jig Swap Puzzle Example	2
2	Components of the Mixed-Bag Puzzle Solver	7
3	Solver Output where a Single Misplaced Piece Catastrophically Affects the Direct Accuracy	25
4	Example Solver Output Visualizations for EDAS and SEDAS	30
5	Example Solver Output Visualization for ENAS	32
6	Visualization of Best Buddies in an Image	33
B.7	First Set of Six Images Comprising the 10 Image Test Set	42
B.8	Second Set of Four Images Comprising the 10 Image Test Set	43
B.9	First Set of Six Mixed-Bag Solver Images for the 10 Image Test Set	44
B.10	Second Set of Four Mixed-Bag Solver Images for the 10 Image Test Set	45
B.11	First Set of Six SEDAS Visualizations for the 10 Image Test Set	46
B.12	Second Set of Four SEDAS Visualizations for the 10 Image Test Set	47

CHAPTER 1

Introduction

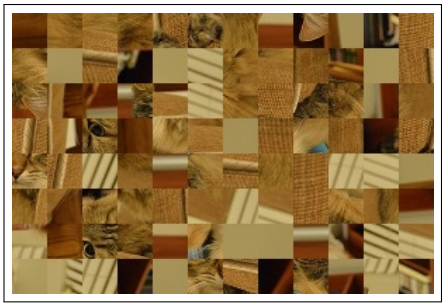
Jigsaw puzzles were first introduced in the 1760s when they were made from wood. The name "jigsaw" derives from the jigsaws that were used to carve the wooden pieces. The 1930s saw the introduction of the modern jigsaw puzzle where an image was printed on a cardboard sheet that was cut into a set of 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 pairwise compatibility between pieces is not a reliable metric for determining adjacency.

Jig swap puzzles are a specific type of jigsaw puzzle where all pieces are equal sized, non-overlapping squares.[1] An example of a jig swap puzzle is shown in Figure 1. These puzzles are substantially more challenging to solve than traditional jigsaw puzzles since piece shape cannot be considered when determining affinity between pieces. Rather, only the image information on each individual piece is used when solving the puzzle.

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

[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

reassembly of archaeological artifacts [6, 7], forensic analysis of deleted files [8], image editing [9], reconstruction of shredded documents [10], DNA fragment reassembly [11], and speech descrambling [12]. In most of these practical applications, the original, also known as ‘‘ground-truth,’’ input is unknown. This significantly increases the difficulty of the problem as the structure of the complete solution must be determined solely from the bag of component pieces.

This thesis describes the first fully-automated solver for the simultaneous assembly of multiple jigsaw puzzles. Unlike all previous solvers, this thesis' algorithm does not require any human input concerning the attributes of the input set of pieces (e.g., the the number of ground-truth puzzles). What is more, this thesis defines a set of new metrics specifically tailored to quantify the quality of outputs of multi-puzzle solvers. Lastly, it outlines a set of standards for visualizing the accuracy and quality of solver outputs.

CHAPTER 2

Previous Work

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

Cho *et al.* [13] proposed in 2010 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),’’ which are pieces whose position is fixed in the correct location before the solver begins. Cho *et al.*’s solver also required that the user specify the puzzle’s actual dimensions. 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] proposed 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 size of the puzzle. Pomeranz *et al.* also generalized and improved on Cho *et al.*’s inter-piece pairwise

distance measure by proposing a "predictive distance measure." Finally, Pomeranz *et al.* introduced the concept of "best buddies." Equation 1 formally defines the best buddy relationship between the side (e.g., top, left, right, bottom), s_x, of puzzle piece, p_i, and the side, s_y, of piece p_j. Note that $C(p_i, s_x, p_j, s_y)$ represents the compatibility between the two piece's respective sides.

$$\begin{gathered} \forall x_k \forall s_c, C(x_i, s_a, x_j, s_b) \geq C(x_i, s_a, x_k, s_c) \\ \text{and} \\ \forall x_k \forall s_c, C(x_j, s_b, x_i, s_a) \geq C(x_j, s_b, x_k, s_c) \end{gathered} \tag{1}$$

Best buddies have served as both an estimation metric for the quality of a solver output [15] as well as the foundation of some solvers' assemblers [16].

An additional key contribution of Pomeranz *et al.* is the creation of three image benchmarks. The first benchmark is comprised of twenty 805 piece images; the sizes of the images in the second and third benchmarks are 2,360 and 3,300 pieces respectively.

In 2012, Gallagher [17] formally categorized jig swap puzzles 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) is known. The orientation/rotation of each piece is also known, which means that there are exactly four pairwise relationships between any two pieces. A single anchor piece, with a known, correct, location is required with additional anchor pieces being optional. This type of puzzle is the focus of [13, 14].

- **Type 2 Puzzle**: This is an extension of a Type 1 puzzle, where pieces may be rotated in 90° increments (e.g., 0°, 90°, 180°, or 270°); in comparison to a Type 1 puzzle, this change alone increases the number of possible solutions by a factor of 4^n, where n is the number of puzzle pieces. What is more, no piece locations are known in advance, hence eliminating the use of anchor piece(s). 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 Puzzles**: The input set of pieces are from multiple puzzles, or there are extra pieces in the input set that belong to no puzzle. The solver may output either a single, merged puzzle, or it may separate the input pieces into disjoint sets that ideally align with the set of ground-truth puzzles. This type of puzzle is the primary focus of this thesis.

In 2013, Sholomon *et al.*[15] proposed a genetic algorithm-based solver for Type 1 puzzles. By moving away from the greedy approach 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 previous 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] published 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 that affects its performance for Mixed-Bag puzzles. For example, similar to all previous solvers, Paikin & Tal's algorithm is supplied the number of input puzzles. In many practical applications, this information may not be known, which limits the algorithm's potential usefulness.

