An Improved Square-Piece, Multiple Simultaneous Jigsaw Puzzle Solver

CS297 Final Report

Zayd Hammoudeh
(zayd.hammoudeh@sjsu.edu)

May 2, 2016

Table of Contents

List of Figures

List of Tables

1 Introduction

Jigsaw puzzles were first introduced in the 1760s when they were made from wood. Their name derives from tool (jigsaw) 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 centuries, it was not until 1964 that the first automated jigsaw puzzle solver was proposed by [3], and that solver could only solve 9 piece puzzles. While an automated jigsaw puzzle solver may seem trivial, it has been shown by [4] and [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 equally sized, non-overlapping squares. Jig swap puzzles are substantially more difficult to solve since piece shape can not be considered when determining affinity between pieces. Rather, only the image information on each individual piece is used when solving the puzzle.

Solving a jigsaw puzzle simplifies to reconstructing an object from a set of component pieces. As such, techniques developed for jigsaw puzzles can be generalized to many practical problems. Examples where jigsaw puzzle solving strategies have been used include: 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 ground-truth (i.e. original) input is unknown. This significantly increases the difficulty of the problem as the overall structure of the complete solution must be determined solely from a bag of individual pieces.

This thesis proposes an improved multiple simultaneously jig swap puzzle solver. What is more, it proposes a set of new metrics for measuring the quality of solvers for multiple simultaneous puzzles.

2 Previous Work

Computational solvers for jigsaw puzzles have been studied since the 1960s when Freeman and Garder proposed an approach that could solve jigsaw puzzles containing up to nine pieces that used only the piece shapes [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 were fixed in their correct location before the the solver began. Cho *et. al.*'s solver was also provided the puzzle's actual dimensions. Future solvers would improve on Cho *et. al.*'s results while simultaneously reducing the amount of information (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 (as opposed to standards such as RGB or CMYK). LAB was selected due to its property of normalizing the lightness and color variation across all three dimensions. Cho *et. al.* also proposed a measure for quantifying the pairwise distance between two puzzle pieces that became the basis of most of the future work (see Section 3).

Pomeranz *et. al.* [14] proposed an iterative, greedy jig swap puzzle solver in 2011. Their solver did not rely on anchor pieces, and the only information passed to the solver were the pieces and the size of the puzzle. Pomeranz *et. al.* also generalized and improved on Cho *et. al.* piece pairwise distance measure by proposing a "predictive distance measure". Finally, Pomeranz *et. al.* introduced the concept of "best buddies", which are any two pieces that are more similar to each other than they are to any other piece. Best buddies have served as both an estimation metric for the quality of solver result as well as the foundation of some solvers' placers [15].

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 while the size of the images in the second and third benchmarks is 2,360 and 3,300 pieces respectively.

In 2012, Gallagher [16] formally categorized jig swap puzzles into three primary types. The following is Gallagher's proposed terminology; his nomenclature is used throughout this thesis.

- **Type 1 Puzzle**: The dimension of the puzzle (i.e. the width and height

of the ground-truth image in number of pixels) is known. What is more, the orientation of each piece is 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 used by [13, 14].

- **Type 2 Puzzle**: This is an extension of a type 1, where pieces may be rotated in 90° increments (e.g. 0°, 90°, 180°, or 270°). This change alone increases the number of possible solutions by a factor of 4^n (where n is the number of pieces in the puzzles) in comparison to a Type 1 puzzle. What is more, no piece locations are known in advance; this change eliminates the use of anchor piece(s). Lastly, the dimensions of the ground-truth image/puzzle may be unknown.

- **Type 3 Puzzle**: All puzzle piece locations are known and only the rotation of the puzzle pieces is unknown. This is the least computationally complex of the three puzzle variants and is generally considered the least interesting. Type 3 puzzles are not explored as part of this thesis.

Sholomon *et. al.* [17] in 2013 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 (e.g. 5,015, 10,375, and 22,834 piece) benchmarks [18].

Paikin & Tal [15] published in 2015 a greedy solver that handles both type 1 and type 2 puzzles. What is more, their solver is able to handle puzzles with missing pieces; it can also solve multiple puzzles simultaneously. Paikin & Tal's solver is explored in significant depth in Section5.

3 Puzzle Piece Pairwise Affinity

Pairwise affinity quantifies the similarity between the sides of two puzzle pieces. $D(x_i, s_i, x_j, s_j)$ and $C(x_i, s_i, x_j, s_j)$ represent the distance and compatibility (i.e. similarity) respectively between side s_i of puzzle piece x_i and side s_j of puzzle piece x_j.

