# An Improved Square-Piece, Multiple Simultaneous Jigsaw Puzzle Solver

CS297 Final Report

Zayd Hammoudeh
(zayd.hammoudeh@sjsu.edu)

May 3, 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 cannot 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 simultaneous, jig swap puzzle solver. What is more, it defines a set of new metrics for measuring the quality of solvers for multiple simultaneous puzzles. Lastly, this thesis proposes enhancements to existing techniques to make them more robust for computer generated images.

## 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. $D(x_i, s_a, x_j, s_b)$ and $C(x_i, s_a, x_j, s_b)$ represent the distance and compatibility (i.e. similarity) respectively between side $s_a$ of puzzle piece $x_i$ and side $s_b$ 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 $x_j$.

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

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).

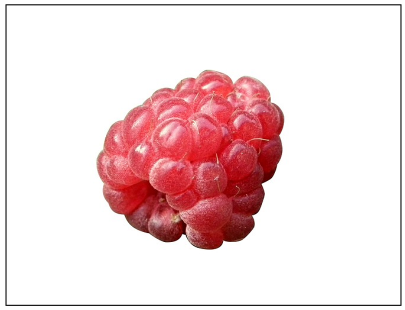


Figure 1: A Computer Manipulated Image with Misleadingly High Asymmetric Compatibility due to a White Background

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

$secondD(x_i, L)$ 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 simultaneously penalizing speciously high compatibility pairings from areas of the image with low variation.

### 3.3.1 Improved Asymmetric Compatibility

It is expected that images generated from an analog input (e.g. a photograph), will have some degree of natural variation in: brightness, the object in the photo, and the image sensor. In contrast, this variation can be trivially removed in images generated or manipulated by computers; this includes images that a single, solid color or object(s) shown in front of a solid background.

Figure 1 shows an object in front of a white background; the puzzles pieces along the image border and in the background have all white borders. Hence, regardless of which of the three previously described metrics is used, the distance between all of these pieces' side is zero. What is more, Paikin & Tal do not explicitly define how to handle asymmetric compatibility for such cases. Therefore, to better handle computer generated images, this thesis proposes an enhanced definition of asymmetric compatibility shown in Equation (6).

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

Note that $\alpha$ is a tunable, penalty term representing low compatibility.

## 3.4 Best Buddies

Pomeranz *et. al.* defined that two pieces, $x_i$ and $x_j$ are best buddies on their respective sides $s_a$ and $s_b$ if and only if they are more compatible (i.e. similar) to each other than they are to any other piece. This is shown more formally in Equation (7). This definition of best buddies applies regardless of whether the compatibility function of Pomeranz *et. al.* or Paikin & Tal is used.

$$\begin{gathered}\forall x_k \forall s_k \in Pieces, \quad C(x_i, s_a, x_j, s_b) \geq C(x_i, s_a, x_k, s_k) \\ \text{and} \\ \forall x_k \forall s_k \in Pieces, \quad C(x_j, s_b, x_i, s_a) \geq C(x_j, s_b, x_k, s_k)\end{gathered} \tag{7}$$

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

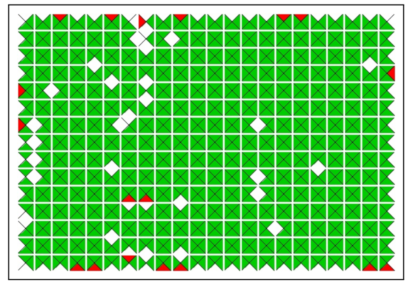
It is relatively rare that two pieces are best buddies and are not actually neighbors [15]. When considering all sides of a piece, it is rarer still that a piece has more best buddies that are not its neighbor than that are its neighbor. These attributes make best buddies a critical tool for many solvers.

### 3.4.1 Unique Best Buddies

It is relatively unlikely that there will be a meaningful number pieces that have multiple best buddies on the same side if the input to the solver is a photograph. However, similar to phenomenon described in the Section 3.3.1, multiple best buddies may occur in computer generated or computer manipulated images. In such cases, best buddies become a far less discerning tool for determining piece adjacency. This thesis addresses this by modifying the definition of best buddies as shown in Equation (8).

$$\begin{gathered}\forall x_k \forall s_k \in Parts, \quad C(x_i, s_a, x_j, s_b) > C(x_i, s_a, x_k, s_k) \\ \text{and} \\ \forall x_k \forall s_k \in Parts, \quad C(x_j, s_b, x_i, s_a) > C(x_j, s_b, x_k, s_k)\end{gathered} \tag{8}$$

(a) Original Image



(b) Best Buddy Visualization

Figure 2: Visualization of Best Buddies in an Image

Rather than relying on best buddies being "greater than or equal" to all other pieces as in Equation (7), 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 an image's best buddies. This thesis proposes such a standard for the first time. As a nomenclature, this thesis refers to any best buddies that are neighbors in the ground-truth 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.