Another limitation arises from the fact that their placer follows a single-pass, kernel growing approach in which a single piece is used as the seed of each output puzzle, and all pieces are placed around the expanding kernel. Because of that, poor seed selection can catastrophically degrade the quality of the solver output. Paikin & Tal's requirements for selecting a seed is that it be a "piece that is both distinctive and lies in a distinctive region." However, their specific criteria is only that the seed piece have best buddies on each of its sides and that each of its best buddies also have best buddies on each of their sides. Therefore, the selection of the seed is based off essentially 13 pieces. Since so few pieces determine the choice of a seed, it is common that the seeds of multiple output puzzles come 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 thesis' assembler. What is more, it is used as the baseline for performance comparison.

CHAPTER 3

Mixed-Bag Solver Overview

This thesis' Mixed-Bag Solver supports Type 1, Type 2, and Mixed-Bag puzzles. It consists of five distinct stages, namely: segmentation, stitching, hierarchical clustering of segments, seed piece selection, and final assembly. The flow of the algorithm is shown in Figure 2; the pseudocode for the solver, including the input(s) and output of each stage is shown in Algorithm 1.

The following subsections describe each of Mixed-Bag Solver's stages/subfunctions. It also discusses the assembler (not shown in Figure 2), which is a separate but associated component of the architecture.

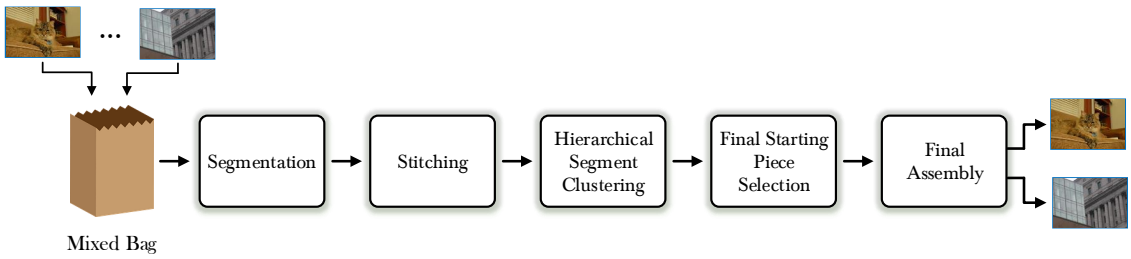


Figure 2: Components of the Mixed-Bag Puzzle Solver

Algorithm 1 Pseudocode for the Mixed Bag Solver

```
function MixedBagSolver(all_pieces)
    solved_segments ← Segmentation(all_pieces)
    overlap_matrix ← Stitching(solved_segments, all_pieces)
    clusters ← HierarchicalClustering(solved_segments, overlap_matrix)
    puzzle_start_pieces ← FindStartingPieces(clusters)
    solved_puzzles ← RunFinalAssembly(puzzle_start_pieces, all_pieces)
    return solved_puzzles
```

3.1 Assembler

The assembler assigns the placement (and optionally rotation) of the puzzle pieces in the solved puzzle. The Mixed-Bag Solver's architecture is largely independent of the particular assembler used. Hence, any improvements or modifications to the assembler can be directly incorporated into the Mixed-Bag Solver to improve the solver's performance. What is more, if a particular assembler performs better than others for a particular application, the assemblers can be interchanged. This provides the Mixed-Bag Solver with significant flexibility and upgradability to maximize performance across a wide range of applications.

For all experiments in this thesis, the assembler proposed proposed by Paikin & Tal [16]. As mentioned in Chapter 2, their assembler is the current state of the art and is one of the few algorithms that natively supports Mixed-Bag puzzles.

3.1.1 Assembler Time Complexity

Paikin & Tal's assembler relies on a set of inter-puzzle piece distance and similarity metrics. Similar to all other jig swap solvers, these distances are calculated between all pairs of possible pieces, making the time complexity of this stage at $O(n^2)$, where n is the number of puzzle pieces. If an input image has sufficient inter-piece variation, then assembly is expected to take $Theta(n\lg(n)$ as a heap is used to determine the next piece to be placed. However, if most pieces are sufficient similar that there are fewer best buddies (see Chapter 2, then assembly can be as slow as $O(n^3)$ as the inter-piece similarity may need to be recalculated after each piece is placed.

Assembly only occurs once in Paikin & Tal's algorithm. In the Mixed-Bag Solver, assembly is run at least once during segmentation stage (usually more) and is

run again during the final assembly stage. Hence, while the Mixed-Bag Solver will necessarily take longer to execute than Paikin & Tal's algorithm, the Mixed-Bag Solver's time complexity is the same.

3.1.2 Assembler Implementation

Paikin & Tal wrote their algorithm in Java, and as of this prublication, their source code has not been released. Hence, their algorithm was implemented as part of this thesis based off of the description in [16]. This thesis' implementation is in the Python programming language and is fully open-source. No execution time comparisons between their algorithm and the Mixed-Bag Solver are included with this thesis since Java is generally significantly faster than Python [19].

3.2 Segmentation

As detailed in Algorithm 1, segmentation stages takes as input only the bag of puzzle pieces created from the original images; unlike all other solvers to date, this algorithm takes no other inputs. The role of segmentation is to a basic provide structure to the unordered input. This is done by partitioning the pieces into disjoint sets, referred to here as segments. These segments are groups of puzzle pieces where there is a high degree of confidence that the pieces are assembled correctly.

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 regarding the input at this early stage. What is more, it is expected that pieces from the same input puzzle may be assigned to multiple disjoint segments. Section 3.4 describes how these

segments are merged using hierarchical clustering.

After the puzzle is assembled in a given round, the solved puzzle is then partitioned into segments; the segmentation procedure is described in Section 3.2.1. Assuming the largest segment exceeds the minimum allowed size[1], it is passed to the Stitching stage described in Section 3.3. Similarly, the term "α" in Algorithm 2 dictates the segments in a given segmentation round, other than the largest one, that are also passed to the next stage of the Mixed-Bag Solver. In this thesis, α was set to 0.5, meaning any segment that was at least half the size of the largest segment in that round is saved. This scalar value provides sufficient balance between finding the largest possible segments for analysis and limiting overall execution time.