3.1 Cho *et. al.* Pairwise Affinity

As mentioned in section 2, Cho *et. al.* [13] proposed one of earliest edge-based pairwise affinity measures for two puzzle pieces. Equation (1) defines the distance between the left ("L") side of piece x_i and the right ("R") side of piece x_j using Cho *et. al.*'s approach. Note that K is the width/height of a puzzle piece in number of pixels[1].

$$D(x_i, L, x_j, R) = \sum_{k=1}^{K} \sum_{d=1}^{3} (x_i(k, K, d) - x_j(k, 1, d))^2 \tag{1}$$

Since the LAB colorspace has three dimensions (e.g. lightness and A/B opponent colors), d ranges between 1 and 3. Similarly, $x_i(k, K, d)$ represents the LAB value in dimension "d" for the pixel at "k", column "K" in puzzle piece x_i.

3.2 Pomeranz *et. al.* Pairwise Affinity

One of the disadvantages of Cho *et. al.*'s metric is that it simply squares the difference between the pieces' pixel values. It is possible that superior results may be achieved by allowing the solver to modify this exponent term. Pomeranz *et. al.* in [14] generalize Equation (1) using the $(L_p)^q$ norm as shown in Equation (2)[2].

$$D(x_i, L, x_j, R) = \left(\sum_{k=1}^{K} \sum_{d=1}^{3} (|x_i(k, K, d) - x_j(k, 1, d)|)^p \right)^{\frac{q}{p}} \tag{2}$$

[1] Cho *et. al.* used 7 for K.

[2] Pomeranz *et. al.* used 28 for K. This has generally become the standard in subsequent papers.

Hence, this measure is identical to that of Cho *et. al.* when p and q are equal to 2.

Another disadvantage of the metric proposed by Cho *et. al.* is that it only considers the border pixels. Hence, if there is a gradient in the ground-truth image, two pieces may appear artificially dissimilar. To address this issue, Pomeranz *et. al.* proposed predictive compatibility; equation (3) is the predictive compatibility between the left ("L") side of piece x_i and the right ("R") side of piece x_j.

$$C(x_i, L, x_j, R) = \sum_{k=1}^{K} \sum_{d=1}^{3} \Big[([2x_i(k, K, d) - x_i(k, K-1, d)] - x_j(k, 1, d))^p \\ -([2x_j(k, 1, d) - x_i(k, 2, d)] - x_i(k, K, d))^p \Big]^{\frac{q}{p}} \tag{3}$$

Since Equation (3) includes the difference between the column of pixels adjacent to each edge, predictive compatibility can adjust for local gradients in an image.

3.3 Paikin & Tal Pairwise Affinity

Paikin & Tal in [15] used Pomeranz *et. al.*'s predictive compatibility as the foundation of their asymmetric distance measure. Paikin & Tal's approach is shown in Equation (4).

$$D(x_i, L, x_j, R) = \sum_{k=1}^{K} \sum_{d=1}^{3} \| [2x_i(k, K, d) - x_i(k, K-1, d)] - x_j(k, 1, d) \| \tag{4}$$

Note that Paikin & Tal set p and q equal to 1 as it not only increased the accuracy of their solver, but is also significantly reduced the computational time. It is important to note that since this distance is asymmetric, in most cases $D(x_i, L, x_j, R)$ will not equal $D(x_j, R, x_i, L)$.

Pomeranz *et. al.* only considered the relationship between two individual pieces when determining the pairwise compatibility. In images (or areas of images) with little variation (e.g. an all white image), non-adjacent puzzle pieces may have artificially low pairwise distances. To account for this, Paikin & Tal proposed asymmetric compatibility as shown in Equation (5).

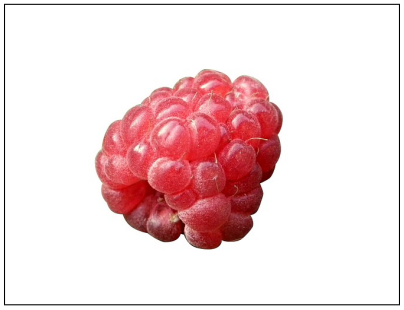


Figure 1: A Computer Manipulated Image with Poor Asymmetric Compatibility

$$C(x_i, L, x_j, R) = 1 - \frac{D(x_i, L, x_j, R)}{secondD(x_i, L)} \tag{5}$$

$secondD(x_i, L)$ is the second best asymmetric distance between the left side of piece x_i and all other pieces. By normalizing compatibility with respect to the second best match, pairings that have actually have high compatibility are more easily identifiable while at the same time penalizing speciously pairings high compatibility due solely to those pieces coming from areas of the image with low variation.

3.3.1 Improved Asymmetric Compatibility

In an image that was generated from an analog input (e.g. a photograph), it is expected that there will be some degree of natural variation across the image due to variations in brightness, the object in the photo, and the image sensor. However, in digital images that are generated or manipulated by computers, this variation can be trivially removed. An example of this would be an image of a solid color or object(s) in front of a white background as shown in figure 1.