Figure 2 shows an image denoted as "(a)" and its accompanying best buddy visualization denoted as "(b)". The color scheme for the different best buddy relationships is shown in Table 1.

### 3.4.3 Interior and Exterior Best Buddies

In 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

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

Table 1: Color Scheme for Puzzles Piece Sides in Best Buddy Visualizations

or that are next to a missing piece are expected to naturally have a higher rate of non-adjacent best buddies due to the fact that such piece sides do not have a true neighbor; this leaves them them more likely to couple with an unrelated piece. As an example, the image in Figure 2 has 4 non-adjacent interior best buddies and 14 exterior non-adjacent best buddies despite there being 16-times more interior sides.

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 a higher weight.

# 4 Quantifying the Quality of a Solver Output

Cho *et. al.* [13] defined two primary metrics for quantifying the accuracy of a solver result namely: direct accuracy and neighbor accuracy. These metrics have been used in subsequent work including [14, 15, 16, 17, 19]. This section describes the existing metrics, their weaknesses, 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 solver output quality are discussed.

## 4.1 Direct Accuracy

Direct accuracy is the most naïve accuracy measure. It is defined as the fraction of pieces placed in the same location in the ground-truth (i.e. original) and solved image with respect to the total number of pieces. Equation (9) shows the formal definition of direct accuracy ($DA$), where $n$ is the total number of pieces and $c$ is the number of pieces placed in the original (i.e. 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 accuracy. This can be particularly true when the ground-truth image's puzzle dimensions are not known by the solver as described in Section 4.1.2.

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 puzzles in the solver output. For a given 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 in $S$, Enhanced Direct Accuracy Score (EDAS) is defined as shown in Equation 10.

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

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

When solving only a single puzzle, EDAS and standard direct accuracy proposed by Cho *et. al.* are equivalent.

When solving multiple puzzles simultaneously, EDAS necessarily marks as incorrect any pieces from $P_i$ that are not in $S_j$ by dividing by $n_i$. What is more, EDAS penalizes for any pieces not from $P_i$ that are in $S_j$ through the term $m_{k,j}$. Hence, two factors enable EDAS to account for both extra and misplaced pieces.

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 $P_i$ and $S_j$.

#### 4.1.2 Shiftable Enhanced Direct Accuracy Score

As mentioned previously, direct accuracy is vulnerable to shifts in the original image. In many cases, direct accuracy is overly punitive when penalizing for shifts.

Figure 3 shows an original image and an actual solver output wen the puzzle boundaries were not fixed. Note that only a single piece is misplaced; this caused 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 shows that direct accuracy can be seriously flawed. This thesis proposes Shiftable Enhanced Direct Accuracy Score (SEDAS) to address the often misleadingly punitive nature of direct accuracy.

Let $d_{min}$ be the Manhattan distance between the upper left corner of the solved image and the nearest puzzle piece. Also let $L$ as the set of all possible puzzle piece locations within distance$d_min$ of the upper left corner of the

(a) Original Image

(b) Solver Output

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

image. Given $l$ a location in $L$ that is used as the reference point for determining the absolute location of each piece, then SEDAS is defined as shown in Equation (11)in Cho *et. al.*'s metric, this would be the upper left corner of the image.

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

In standard definition of direct accuracy, $l$ would be the upper left corner of the image; SEDAS shifts this reference point around 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 (11). However, that approach can 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 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, 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' side in both the ground-truth and solved image versus the total number of puzzle piece sides. More formally, let $q$ be the number of sides of each puzzle piece (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 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 only a piece's neighbors and not its absolute 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

Enhanced Neighbor Accuracy Score (ENAS) improves the neighbor accuracy metric's limitations by providing a framework to quantify 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 adjacent puzzle piece sides in puzzle $P_i$ that are also 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 ENAS for $P_i$ is defined as shown in Equation (13).

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

Similar to the technique described for EDAS in Section 4.1.1, ENAS divides by the number of pieces $n_i$ in input puzzle $P_i$. By doing so, it effectively marks as incorrect any pieces from $P_i$ that are not in $S_j$. What is more, by including in summation of all $m_{k,j}$ 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 allows 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 incorrectly placed. What is more, visual tools help developers quickly detect and fix latent bugs.

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 (i.e. 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 2 shows the colors assigned to puzzle pieces depending on their direct accuracy classification.