Once a piece is assigned to a saved segment, it is removed from the set of unassigned pieces. Hence, those pieces will not be placed in the next segmentation round. Segmentation continues until all pieces have been assigned to sufficiently large segments, or no segment exceeds the minimum allowed segment size.

3.2.1 The SegmentPuzzle Function

The SegmentPuzzle function shown in Algorithm 3 is adapted from the kernel growing segmentation procedure proposed by Pomeranz *et al.*, where it was shown to have greater than 99.7% accuracy identifying genuine neighbors [14]. The kernel of each segment is a single seed piece.

Whenever a piece is added to a segment, the algorithm examines all of that piece's neighbors. Any of the adjacent pieces are added to the segment if they satisfy the "best buddy" criteria, as determined by the predicate IsBestBuddies. Pieces

[1]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 2 Pseudocode for the Segmentation Algorithm

```
1: function SEGMENTATION(all_pieces)
2:     saved_segments ← {}
3:     unassigned_pieces ← {all_pieces}
4:     loop
5:         solved_puzzle ← RUNSINGLEPUZZLEASSEMBLY(unassigned_pieces)
6:         puzzle_segments ← SEGMENTPUZZLE(solved_puzzle)

7:         max_segment_size ← maximum size of segment in solved_segments
8:         if max_segment_size < smallest_allowed then
9:             return saved_segments

10:        for each segment ∈ puzzle_segments do
11:            if |segment| > α × max_segment_size then
12:                add segment to saved_segments
13:                remove pieces in segment from unassigned_pieces
```

continue to be added to the segment until there are no more neighboring best buddies that have yet to be added. In Pomeranz *et al.*'s algorithm, no changes were made to the segment after it reached the maximum size. Their approach is sufficient when solving only a single puzzle at a time. However, it is common in Mixed-Bag puzzles that two or more correctly assembled segments from different ground-truth inputs to be be joined into a single cluster by very tenuous linking, usually in the form of narrow bridges no wider than a single piece. As described in the next section, the Mixed-Bag Solver post processes each segment after it has grown to its maximum size to prevent erroneous segment merging.

3.2.2 Articulation Points

A segment can be modeled as a graph with a single connected component. The individual puzzle pieces represent the vertices while the edges are the best buddy relationships between adjacent. An articulation point is any vertex (i.e., puzzle piece) in the graph whose removal increases the number of connected components. The

Algorithm 3 Pseudocode to Segment 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_piece of piece do
12:                if IsBestBuddies(neighbor_piece, piece) then
13:                    add neighbor_piece 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, seed_piece)
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
```

Mixed-Bag Solver identifies the articulation points using the the algorithm proposed by [20]; once they have been identified, all articulation points are removed from the segment and returned to the set of unassigned pieces. This necessarily causes some other pieces to become disconnected from the segment's seed. Any disconnected pieces are also removed from the segment and marked as unassigned.

3.3 Stitching

As discussed previously, a segment represents an ordering of pieces where there is a particularly high degree of confidence that the placement is correct. In some areas of an image (e.g., a sky where there is little variation between pieces), the segmenter may not have high confidence that the puzzle is assembled correctly. This often leads to a single ground-truth image being comprised of multiple segments. Since the Mixed-Bag Solver is not supplied with the number of input puzzles, 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.

It is expected that two segments that were adjacent in their ground-truth image would eventually merge if one segment were allowed to expand. Since it is not known in which relative direction adjacent segment(s) may be located, the segment should be allowed to grow in all directions; however, the segment should not be forced to expand in a certain direction as it may lead to the formation of erroneous inter-segment coupling. This concept serves the foundation of the inter-segment stitching used by the Mixed-Bag Solver. The stitching process is described in the following subsections.

3.3.1 Stitching Pieces

As mentioned previously, a segment should be allowed, but not forced, to expand in all directions to try 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 it

Algorithm 4 Pseudocode for Selecting the Stitching Pieces in a Segment

```
1: procedure FindStitchingPieces(segment_pieces)
2:     FindPieceDistanceToOpen(segment_pieces)
3:     segment_stitching_pieces ← {}
4:     segment_grid_cells ← PartitionIntoGrid(segment, grid_cell_width)

5:     for each grid_cell ∈ segment_grid_cells do
6:         if HasPieceAdjacentToOpen(grid_cell) then
7:             candidates ← set of pieces with distance_to_open closest to target
8:             stitching_piece ← piece ∈ candidates closest to center of grid_cell