The puzzles pieces along the border of the image as well as the pieces in the background have sides that are all white. This entails that regardless of which of the three metrics are used, the distance between them is zero. What is more, Paikin & Tal do not explicitly define how to handle this case for asymmetric compatibility. Therefore, to better handle computer generated images, this thesis proposes an enhanced definition of asymmetric compatibility as shown in Equation (6).

$$C(x_i, L, x_j, R) = \begin{cases} 1 - \frac{D(x_i, L, x_j, R)}{secondD(x_i, L)} & secondD(x_i, L) \neq 0 \\ -\alpha & secondD(x_i, L) = 0 \end{cases} \tag{6}$$

where α is a penalty term that denotes that such pairings have low compatibility.

3.4 Best Buddies

As defined by Pomeranz *et. al.*, two pieces, x_i and x_j are best buddies on their respective sides s_i and s_j if they are more compatible (i.e. similar) to each other than they are to each other. This is more formally described in Equation (7). This definition of best buddies applies regardless of whether the compatibility definition of Pomeranz *et. al.* or Paikin & Tal is used.

$$\begin{gathered} \forall x_k \forall s_k \in Parts, \quad C(x_i, s_i, x_j, s_j) \geq C(x_i, s_i, x_k, s_k) \\ \text{and} \\ \forall x_k \forall s_k \in Parts, \quad C(x_j, s_j, x_i, s_i) \geq C(x_j, s_j, x_k, s_k) \end{gathered} \tag{7}$$

$Parts$ represents the set of all pieces in the puzzle, and s_k is any of the four sides of piece x_k.

In most cases, it is relatively rare that two pieces are best buddies and are not actually neighbors [15]. When considering all sides of a piece, it is rarer still that a piece has more best buddies that are not its neighbor than that are its neighbor. This makes best buddies a critical tool in many placers.

3.4.1 Unique Best Buddies

As explained in Section 3.3.1, reconstructing computer generated or computer manipulated image may introduce new challenges not found with photographs; best buddies is another example of this.

In photographs, it is generally unlikely that there will be a significant number of cases where a piece has multiple best buddies on the same side. However, computer generated images may have areas of truly solid colors, and any pieces in those areas will all be best buddies with one another, making best buddies a far less discerning indicator of compatibility. This thesis addresses such issues by modifying the definition of best buddies as shown in Equation (8).

$$\forall x_k \forall s_k \in Parts, \quad C(x_i, s_i, x_j, s_j) > C(x_i, s_i, x_k, s_k)$$

$$\text{and} \tag{8}$$

$$\forall x_k \forall s_k \in Parts, \quad C(x_j, s_j, x_i, s_i) > C(x_j, s_j, x_k, s_k)$$

Rather than relying on best buddies being "greater than or equal" to all other pieces, the modified requirement is that pairings must be strictly "greater than." Hence, best buddy pairings are exclusive with cliques of size greater than two being explicitly disallowed.

3.4.2 Visualizing Best Buddies

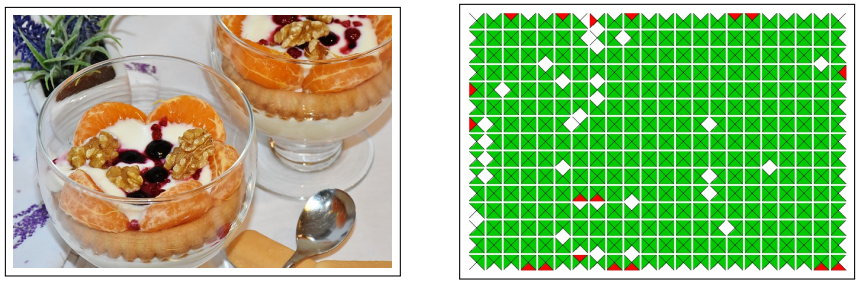
There is currently no standard for visualizing best buddy relationships in images. This thesis proposes a format for the first time. As a nomenclature, this thesis refers to any best buddies that are neighbors in the image as "adjacent best buddies" while any best buddies that are not neighbors are referred to as "non-adjacent best buddies".

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 best buddy visualization is divided into four isosceles triangles; the base of each triangle is along the side of the puzzle piece whose best buddy is being denoted as either adjacent or non-adjacent. The four isosceles triangles all share a common, non-base vertex in the center of the puzzle piece. A white border is placed around all puzzle pieces to make it easier to differentiate pieces from one another. A black square represents a missing or unpopulated piece location.

Figure 2 shows an image denoted as "(a)" and its accompanying best buddy visualization denoted as "(b)". Adjacent and non-adjacent best buddies are represented by green and red isosceles triangles. Any edge that has no best buddies is shown as a white triangle.

3.4.3 Interior and Exterior Best Buddies

In all previous research, all best buddies (in particular best buddy errors) were treated the same. However, pieces that are on the exterior (i.e. edge) of a puzzle or that are next to a missing piece are expected to naturally have a higher rate of non-adjacent best buddies than pieces on the interior of the puzzle. This is because such piece sides do not have an true neighbor leaving



(a) Original Image (b) Best Buddy Visualization

Figure 2: Visualization of Best Buddies in an Image

them more likely to couple with an unrelated piece. As an example, Figure 2 has four non-adjacent interior best buddies and 14 exterior non-adjacent best buddies despite there being despite there being 16 times more interior edges than exterior edges.