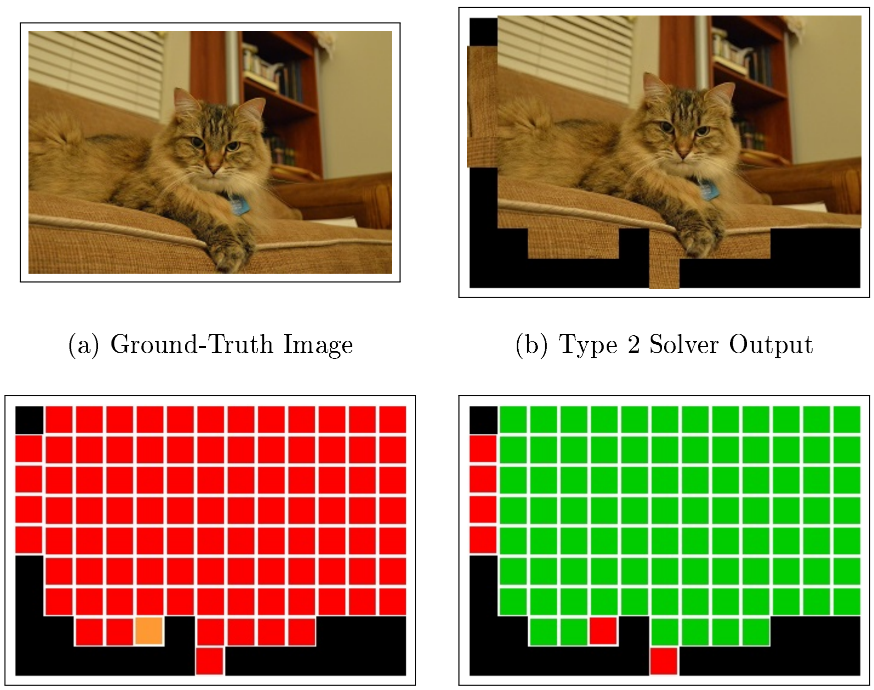
| Wrong Puzzle | Wrong Location | Wrong Rotation | Correct Location | No Piece Present |
|---|---|---|---|---|
| | | | | |

Table 2: Color Scheme for Puzzles Pieces in Direct Accuracy Visualizations

Figure 4 shows a type 2 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.

#### 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



(a) Ground-Truth Image

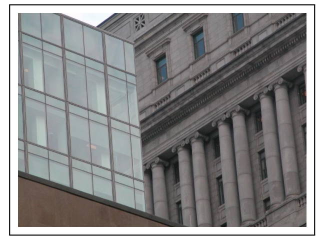
(b) Type 2 Solver Output

(c) EDAS Visualization

(d) SEDAS Visualization

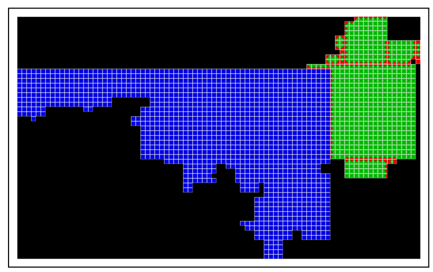
Figure 4: Example Solver Output Visualizations for EDAS and SEDAS

(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

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 3 defines the colors used to represent the different classification of puzzle piece sides in neighbor accuracy visualization.

| Wrong Puzzle | Wrong Neighbor | Correct Neighbor | No Piece Present |
|---|---|---|---|
| | | | |

Table 3: Color Scheme for Puzzles Piece Sides in Neighbor Accuracy Visualizations

Figure 5 shows an actual output when solving two images simultaneously. Note that that the puzzle of interest is the glass and stone building while the other puzzle 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 image of interest. All neighbors from the of interest 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 shown with red triangles.

# 5 Paikin & Tal Solver

This section reviews the solver proposed by Paikin & Tal [15]; their Java implementation is not open-source. As such, this section also describes a complete implementation of their approach that was developed as part of this thesis. The section concludes with a summary of some of the weaknesses of Paikin & Tal's algorithm.

## 5.1 Overview of Paikin & Tal's Algorithm

Paikin & Tal's solver was inspired by the solver developed by Pomeranz *et. al.* in [14]. Paikin & Tal's uses a deterministic, greedy algorithm that iteratively places the puzzle piece that has the maximum confidence score at each stage. Paikin & Tal's approach is able to handle puzzles of unknown size with missing pieces and where piece orientation is not known. The only required input to the algorithm is the expected number puzzles.

Paikin & Tal's algorithm has three distinct phases namely: inter-puzzle piece distance calculation, selection of the seed piece, and placement. These stages are described in the following subsections. The modification required during placement to solve multiple puzzles simultaneously is described 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 pieces. This is done using the asymmetric distance measure described in Section 3.3. The distance information is stored in an $n$ by $n$ matrix (where $n$ is the number of puzzle pieces); after being calculated, the asymmetric distance is never recalculated throughout the duration of the algorithm.