9:             add stitching_piece to segment_stitching_pieces

10:    return segment_stitching_pieces
```

only places a limited number of pieces.[2] The seed piece of each of these assemblies is referred to as a ‘‘stitching piece’’ since they serve the role of ‘‘stitching’’ together associated segments.

3.3.2 Stitching Piece Selection

Poor selection of stitching pieces can cause more subtle inter-segment links to be missed or lead to unnecessary execution time. Algorithm 4 outlines the procedure used by the Mixed-Bag Solver to select the stitching pieces. It ensures that stitching pieces are sufficiently spaced from one another as well as from segment boundaries to minimize execution time while simultaneously ensuring sufficient coverage to detect subtle inter-segment relationships. The following two subsections describe implementation of this algorithm.

[2]This thesis' implementation of the Mixed-Bag Solver uses 100 as the number of pieces placed during a mini-assembly.

3.3.2.1 Spacing the 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 where no pieces are present. As such, stitching pieces are placed near all ‘‘open locations,’’ which are all valid puzzle locations that has either a piece from a different segment or no piece at all.

If a stitching piece is too close to the boundary of a segment, erroneous coupling between unrelated segments may occur. Algorithm 4 invoked the function FindPieceDistanceToOpen to determine distance of each piece in the segment to the nearest open location; the implementation of this function is shown in Algorithm 5 is used by the Mixed-Bag Solver to determine the distance between each piece in the segment and the nearest open location.

FindPieceDistanceToOpen follows an iterative boundary tracing technique; hence, during each iteration of the **while** loop on line 5, all segment pieces whose distance to the nearest open location is equal to *distance_to_open* are explored. Therefore, any pieces explored in the first iteration of the **while** loop would have a distance of 1 to the nearest open while those explored in the second iteration would have distance 2, etc.

There are two primary reasons iterative boundary tracing was used for Algorithm 5. First, the algorithm is robust enough to handle segment voids as well as potential necking within the segment where two large segment components are joined by a narrower bridge. What is more, 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 ← {all 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:                    remove piece from 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 Spacing between Stitching Pieces

If stitching pieces are too close together, the outputs from several mini-assemblies will be almost identical. In contrast, if stitching pieces are too far apart, the segment may not be able to grow towards any segments.

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. This grid spans the entire segment starting from the piece in the upper left corner of the segment. 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

[3]Along the bottom and right boundary of the segment, some grid cells will be narrower than the target grid cell width if the segment dimensions are not evenly divisible by the target grid cell width.

the set of pieces (if any) whose distance to the nearest open location equals the target.[4] If no pieces satisfy that criteria, then the target value is decremented until at least one piece is identified. From amongst the set of candidates that satisfy the open distance to location criteria, the algorithm selects as the stitching piece the one that is closest to center of the grid cell.

3.3.3 Quantifying Inter-Segment Relationships

As mentioned previously, a mini-assembly will be performed for each stitching piece ζ_x in segment Φ_i where $\zeta_x \in \Phi_i$. The output from the mini-assembly, MA_{ζ_x}, will contain puzzle pieces from one or more segments. If the solver output includes pieces from a different segment, there is a significantly increased likelihood the two segments came from the same input puzzle.

Equation (2) defines an overlap coefficient based on the mini-assembly outputs to quantify extent of association between any two segments Φ_i and Φ_j. The intersection between the mini-assembly output and the corresponding segment needs to be normalized by the number of pieces in mini-assembly. It must also take into account the size of the corresponding segment since that can dictate in some cases dictate the maximum overlap if $|\Phi_j| < |MS_{\zeta_x}|$.

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

The outputs of the mini-assemblies will vary between segments based off their respective stitching piece selection as well as potentially the size of the segments. Hence, in most cases, the overlap coefficient is asymmetric meaning: $Overlap_{\Phi_i,\Phi_j} \neq Overlap_{\Phi_j,\Phi_i}$. Section 3.4.1 defines how the overlap coefficients are

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

Algorithm 6 Pseudocode for the Hierarchical Clustering of Segments