When using best buddy accuracy as an estimation metric, this thesis differentiates between interior and exterior best buddy errors by giving interior best buddy errors higher weight.

4 Quantifying the Quality of a Solver Output

Cho *et. al.* [13] defined the two metrics for quantifying the accuracy of a solver result namely: Direct Accuracy and Neighbor Accuracy. These metrics have been used in subsequent work [14, 15, 16, 17, 19]. This section describes the existing metrics, the weaknesses of these metrics, and proposes enhancements to these metrics to make them more meaningful for type 2 puzzles as well as when solving multiple puzzles simultaneously.

In the final subsection, tools developed to visualize the different solver quality metrics are discussed.

4.1 Direct Accuracy

Direct accuracy is the most naïve accuracy measure. It is defined as the ratio of the number of pieces placed in the same location as the ground-truth (i.e. source). The formal definition of direct accuracy (DA) is shown in Equation (9) where n is the number of pieces in the source puzzle and c is the number of pieces placed in their correct location.

$$DA = \frac{c}{n} \tag{9}$$

Direct accuracy is vulnerable to shifts in the solved image where even a few misplaced pieces can cause a significant decrease in the accuracy. This can be particularly true when the ground-truth image's puzzle dimensions are not known/fixed as described in Section 4.1.2.

This thesis proposes two modifications to the direct accuracy metric known as "Enhanced Direct Accuracy Score" and "Shiftable Enhanced Direct Accuracy Score". They each described in the following two subsections; afterwards, the thesis describes why both metrics serve complementary roles.

4.1.1 Enhanced Direct Accuracy Score

The standard direct accuracy metric does not account for the possibility that in the solver output there may be pieces from multiple puzzles. For a puzzle P_i in the set of input puzzles P (where $P_i \in P$) and a set of solved puzzles S where S_j is a solved puzzle in S, then Equation 10 defines the Enhanced Direct Accuracy Score (EDAS).

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

c_{ij} is the number of pieces from input puzzle P_i correctly placed (with no rotation in 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 solved S_j.

By dividing by the total number of pieces n_i in puzzle P_i, EDAS necessarily marks as incorrect any pieces from P_i that are not in solved puzzle S_j. In addition, EDAS also penalizes any pieces not from P_i that are in S_j through the term $m_{k,j}$. Hence, the combination of these two factors ensures that EDAS accounts both for extra and misplaced pieces.

When solving only a single puzzle, EDAS and the standard Direct Accuracy proposed by Cho *et. al.* are equivalent. In the case of solving multiple puzzles simultaneously, EDAS considers all pieces from the input puzzle of interest (P_i) while penalizing for pieces that are present from other input puzzles (P_k).

It is important 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 may be greater than the number of pieces in the solved puzzle S_j.

4.1.2 Shiftable Enhanced Direct Accuracy Score

As mentioned previously, direct accuracy is vulnerable to shifts in the original image. At times, the degree to which direct accuracy punishes shifts can be overly punitive.

Figure 3 shows an original image and an actual solver output where the puzzle boundaries where not fixed. Note that only a single piece is misplaced, causing all pieces to be shifted to the right one location. This single misplaced piece causes the direct accuracy score to drop to 0%. Had this same piece been misplaced at 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 indicates that it has room for improvement.

To address the often misleadingly punitive nature of the direct accuracy measure, this thesis proposes the Shiftable Enhanced Direct Accuracy Score (SEDAS). Let p_n be the puzzle piece that is closest to the upper left corner of

(a) Original Image (b) Solver Output

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

the solved image's boundary with a minimum Manhattan distance d_min. Also let L as the set of all possible puzzle piece locations that minimum distance d_min, then the SEDAS for a puzzle P_i is defined as:

$$SEDAS_{P_i} = \arg\max_{l \in L} \left(\arg\max_{S_j \in S} \frac{c_{i,j,l}}{n_i + \sum_{k \neq i}(m_{k,j})} \right) \tag{11}$$

Note that as reference point (l) for the solved is shifted, then only the number of pieces in the correct location ($c_{i,j,l}$) will also change.

Rather than defining SEDAS based off the closest puzzle to the upper left corner of the image, an alternative is to use the point anywhere in the image that maximizes Equation (11). However, that approach will usually be significantly more computationally complex in particular on large puzzles of several thousand pieces. Hence, this thesis' approach balances finding a meaningful direct accuracy score with computational efficiency.