As explained in Section 3.3, Paikin & Tal normalize the asymmetric distances using the asymmetric compatibility function in Equation (5). This has the effect of amplifying truly unique pairings while penalizing speciously similar pairings that arise from low variation areas of the image. The asymmetric compatibility is then used to find the best buddies for each piece (if any).

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). $C(x_i, s_a, x_j, s_b)$ is the asymmetric compatibility between side $s_a$ of piece $x_i$ and side $s_b$ of piece $x_j$; $C(x_j, s_b, x_i, s_a)$ is defined similarly. It is important to note that mutual compatibility is symmetric.

$$\tilde{C}(x_i, s_a, x_j, s_b) = \tilde{C}(x_j, s_b, x_i, s_a) = \frac{C(x_i, s_a, x_j, s_b) + C(x_j, s_b, x_i, s_a)}{2} \tag{14}$$

#### 5.1.2 Selecting the Seed Piece

Similar to Pomeranz *et. al.*, Paikin & Tal used a kernel growing algorithm. Hence, a seed piece is selected, and all pieces are placed around that seed. Since the algorithm is greedy, the selection of a poor seed can have a significant, deleterious 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 with its neighbors.

#### 5.1.3 Placement

Paikin & Tal utilized 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

**while** $|UnplacedPieces| > 0$ **do**
    **if** $|BestBuddyPool| > 0$ **then**
        Get best candidate from the *BestBuddyPool*
    **else**
        Recalculate the asymmetric and mutual compatibility
        Select piece with the highest asymmetric compatibility
    Place the best piece
    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 of correctness. As mentioned, two pieces are much more likely to be adjacent if they are best buddies. Due to this, the algorithm keeps a pool of the already placed pieces best buddies; whenever a new piece is placed, all of that piece's unplaced best buddies are added to the *BestBuddPool*. What is more, even among a set of best buddies,

As long as the *BestBuddyPool* is not empty, candidates for placement can only come from that pool; as the best candidate from the *BestBuddyPool* is the one with the maximum asymmetric compatibility with an open slot in the puzzle. This prioritizes placing the the best buddy pairings upon which there is the greatest confidence.

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 the 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 the expected number of puzzles. 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 its associated open slot drops below a predefined threshold (e.g. Paikin & Tal use 0.5) and the current number of puzzles is less than the expected number, 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, one was developed as part of this thesis. It is written entirely using Python 2.7 [22]. The following subsections describe the implementation's two Python packages.

#### 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: an identification number, size, type (e.g. type 1, type 2 - this is via the class `PuzzleType`), 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. Attributes of objects of this type are: an identification number, width (in number of pixels), rotation (via the class `PuzzlePieceRotation`), image information (stored in a NumPy array [24]), and location in the puzzle.

Additional features included in the `hammoudeh_puzzle` package include calculating and visualizing 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 5.1. The primary interface for the user is through the `PaikinTalSolver` class; objects of this type are created using a constructor that takes as parameters: the expected number of puzzles, a set of `PuzzlePiece` objects, a distance function, the puzzle type (class "`PuzzleType`"), and optionally a set of fixed puzzle dimensions. A `PaikinTalSolver` object is composed of a set of component object, which are described in the subsequent paragraphs.

An object of type `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 tune the solver's performance using different distance metrics. While calculating the asymmetric compatibilities, the `InterPieceDistance` class also finds each piece's best buddies; this made the `InterPieceDistance` class 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). However, the choice of data structure is critical as the algorithm must be able to quickly remove the best (i.e. maximum) value 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 in this Pool** - Stores the identification numbers of best buddies currently in the pool.
  A dictionary enables new best buddies to be quickly add best buddies to

the pool when a piece is placed and to delete the placed piece from the pool.

- **Dictionary of the Open Slots in the Puzzle** - Stores the set of valid locations pieces can be placed.
  Similar to the best buddy dictionary, a dictionary enables valid locations to be quickly added and removed.

- **Best Candidate Max Heap** - Used to rank the pairings of best buddies in the pool versus valid location pieces can be placed. 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.
  A max heap allows the best candidate to be found quickly in time $O(log(n)))$. 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 efficiently 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 valid location in the puzzle being previously filled). If the heap grows 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 puzzles 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 major, deleterious effect on the final result. An improved placer could prioritize the placement of pieces based off the number of best buddies that 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 the 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 spawned 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 is 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 to where in the placement the algorithm should backtrack and then 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–Ű127, 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*, 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," *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.," *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 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*, 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 *Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, CVPR '13, pp. 1767–1774, IEEE Computer Society, 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 *Proceedings of the 2014 European Conference on Computer Vision (ECCV)*, pp. 32–46, Springer, 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 Engineering*, vol. 13, pp. 22–30, Mar. 2011.