```
1: function HierarchicalClustering(solved_segments, overlap_matrix)
2:     segment_clusters = {}
3:     for each segment_i ∈ solved_segments do
4:         add new segment cluster Φ_i containing segment_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 Φ_i and Φ_j in segment_clusters
8:         Update the similarity matrix, Γ for the merged clusters
9:     return cluster_segments
```

normalized to quantify inter-segment similarity.

3.4 Hierarchical Clustering of Segments

Agglomerative hierarchical clustering is a bottom-up clustering algorithm where in each clustering round, two clusters are merged. Algorithm 6 shows the basic flow of the hierarchical clustering algorithm of the Mixed-Bag Solver; it is adapted from [21].

The only inputs to the hierarchical clustering algorithm are the segments found in the segmentation stage and the Segment Overlap Matrix from the Stitching stage.

3.4.1 Calculating the Initial Similarity Matrix

The Segment Overlap Matrix is a form of hollow matrix, where all elements in the matrix, except those along the diagonal, are populated with meaningful values. In contrast, hierarchical clustering merges segments using a triangular, similarity matrix. Equation (3) defines the similarity, $\omega_{i,j}$ between any two clusters Φ_i to Φ_j.

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

If there are n solved segments found during segmentation, then the initial similarity matrix Γ is size n by n. Each element in Γ is defined by Equation (4). Both i and j are bounded between 1 and n inclusive. What is more, all elements in Γ are normalized between 0 and 1, also inclusive.

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

3.4.2 Updating the Similarity Matrix via Single Linking

The Mixed-Bag Solver uses the Single Link version of hierarchical clustering. Hence, the similarity between any two cluster segments is defined as the similarity between the two most similar segments in either cluster. This approach is required because two segments clusters may only be adjacent along the border of two of the composite segments.

Equation (5) defines the similarity between any a merged cluster containing segment clusters, Σ_x and Σ_y, 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.

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

3.4.3 Terminating Hierarchical Clustering

Unlike traditional hierarchical clustering, the Mixed-Bag Solver does not always continuing merging the segment clusters until only a 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 segments clusters remaining at the end of hierarchical clustering represents the expected number of ground-truth images provided to the solver. The segment clusters are then passed to the next stage to determine the final seed pieces for each output puzzle.

3.5 Final Seed Piece Selection

Most of the 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. As such, once the Mixed-Bag Solver has determined the expected number of input puzzles via hierarchical clustering, the algorithm then selects the seed piece for each of the output solutions.

In Chapter 2, it was explained that Paikin & Tal select a piece as an output puzzle's seed if seed piece to be the ‘‘is both distinctive and lies in a distinctive region.'' They apply this criteria greedily during runtime. 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, the combination of segmentation and hierarchical clustering in the Mixed-Bag Solver partitions the set of input pieces into disjoint sets, with each set roughly approximating a single solved puzzle. Therefore, the Mixed-Bag Solver selects the seed for each output puzzle from the members of its associated segment cluster. The algorithm uses the same approach as Paikin & Tal wherein the selected seed from the segment cluster is a ‘‘piece that is both distinctive

and lies in a distinctive region.'' However, since this selection is not made greedily and instead uses the segment clusters, the quality of the selection is superior.

3.6 Final Assembly

Once the seed pieces have been selected from the segment clusters, the seeds are used as the initial kernel for the solver outputs. Assembly then proceeds simultaneously across all boards. 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 in most 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 section describes the existing quality metrics, their weaknesses, and proposes enhancements to these metrics to make them more meaningful for Type 2 and Mixed-Bag puzzles. This thesis also proposes advanced metrics for quantifying the best buddy attributes of an image.

In the final section, tools developed as part of this thesis to visualize the solver output quality are discussed.

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 image with respect to the total number of pieces. Equation (6) shows the formal definition of direct accuracy (DA), where n is the total number of pieces and c is the number of pieces in the solved image that are placed in their original (i.e., correct) location.