4.1.3 Necessity of Using Both EDAS and SEDAS

While EDAS can be misleadingly punitive, it is not correct that EDAS should 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 of minimum distance locations "L" (as defined in Equation (11)). Hence, there is no additional computational time required to calculate EDAS. What is more, by continuing to use EDAS along with SEDAS, it is often possible to detect whether there was any shifting in the original image. 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 the ground-truth image as in the solved image over the total number of piece sides. More formally, let q be the number of sides of each puzzle piece (i.e. 4), and n be the number of pieces in the original. If a is the number of puzzle piece sides adjacent in the ground-truth and solved images, then the neighbor accuracy (NA) is defined as shown in Equation (12).

$$NA = \frac{a}{n\ q} \tag{12}$$

Unlike direct accuracy, neighbor accuracy is largely unaffected by shifts in the solved image since it considers a piece's neighbors and not its actual location. However, it does not define how to handle the presence of pieces from multiple puzzles in the solved result. Therefore, it must be amended to consider such cases.

4.2.1 Enhanced Neighbor Accuracy Score

Similar to EDAS for Direct Accuracy, Enhanced Neighbor Accuracy Score (ENAS) addresses neighbor accuracy's limitations when quantifying the quality of a solver output when solving multiple puzzles simultaneously.

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 in puzzle P_i that are adjacent in solved puzzle 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 Enhanced Neighbor Accuracy Score for P_i is defined in Equation (13).

$$ENAS_{P_i} = \arg\max_{S_j \in S} \frac{a_{i,j}}{q\ (n_i + \sum_{k \neq i}(m_{k,j})} \tag{13}$$

Similar to EDAS as described 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 not in S_j. What is more, by including in the summation in the denominator of (13), ENAS marks as necessarily incorrect any pieces not from

P_i that are in S_j. The combination of these two factors allow ENAS to account for extra and misplaced pieces.

4.3 Visualizing Solver Output Quality

In images with thousands of pieces, it is often difficult to visually determine the location of pieces that are placed incorrectly. What is more, visual tools help developers quickly detect and fix latent bugs that may be in their program.

The following two subsections show the tools developed as part of this thesis for visualizing direct accuracy and neighbor accuracy.

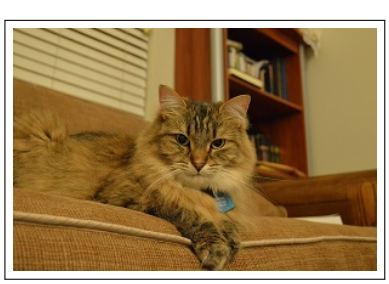
4.3.1 Visualizing EDAS and SEDAS

Regardless of whether standard direct accuracy, EDAS, or SEDAS is used, each puzzle piece is assigned a single value (e.g. correct or incorrect). Due to that, the direct accuracy visualization represents each puzzle by a solid color square. One additional refinement used is to subdivide the "incorrect" placements into a set of subcategories namely (in order of severity): wrong puzzle, wrong location, and wrong rotation. Table 1 shows the colors assigned to puzzle pieces depending on their direct accuracy classification.

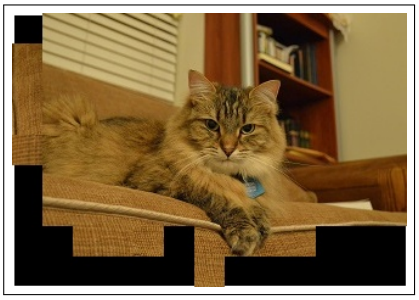
Wrong Puzzle	Wrong Location	Wrong Rotation	Correct Location	No Piece Present

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

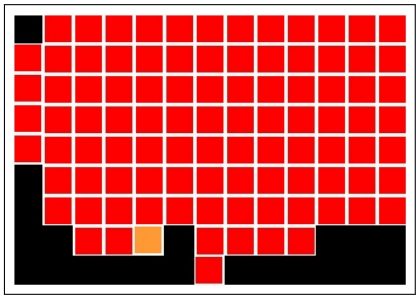
Figure 4 shows a solver output along with the EDAS and SEDAS visualizations. Since four puzzle pieces were erroneously placed on the left of the image, almost all pieces had the wrong location according to EDAS; the only exception is a single piece that had the right location, but wrong rotation. In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



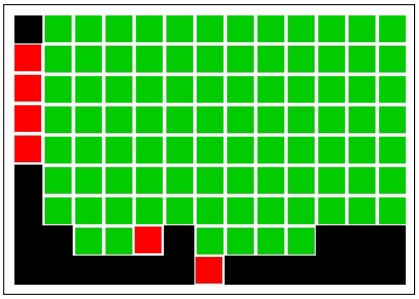
(a) Ground-Truth Image



(b) Type 2 Solver Output



(c) EDAS Visualization



(d) SEDAS Visualization

Figure 4: Example Solver Output Visualizations for EDAS and SEDAS

4.3.2 Visualizing Neighbor Accuracy

The visualization for neighbor accuracy is very similar to the techniques described in Section 3.4.2 for visualizing best buddies. Each puzzle piece is divided into four equal-sized isosceles triangles (i.e. one for each side). The triangles are assigned colors depending on whether their neighbor in the solver output matches their neighbor in the ground-truth image. The visualization includes a subcategory known as "wrong puzzle" which is a special case that occurs when solving multiple puzzles simultaneously and some of the pieces in the solved puzzle are not from the puzzle of interest (i.e. P_i). Table 2 defines the colors used to represent the different classification of puzzle piece sides for neighbor accuracy.

