Solving Jigsaw Puzzles using Paths and Cycles

Lajanugen Logeswaran
lajanugenl.14@cse.mrt.ac.lk

Dept. of Electronic and
Telecommunication Engineering
University of Moratuwa
Sri Lanka

Abstract

There has been a growing interest in image jigsaw puzzles with square shaped pieces. A solver takes as input square shaped patches of the same size belonging to an image and attempts to reconstruct the image. The key components of a jigsaw solver are a compatibility metric and an assembly algorithm. A compatibility metric uses the color content of the image patches to identify which pairs of pieces are likely to be neighbors in the correct assembly. As the piece size gets smaller, it becomes increasingly difficult for any compatibility metric to confidently identify matching piece pairs. We attempt to exploit more contextual information offered by a compatibility metric compared to previous work to improve its neighbor identification accuracy. We introduce the concept of paths and cycles in jigsaw puzzles and show that they provide a means of identifying correct and incorrect matches. We propose refinement techniques based on this idea which improve the neighbor identification accuracy of a given compatibility metric. We also propose a means of combining the strengths of different compatibility metrics. Whereas recently proposed greedy solvers use the cost values produced by the metric directly to pick piece pairs for assembly, we define a new measure of piece pair compatibility and use it to guide a greedy solver. The proposed solver beats state of the art performance, achieving a mean improvement of more than 15% in absolute placement accuracy for puzzles with piece size 14x14.

1 Introduction

Jigsaw Puzzles have been in existence since long. The first jigsaw solver was proposed by Freeman & Gardner in 1964 for apictorial 9 piece puzzles [6]. In this particular variant no texture information is present in the puzzle pieces and assembly requires looking at the shape of the pieces to place them in the correct configuration. Solvers for pictorial jigsaw puzzles were later proposed which consider both the shape and the appearance or the appearance alone. Solving puzzles with more number of pieces making less assumptions has been the trend over the years.

Our focus is on the square shaped pieces variant, which has also been the concern of most recent work. A solver would take as input the square shaped patches of an image which are of the same size, and tries to assemble the pieces in order to construct the original image making use of the texture information in the pieces. The major components of such a solver are a compatibility metric and an assembly algorithm. A compatibility metric assigns

a numerical cost value to a given pair of puzzle pieces which represents how likely it is that the pieces are neighbors in a given configuration. The assembly algorithm makes use of the compatibility metric to determine the placement of pieces. Our work assumes that the orientation of the pieces is known as in [2, 8, 9]. But the ideas are easily extended to the case of unknown piece orientation.

Cho et al. [3] discuss several interesting applications of the square piece jigsaw in image editing and synthesis. They consider integrating a set of constraints on the puzzle pieces such as the location of certain pieces, the set of pieces to be considered for assembly and demonstrate applications such as subject reorganization, texture control, etc. Demaine & Demaine [5] show that the problem of solving jigsaw puzzles is NP-hard when there is uncertainty in the neighbor relationships. Possessing a well defined solution which is difficult to arrive at makes jigsaw solving an interesting research problem. Further, the appealing nature of the problem adds to the above justification.

Significant work on the square shaped pieces variant began only recently during the last decade. Alajlan [1] proposed a compatibility measure based on dynamic time warping and uses the Hungarian procedure for puzzle assembly. Cho et al. [4] propose a probabilistic solver which performs inference in a graphical model to obtain a solution. The compatibility metric used is based on the sum of squared differences along the abutting boundary of piece pairs. Yang et al. [10] use a similar formulation and propose a particle filter inference framework. Pomeranz et al. [8] propose the prediction based compatibility metric and suggest an algorithm that works in three phases - placement, segmentation and shifting. Gallagher [7] proposed a greedy assembly procedure based on the minimum spanning tree algorithm. A notable contribution of this work is the MGC (Mahalanobis Gradient Compatibility) metric, which is shown by the author to outperform other existing metrics by a large margin in neighbor identification accuracy. Andalo et al. [2] formulate a combinatorial optimization problem and minimize a defined energy function via gradient ascent. A solver based on genetic algorithms was proposed recently by Sholomon et al. [9].

A principal weakness identified in previous work is that the proposed algorithms do not attempt to exploit the full potential of the information provided by a compatibility metric. For instance, in the greedy assembly procedure of Gallagher [7] piece pairs are picked simply based on the (normalized) cost values of neighboring piece pair configurations. In [2, 9] the energy/fitness function of an assembly is defined as the sum of the cost values of neighboring piece pairs. Such restricted use of the compatibility metric can have an adverse impact on assembly performance, especially when the piece size is small in which case it is difficult for any metric to discriminate between piece pair candidates.