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

Direct accuracy is vulnerable to shifts in the solved image where even a few misplaced pieces can cause a significant decrease in accuracy. As shown in Figure 3, this can be particularly true when the ground-truth image's dimensions are not known by the solver.

This thesis proposes two new direct accuracy metrics namely: Enhanced Direct Accuracy Score (EDAS) and Shiftable Enhanced Direct Accuracy Score (SEDAS). They 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 a 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 is in S, Enhanced Direct Accuracy Score (EDAS) is defined as shown in Equation (7). c_{ij} 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.

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

Standard direct accuracy (see Equation (6)) and EDAS are equivalent when solving a single puzzle. For Mixed-Bag solvers, EDAS necessarily marks as incorrect any pieces from P_i that are not in S_j by dividing by n_i. What is more, the summation of $m_{k,j}$ in EDAS penalizes for any puzzle pieces in the solver output S_j

that are not from input puzzle P_i. Therefore, 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.

4.1.2 Shiftable Enhanced Direct Accuracy Score

As mentioned previously, the direct accuracy decreases if there are shifts in the solved image. In many cases such, direct accuracy is overly punitive.

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 fundamental flaw. This thesis proposes Shiftable Enhanced Direct Accuracy Score (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 placed puzzle piece. Similarly, L is the set of all puzzle piece locations within radius d_{min} (inclusive) of the upper left corner of the image. Given that l is a location in L that is used as the reference point for determining the absolute location of all

(a) Ground-Truth Image (b) Solver Output

Figure 3: Solver Output where a Single Misplaced Piece Catastrophically Affects the Direct Accuracy

pieces, then SEDAS is defined as shown in Equation (8).

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

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 off the distance d_{min}, an alternative approach is to use the point anywhere in the image 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 computation efficiency.

4.1.3 The Necessity to Use 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. Hence, there is no additional time required to calculate EDAS. What is more, by continuing to use EDAS along with SEDAS, some shifts in the solved image may be quantified; this would not be possible if SEDAS was used alone.

4.2 Neighbor Accuracy

Cho *et al.* [13] defined neighbor accuracy as the ratio of the number of puzzle piece sides adjacent to the same piece's side in both the ground-truth and solved image 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).

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

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 cannot encompass the case where pieces from multiple input puzzles 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 solver outputs.

Let n_i be the number of puzzles pieces in the input puzzle P_i and $a_{i,j}$ be the

number of puzzle piece sides adjacent in P_i and S_j. If $m_{k,j}$ is the number of puzzle pieces from an input puzzle P_k (where $k \neq i$) in S_j, then the ENAS for P_i is defined as shown in Equation (10).

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

In the same fashion as 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 a summation of all $m_{k,j}$ in the denominator of (10), 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 extra and misplaced 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 pieces. 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. 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 citepaikin 2015, and as an estimation metric [15]. To date, no specific metrics or additional subclassifications of best buddy relationships has been proposed. The following subsections propose new metrics for studying the best buddy profile of both input and solved puzzles.

4.3.1 Interior and Exterior Best Buddies

If an image has fewer non-adjacent best buddies, it means that best buddy relationships become 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 are have no neighbor at all (i.e., the piece(s) is next to an open location like the edge of the image). 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 described in Section 4.4.3.

This thesis subcategorizes non-adjacent best buddies depending on whether they are internal (i.e., the piece has an actual neighbor) or external (i.e., the piece has no neighbor). Interior non-adjacent best buddies are generally more concerning since they are used for segmentation and potentially as an estimation metric.