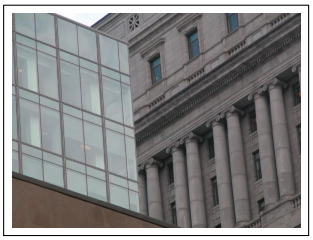
Wrong Puzzle	Wrong Neighbor	Correct Neighbor	No Piece Present

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

Figure 5 shows the output when solving two images simultaneously. Note that one of the input puzzles is a rainforest house while the other is the exterior of a glass and stone building. In this case, the puzzle of interest was the image of the glass and stone building.

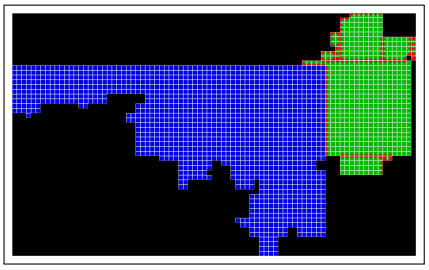
All pieces that came from the rainforest house image are shown as blue despite being assembled correctly; this is because the image of interest is the glass and stone building. All neighbors that are placed next to their original neighbor are represented by green triangles while all incorrect neighbors (e.g. the rainforest image) are shown with red triangles.

(a) Input Image # 1 - Rainforest House [20]



(b) Input Image # 2 - Building Exterior [21]

(c) Solver Output



(d) ENAS Visualization

Figure 5: Example Solver Output Visualization for ENAS

5 Paikin & Tal Solver

This section reviews the solver proposed by Paikin & Tal [15]; their Java implementation was never open sourced. As such, this section also describes a complete implementation of their approach developed as part of this thesis. The section concludes with a summary of some of the weaknesses of Paikin & Tal's approach.

5.1 Overview of Paikin & Tal's Algorithm

Paikin & Tal's solver was inspired by the solver developed by Pomeranz *et. al.* in [14]. However, unlike the previous work, Paikin & Tal's is deterministic; their greedy algorithm iteratively places the puzzle piece that at each stage has the maximum confidence score. Paikin & Tal's approach is able to handle puzzles with missing pieces where the puzzle size and piece orientation are unknown. The only input provided to the algorithm is the expected number of output puzzles.

Paikin & Tal's algorithm has three distinct phases namely: inter-puzzle piece distance calculation, selecting of the seed piece, and placement. These are described in the following subsections. The modification required to the algorithm to solve multiple puzzles simultaneously is describing in an additional subsection.

5.1.1 Inter-Puzzle Piece Distance Calculation

The first stage of Paikin & Tal's algorithm is to calculate the inter-piece distance between all pairs of pieces. This is done using the asymmetric distance measure described in Section (4). This information is stored in an n by n matrix (where n is the number of puzzle pieces); after being calculated, the asymmetric distance never needs to be recalculated again throughout the duration of the algorithm.

As explained in Section 4, Paikin & Tal normalizes all of the asymmetric distances between each possible pairing of pieces by the second best distance for each piece. This has the effect of amplifying truly unique pairings versus pairings that arise from low variation areas of the image. They refer to this as "asymmetric compatibility". The asymmetric compatibility is then used to find the best buddies (if any) for all pieces.

The last step is to calculate the mutual compatibility ($\tilde{C}$) between pairs of

pieces (e.g. x_i and x_j), which is defined in Equation (14). $C(x_i, s_i, x_j, s_j)$ is the asymmetric compatibility between side s_i of piece x_j and side s_j of piece x_j; $C(x_j, s_j, x_i, s_i)$ is defined similarly. It is important to note that mutual compatibility is symmetric.

$$\tilde{C}(x_i, s_i, x_j, s_j) = \tilde{C}(x_j, s_j, x_i, s_i) = \frac{C(x_i, s_i, x_j, s_j) + C(x_j, s_j, x_i, s_i)}{2} \quad (14)$$

5.1.2 Selecting the Seed Piece

Similar to Pomeranz *et. al.*, Paikin & Tal is a kernel growing algorithm. Hence, a seed piece is selected, and all pieces are placed around that initial seed. Since the algorithm is greedy, the selection of a poor seed can have a significant effect on the final solution. Due to this, Paikin & Tal select a piece that is itself "distinctive" and comes from a "distinctive region."

Paikin & Tal define a seed piece as "distinctive" if it has best buddies on each of its sides. To ensure that a piece comes from a distinctive region, all of the seed piece's best buddies must also have four best buddies. In a puzzle, there may be multiple pieces that satisfy the "distinctive" piece in a "distinctive region" criteria; ties are broken by selecting the piece that has the maximum sum of mutual compatibilities among its neighbors.