We demonstrate that significant contextual information can be extracted from compatibility metrics that can help produce improved assemblies. We also try to capture long range relationships between piece pairs, beyond just the immediate neighbors. The main contributions of this paper are as follows :

- We introduce the idea of paths and cycles in jigsaw puzzles and define a compatibility measure based on this idea.
- A *cost refinement* procedure is introduced that modifies the cost values suggested by a compatibility metric to improve neighbor identification accuracy.
- A *neighbor refinement* procedure is proposed that utilizes the properties of paths to correct some of the mistakes made by the metric.
- We propose a means of combining the strengths of multiple compatibility metrics.

We use the proposed compatibility measure as a heuristic to guide a greedy assembly algorithm and demonstrate the effectiveness of our techniques.

The rest of the paper is organized as follows. In section 2 we briefly discuss about compatibility metrics. Section 3 introduces the proposed techniques. We evaluate the proposed techniques in section 4 and conclude in section 5.

2 Compatibility Metrics

Define $\mathcal{D} = \{left, right, top, bottom\} = \{l, r, t, b\}$ - the set of directions in which we shall consider neighbors. Given two puzzle pieces x, y and a direction $d \in D$, let (x, y, d) refer to the configuration where puzzle piece y is to the immediate neighboring position of puzzle piece x in the direction indicated by $d \in \mathcal{D}$. A compatibility metric C returns a numerical value $C(x, y, d)$ which represents how likely it is for the pieces x, y to be in the above configuration in the correct assembly. The lower the value, the more likely that the configuration is correct. Let $N(x, d)$ represent the best neighbor suggested by the compatibility metric for piece x in direction d, i.e., $N(x, d) = \text{argmin}_y C(x, y, d)$. We shall refer to N as the neighbor matrix. The above terminology is used in the rest of our discussion.

We briefly discuss two metrics that will be relevant to our discussion. Assume puzzle pieces x, y are of size $P \times P$ (pixels x pixels). We describe the compatibility computation only for the configuration where y is to the immediate right of x. The computation for other configurations are easily interpreted.

- *Sum of Squared Differences (SSD)*
  A naive approach in designing a compatibility metric is to simply consider the dissimilarity of the color values across the abutting boundary of the two pieces :
  $$C(x, y, r) = \sum_{k=1}^{P} \sum_{c=1}^{3} (x(k, P, c) - y(k, 1, c))^2 \tag{1}$$
  We shall refer to the above computation in RGB colorspace as SSD in our discussion.
- *Mahalanobis Gradient Compatibility (MGC)*
  This metric was introduced by Gallagher [7] and is computed as follows :
  Let $G_L(k, c) = x(k, P, c) - x(k, P-1, c)$ (gradient at boundary of piece x)
  Each $G_L(k)$ is a vector of 3 values (corresponding to the 3 color channels). Let μ_L, S_L be the mean and covariance of these vector samples.
  Let $G_{LR}(k, c) = y(k, 1, c) - x(k, P, c)$ (gradient across boundary). Compute :
  $$D_{LR} = \sum_{k=1}^{P} (G_{LR}(k) - \mu_L) S_L^{-1} (G_{LR}(k) - \mu_L)^T \tag{2}$$
  Similarly D_{RL} is computed.
  Compatibility is defined as $C(x, y, r) = D_{LR} + D_{RL}$

3 Improving Compatibility Metrics

This section is organized as follows. First we discuss our idea of paths and cycles which form the basis of all our techniques in section 3.1. Next we describe a measure of confidence of neighboring piece pair configurations that we propose in section 3.2. Our cost refinement and neighbor refinement algorithms are presented in sections 3.3 and 3.4. Section 3.5 describes

our algorithm for combining the strengths of multiple metrics. The assembly procedure is discussed in section 3.6.