4.3.2 Best Buddy Density

As mentioned previously, each puzzle piece side may or may not have a best buddy relationship. For a puzzle consisting of n pieces each of which has q sides [1], Equation (11) defines the best buddy density, BBD given the image has b total best buddies. BBD is bounded between 0 and 1 (inclusive); a higher best buddy density indicates that the individual puzzle pieces can be more easily differentiated from one another.

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

Ideally, a puzzle would have only adjacent best buddies, and that all

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

neighboring puzzle piece sides were also adjacent best buddies. In such cases, the best buddy density would be less than one; the exact to which it would be below one is dependent on the puzzle dimensions and any missing pieces. BBD could be further refined to only consider adjacent best buddies in the term b.

Best buddy density may vary across an image with some areas showing lower density than others. Equation (11) can be adjusted to a more local metric by reducing the value of b and n based off the best buddy profile of a connected subset of pieces.

4.4 Visualizing Solver Output Quality and Best Buddies Relationships

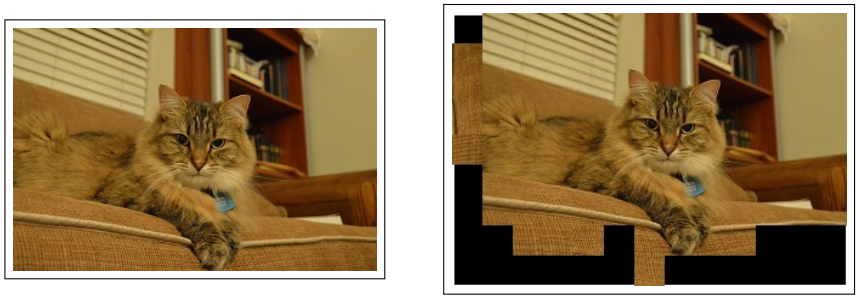
In images with thousands of pieces, it is often difficult to visually determine the location of individual pieces that are incorrectly placed. What is more, visual tools help developers quickly detect and fix latent bugs. The following two subsections describe the tools developed as part of this thesis for visualizing direct and neighbor accuracy as well as for visualizing best buddies.

4.4.1 Visualizing EDAS and SEDAS

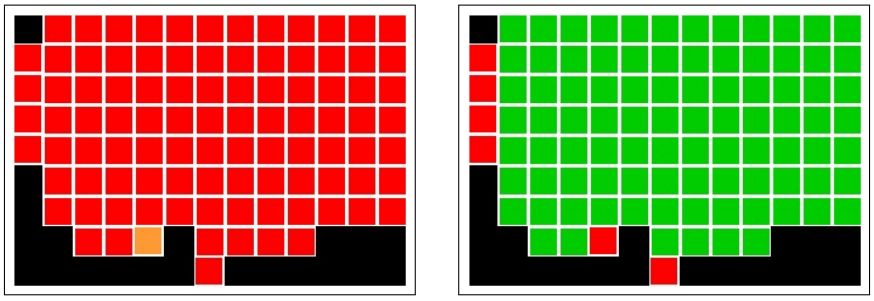
In standard direct accuracy, EDAS, and SEDAS, each puzzle piece is assigned a single value (i.e., correct or incorrect). Due to that, the direct accuracy visualization represents each puzzle by a square filled with a solid color. One additional refinement used in this thesis is to subdivide the ‘‘incorrect’’ placements into a set of subcategories; they are, in order of precedence: wrong puzzle, wrong location, and wrong rotation. 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 visualization would have the same dimensions as the ground-truth input and would consist of only green squares.

Wrong Puzzle	Wrong Location	Wrong Rotation	Correct Location	No Piece Present

Table 1: Color Scheme for Puzzles Pieces in Direct Accuracy Visualizations



(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

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, all but one 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 in SEDAS.

Wrong Puzzle	Wrong Neighbor	Correct Neighbor	No Piece Present

Table 2: Color Scheme for Puzzles Piece Sides in Neighbor Accuracy Visualizations

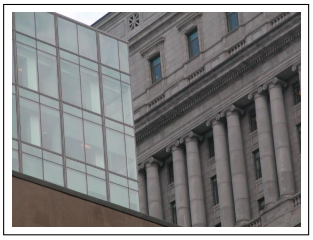
4.4.2 Visualizing ENAS

In a jig swap puzzle, a piece may have best buddies on up to four sides (since the pieces are square). As such, each piece in the ENAS visualization is divided into four isosceles triangles; the base of each triangle is along the side of the puzzle piece whose neighbor accuracy is being represented. A puzzle piece's 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 in the solver output and ground-truth image match. The "wrong puzzle" classfication applies only to Mixed-Bag puzzles and occurs when a piece in the solver output is does not from the puzzle of interest, P_i.

Figure 5 shows an actual output when solving a Mixed-Bag puzzle with two images. Note that that the puzzle of interest (P_i) is the glass and stone building while the other puzzle (P_k) is a rainforest house.

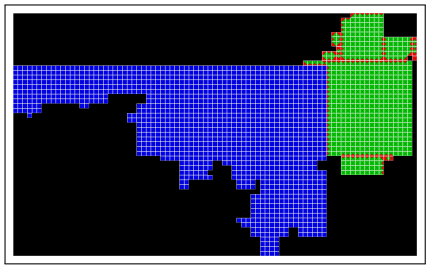
All pieces that came from the rainforest house image are shown as blue despite being assembled correctly; this is because they are not from the puzzle of interest. All neighbors from the puzzle of interest (i.e., the glass and stone building) that are placed next to their original neighbor are represented 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]



(b) Input Image # 2 - Building Exterior [23]

(c) Solver Output



(d) ENAS Visualization

Figure 5: Example Solver Output Visualization for ENAS

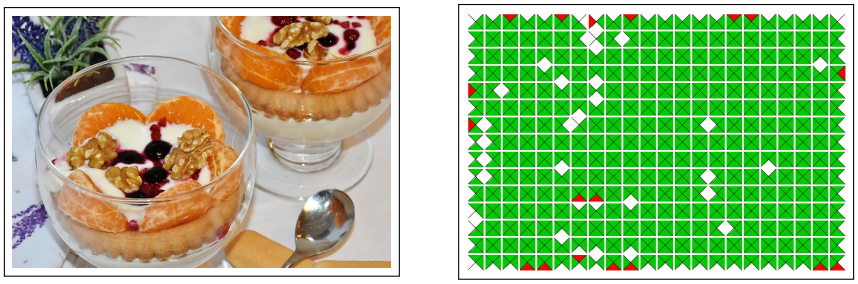
No Best Buddy	Non-Adjacent Best Buddy	Adjacent Best Buddy	No Piece Present

Table 3: Color Scheme for Puzzles Piece Sides in Best Buddy Visualizations

4.4.3 Visualizing Best Buddies

The visualization for best buddies is similar to that of neighbor accuracy. Hence, each puzzle piece in the best buddy visualization 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.



(a) Original Image [24] (b) Best Buddy Visualization

Figure 6: Visualization of Best Buddies in an Image