5.1.3 Placement

Paikin & Tal utilize an iterative, greedy placer that places pieces around an expanding seed. Pseudocode for their algorithm is shown in Algorithm 1. Placement continues all pieces have been placed.

Algorithm 1 Paikin & Tal Placer

```
1: while |UnplacedPieces| > 0 do
2:     if |BestBuddyPool| > 0 then
3:         Get best candidate from the BestBuddyPool
4:     else
5:         Recalculate the asymmetric and mutual compatibility
6:         Select piece with the highest asymmetric compatibility
7:     Place the best piece
8:     Add the best piece's unplaced best buddies to the BestBuddyPool
```

The selection of the next (i.e. best) piece to place emphasizes placing pieces about which there is the highest confidence. First, if two pieces are best buddies,

then there is a very significant chance that they are best buddies. Second, even among a set of best buddies, some best buddies have stronger pairwise affinity than others as determined by the asymmetric compatibility.

Whenever a piece (including the seed) is placed, all of that piece's unplaced best buddies are added to the *BestBuddyPool*. As long as the best buddy pool is not empty, candidates for placement can only come from that pool; the best candidate from the *BestBuddyPool* is the one with the maximum asymmetric compatibility with an open slot in the puzzle.

If the *BestBuddyPool* is ever empty, the algorithm recalculates the asymmetric and mutual compatibilities between the unplaced pieces and any piece's open slot. The piece that has maximum asymmetric compatibility with an open slot is then selected as the next piece for placement.

5.1.4 Solving Multiple Puzzles

As mentioned in Section 5, the only input to the Paikin & Tal algorithm is that number of puzzles being solved. When solving more than one puzzle at a time, only a minor change to the placer described in Algorithm 1 is required.

If at any time the mutual compatibility between the next piece to place and the open slot drops below a predefined threshold (e.g. Paikin & Tal use 0.5) and the current number of boards is less than the specified number of boards, then the algorithm spawns a new puzzle. This entails clearing the *BestBuddyPool* and selecting a seed piece using the approach previously described in Section 5.1.2. Placement then continues simultaneously across all puzzles.

5.2 A Python Implementation of Paikin & Tal's Algorithm

Since no open-source implementation of the Paikin & Tal solver exists, these thesis developed one. It is written entirely using Python 2.7 [22]. The following subsections describe the two major Python packages that were developed.

5.2.1 The `hammoudeh_puzzle` Package

This package consists of two primary classes: `Puzzle` and `PuzzlePiece`. It is a generic infrastructure that is independent of the solver used.

The `Puzzle` class encapsulates all attributes of a puzzle including: identification number, size, type (e.g. type 1, type 2 - this is stored as an object of type "`PuzzleType`"), piece width, and all associated puzzle pieces. A puzzle can be created either from an image file or from a set of puzzle pieces. When parsing image files and exporting a solved puzzle, the `Puzzle` class uses the OpenCV Python package [23].

Individual puzzle pieces are represented using the `PuzzlePiece` class. Objects of this type have as attributes: identification number, location, width (in number of pixels), rotation (represented using class `PuzzlePieceRotation`), and image information (stored in a NumPy array [24]).

Additional features included in the `hammoudeh_puzzle` package include calculating EDAS, SEDAS, and ENAS as well as performing best buddy analysis on an image.

5.2.2 The `paikin_tal_solver` Package

The `paikin_tal_solver` package implements the Paikin & Tal algorithm described in Section 5. The primary interface for the user is through the `PaikinTalSolver` class; objects of this type are created using a constructor that takes the following parameters: the expected number of puzzles, a set of `PuzzlePiece` objects, the asymmetric distance function, the puzzle type (class "`PuzzleType`"), and optionally a set of fixed puzzle dimensions. The `PaikinTalSolver` class is composed of a set of component classes, which are described in the subsequent paragraphs.

The `InterPieceDistance` class calculates the asymmetric distance, asymmetric compatibility, and mutual compatibility between all pieces. What is more, since the `PaikinTalSolver` takes a distance function in its constructor, the user is able to experiment with how different distance metrics affect the solver's performance. While calculating the asymmetric compatibilities, the `InterPieceDistance` class also finds each piece's best buddies; this made it the natural choice to identify the starting piece(s).

Paikin & Tal do not identify the data structure used to implement the *BestBuddyPool* (see Algorithm 1); instead, the choice of the term "pool" may indicate that it is unstructured. However, the choice of data structure is critical as algorithm must be able to quickly removal of best (i.e. maximum) values from the pool and quickly insert new values into the pool.

This thesis' implementation of the *BestBuddyPool* relies on a combination three data structures. They are:

- **Dictionary of the Best Buddies Currently in this Pool** - This allows new best buddies to be quickly added to the list when a piece is placed and to delete the placed piece from the pool.
- **Dictionary of the Open Slots in the Puzzle** - Similar to the best buddy dictionary, this allows open slots in the puzzle to be quickly added and removed.
- **Best Candidate Max Heap** - A max heap was used as it allows the best candidate to be found quickly ($O(n)$). Each object in the pool is of type `BestBuddyHeapInfo` and represents the level of similarity between a best buddy in the pool and an open slot. Every time a new best buddy is added, pairings between it and all open slots are added to the heap. Similarly, whenever a new open slot in the puzzle is opened, pairings between that new slot and all best buddies currently in the pool are added to the heap.

The best candidate heap is not cleaned after each placement. Rather, as items are popped off the heap, they are check for validity (due to either the best buddy being already placed or the open slot in the puzzle being filled). If the heap size does get too large (currently set to one million elements), the algorithm will periodically clean the entire heap.

5.3 Limitations of Paikin & Tal's Algorithm

This section details some of the limitations with Paikin & Tal's algorithm.

5.3.1 Taking the Number of Puzzles as an Input

The only input to the solver other than the number of pieces is the expected number of puzzles (see Section 5). Pomeranz *et. al.* and Sholomon *et. al.* have both used best buddy accuracy as an estimation metric. It is expected that a solver could be developed that takes no input from the user other than the set of input pieces; the number of puzzles could then be inferred from the best buddy accuracy.

5.3.2 Using Only a Single Side for Placement

When performing placement, Paikin & Tal's algorithm only considers a single side of the puzzle piece. While this may only lead to a handful of poor placements, it is known that a single suboptimal decision in a greedy algorithm

can have a significant deleterious effect on the final result. An improved placer could prioritize based off the number of best buddies a piece has with respect to an open slot.

5.3.3 Determining the Seed Piece for Multiple Puzzles

When each new puzzle is spawned, Paikin & Tal's algorithm use the same approach for choosing seed piece. This can lead to issues if two puzzles are spawned using seed pieces from the same input puzzle. The most naive way to address this issue is to select multiple seed pieces when a puzzle is spawn and perform placement using all of the seed pieces in parallel. There may be additional techniques that can be used to identify the optimal seed piece.

5.3.4 Determining when to Spawn a New Puzzle

Paikin & Tal spawn a new puzzle when the mutual compatibility between the best candidate and the open slot falls below a preset threshold. This can cause the algorithm to prematurely spawn boards which further complicates the process of selecting the next seed piece since there are more pieces from which to choose. One possible approach that may be useful in selecting when to spawn to consider when the best buddy accuracy falls below a certain threshold. When that occurs, the algorithm can then use the trend of the best buddy accuracy to determine where in the placement the algorithm should backtrack and spawn the new board.

6 Conclusions

Significant progress was made over the course of this semester. Major accomplishment include: a thorough review of existing solvers, building a jigsaw puzzle solver using state-of-the art techniques, defining new solution quality metrics, and building visualization tools. What is more, key potential improvements to existing techniques have been identified and will serve as the initial set of tasks for next semester.

Bibliography

[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–Ú127, 1964.

[4] T. Altman, "Solving the jigsaw puzzle problem in linear time," *Appl. Artif. Intell.*, vol. 3, 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 (Proc. SIGGRAPH)*, vol. 27, 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," *Digit. Investig.*, 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.," *IEEE Conference on Computer Vision and Pattern Recognition*, 2008.

[10] L. Zhu, Z. Zhou, and D. Hu, "Globally consistent reconstruction of ripped-up documents.," *IEEE Trans. Pattern Anal. Mach. Intell.*, 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 Annual Conference on International Conference on Computer Engineering and Applications*, pp. 171–176, World Scientific and Engineering Academy and Society (WSEAS), 2007.

[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)*, CVPR '10, pp. 183–190, IEEE Computer Society, 2010.

[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)*, CVPR '11, pp. 9–16, IEEE Computer Society, 2011.

[15] 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)*, CVPR '15, IEEE Computer Society, 2015.

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

[17] D. Sholomon, O. David, and N. S. Netanyahu, "A genetic algorithm-based solver for very large jigsaw puzzles," in *2013 IEEE Conference on Computer Vision and Pattern Recognition, Portland, OR, USA, June 23-28, 2013*, pp. 1767–1774, 2013.

[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] K. Son, J. Hays, and D. B. Cooper, "Solving square jigsaw puzzles with loop constraints," in *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part VI*, pp. 32–46, 2014.

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

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

[22] P. S. Foundation, "Python language reference, version 2.7." `https://www.python.org/`.

[23] G. Bradski, "The opencv library," *Dr. Dobb's Journal of Software Tools*, Nov. 2000.

[24] S. v. d. Walt, S. C. Colbert, and G. Varoquaux, "The numpy array: A structure for efficient numerical computation," *Computing in Science and Engg.*, vol. 13, pp. 22–30, Mar. 2011.