Note that the image has 4 interior and 14 exterior non-adjacent best buddies. This is despite having 16-times more interior interior sides. What is more, one can see that the best buddy density is high and generally uniform.

CHAPTER 5

Experimental Results

CHAPTER 6

Conclusions and Future Work

This thesis presented the first fully-automated solver for Mixed-Bag jigsaw puzzles. No other solver today has formalized an algorithm estimate the number of input images; what is more, the algorithm has been shown to be able to successfully solve twice as many puzzles as the current state, despite receiving less information. What is more, the solver does not relay on a specific assembly strategy, meaning the solver's performance will improve as better solvers are proposed.

Opportunities currently exist to further improve the Mixed-Bag Solver's performance. First, the ceiling on the quality of solved outputs is significantly affected by the assembler. This solver was designed to being largely independent of the assembler used, meaning the solver's performance will improve as better solvers are proposed. As such, an improved solver that further prioritizes assembly based off best buddies is currently under development. What is more, Paikin & Tal's algorithm often poorly assembles areas with low best buddy density, which this new assembler should also improve.

Another area where future investigation is planned is the threshold for hierarchical clustering, which is currently set at a fixed value. It is expected that a more dynamic approach may improve the clustering overall.

One final area where further work is planned is around the selection of seed pieces. As explained in Section 3.3.2, a stitching piece is by definition a member of a saved segment. In some cases, the mini-assembly may not actually expand the segment, which would in turn prevent segment clustering. An improved approach

may involve selecting stitching pieces that belong to no segment, with the expectation that such pieces may have superior stitching properties.

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).

APPENDIX A

Example Output of a Single Segmentation Round

APPENDIX B

Ten Puzzle Solver Result

Paikin & Tal [16] showed the results solving a maximum of five puzzles; what is more, they also provided the solver with the number of input puzzles. The Mixed-Bag Solver has shown that it can assemble up to 10 puzzles with results comparable to the Paikin & Tal solver on each image individually.

Figures B.7 and B.8 show the input images supplied to both the Mixed-Bag and Paikin & Tal solvers. The 5,850 pieces come from images that are five different sizes.

Figures B.9 and B.10 show the Mixed-Bag Solver outputs for these images. Four output images (e.g., (a), (b), (e), (i)) perfectly match their original images. The rest have only have a small percentage of pieces out of place. This is shown in the SEDAS visualizations in Figures B.11 and B.12.

(a) 264 Pieces [24]

(b) 330 Pieces [24]

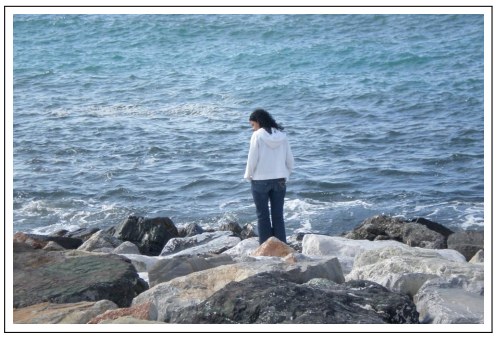
(c) 432 Pieces [13]

(d) 540 Pieces [23]

(e) 540 Pieces [23]

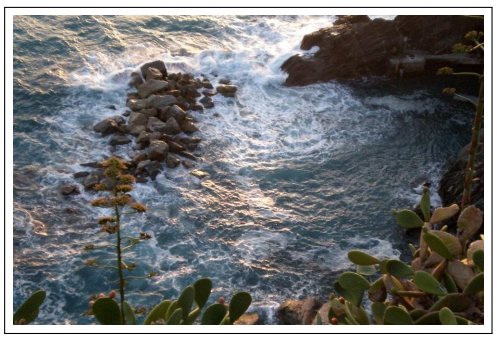
(f) 540 Pieces [23]

Figure B.7: First Set of Six Images Comprising the 10 Image Test Set

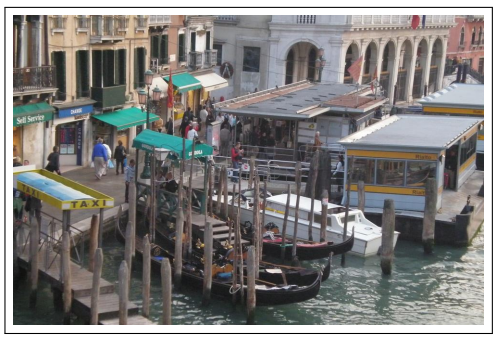


(g) 805 Pieces [23]

(h) 805 Pieces [23]



(i) 805 Pieces [23]



(j) 805 Pieces [23]

Figure B.8: Second Set of Four Images Comprising the 10 Image Test Set

Reconstructed Image (a)

Reconstructed Image (b)

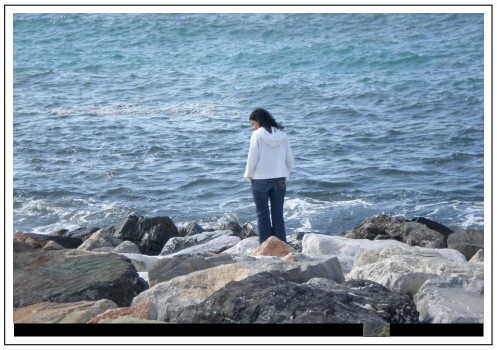
Reconstructed Image (c)

Reconstructed Image (d)

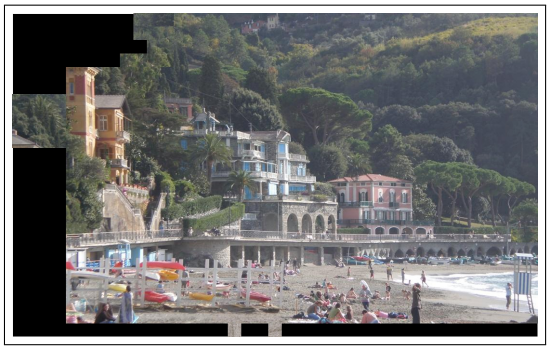
Reconstructed Image (e)

Reconstructed Image (f)

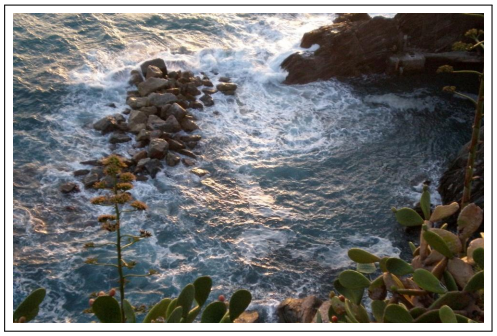
Figure B.9: First Set of Six Mixed-Bag Solver Images for the 10 Image Test Set



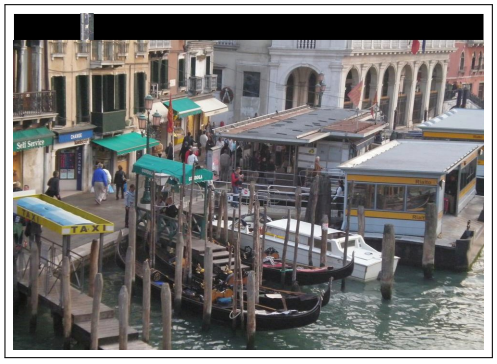
Reconstructed Image (g)



Reconstructed Image (h)

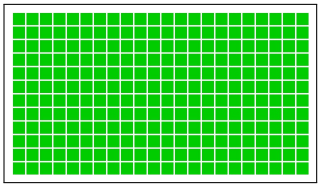


Reconstructed Image (i)

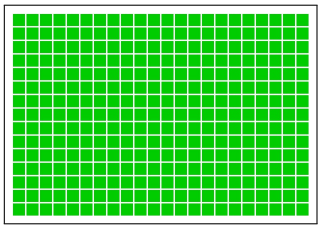


Reconstructed Image (j)

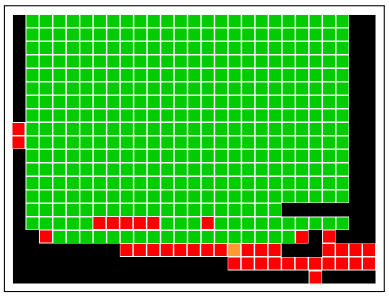
Figure B.10: Second Set of Four Mixed-Bag Solver Images for the 10 Image Test Set



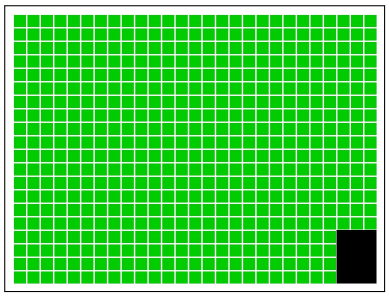
SEDAS Visualization of Image (a)



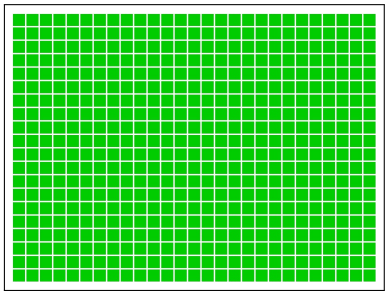
SEDAS Visualization of Image (b)



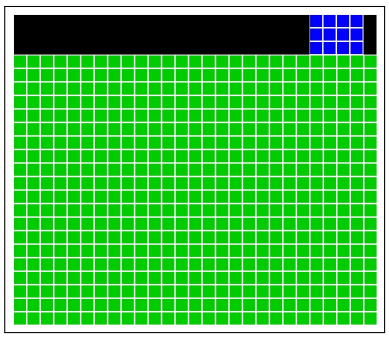
SEDAS Visualization of Image (c)



SEDAS Visualization of Image (d)

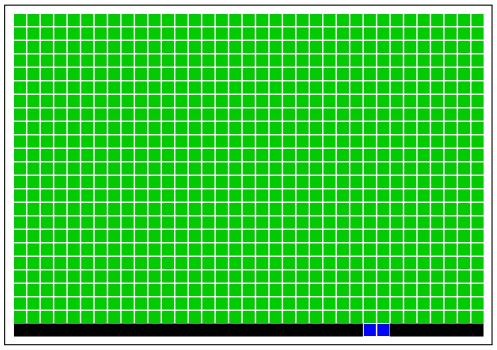


SEDAS Visualization of Image (e)

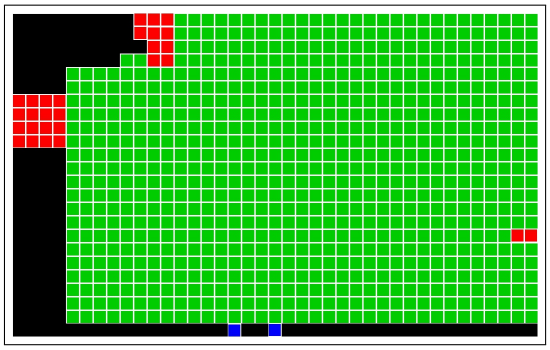


SEDAS Visualization of Image (f)

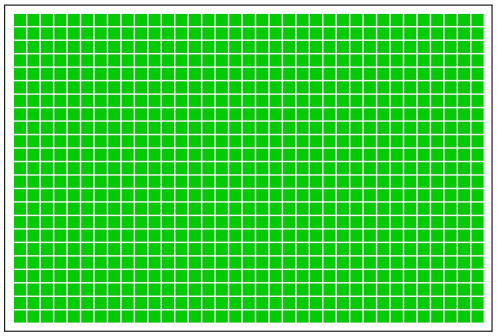
Figure B.11: First Set of Six SEDAS Visualizations for the 10 Image Test Set



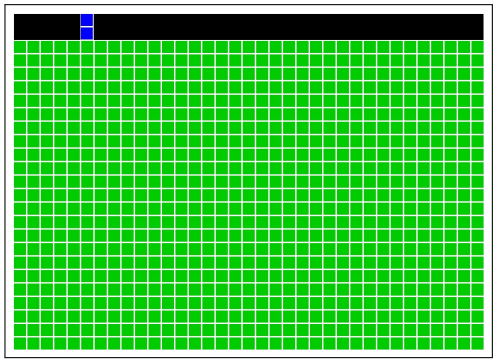
SEDAS Visualization of Image (g)



SEDAS Visualization of Image (h)



SEDAS Visualization of Image (i)



SEDAS Visualization of Image (j)

Figure B.12: Second Set of Four SEDAS Visualizations for the 10 Image Test Set