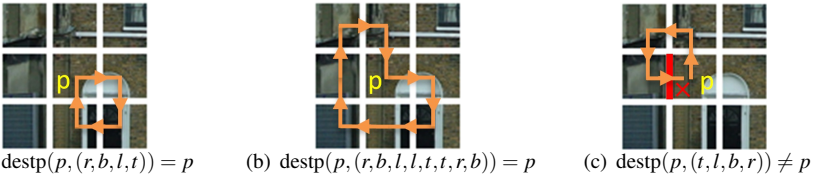
3.1 Paths and Cycles

Let $\mathbf{d} = (d_1, d_2, ..., d_n)$ be an ordered sequence of directions ($d_i \in \mathcal{D}$). Define a function $dest(\mathbf{d})$ which returns the location we would end up in if we start at a point and traverse unit distances sequentially in the directions specified by the d_i's, relative to the starting point. For instance, $dest((r,t)) = (1,1)$ and $dest((l,t,r,b)) = (0,0)$.

Define a function $path(x, \mathbf{d})$ which returns a sequence of puzzle pieces $(x_0, x_1, ..., x_n)$ where $x_0 = x, x_i = N(x_{i-1}, d_i)$ for $i = 1...n$. We define this sequence of pieces to be a ***path***, and we say that the *links* (x_{i-1}, x_i, d_i) make up the path. Define $destp(x, \mathbf{d}) = x_n$ to be the destination piece if we start from piece x and traverse through the sequence of directions $\mathbf{d}$.

Let us call a sequence of directions $\mathbf{d_c} = (d_1, d_2, ..., d_n)$ a *direction cycle* if, as indicated by the name, starting from a point and travelling unit distances in the directions indicated by the d_i's sequentially leads back to the point we started from, i.e., if $dest(\mathbf{d_c}) = (0,0)$. The subscript c shall be used to indicate direction cycles. For instance, the sequences (l,t,r,b) and (l,l,t,t,r,b,r,b) are observed to be direction cycles.

Let us call a sequence of pieces $\mathbf{X} = (x_0, x_1, ..., x_n)$ a ***cycle*** if there exists a direction cycle $\mathbf{d_c}$ for which $\mathbf{X} = path(x_0, \mathbf{d_c})$ and $x_n = x_0$. If all the entries of our neighbor matrix are correct, a direction cycle $\mathbf{d_c}$ will always yield a cycle - $destp(x, \mathbf{d_c}) = x$ for all pieces x for which the considered cycle lies completely within the dimensions of the puzzle. Because of incorrect entries in the neighbor matrix direction cycles may not always produce cycles. This is illustrated in Figure 1. In Figures 1(a) and 1(b) all neighbor relationships among the pieces shown have been identified correctly, and hence the indicated direction cycles produce cycles as shown in the captions. Figure 1(c) shows a situation where we have incorrectly identified neighbor relationships (the neighbors estimated for these pieces are not correct), and hence the direction cycle doesn't produce a cycle.



(a) $\text{destp}(p,(r,b,l,t)) = p$ (b) $\text{destp}(p,(r,b,l,l,t,t,r,b)) = p$ (c) $\text{destp}(p,(t,l,b,r)) \neq p$

Figure 1: In the first two instances direction cycles produce cycles. In (c) incorrectly estimated neighbors prevent the formation of a cycle.

Cycles can provide vital information about the neighbor relationships identified by a compatibility metric. For instance, if $destp(x, \mathbf{d_c}) \neq x$ for a particular piece x and direction cycle $\mathbf{d_c} = (d_1, d_2, ..., d_n)$ we can conclude that at least one of the relationships (x_{i-1}, x_i, d_i) ; $i = 1...n$ is incorrect, where $(x_0, x_1, ..., x_n) = path(x, \mathbf{d_c})$. On the other hand, if $destp(x, \mathbf{d_c}) = x$, then this makes the configurations (x_{i-1}, x_i, d_i) ; $i = 1...n$ likely to be correct. We note in passing that the idea of *best buddies* proposed by Pomeranz et al. [8] is a special case of a cycle - a cycle of length 2.

We shall use these ideas to identify and correct some of the incorrect neighbor relationships suggested by the compatibility metric and to aid assembly. In the following sections we discuss these techniques.

(a) Pieces with incorrect neighbor (b) cycle length = 2 (c) cycle length ≤ 4 (d) cycle length ≤ 6

Figure 2: Link Strengths of vertical links in puzzle for different values of maximum cycle length considered

3.2 Link Strength

Motivated by the idea of cycles, we attempt to define a measure of piece pair compatibility that represents more contextual information compared to the raw compatibility scores. We refer to a configuration (x,y,d) as a *link* and define its *strength* to be the number of cycles to which it belongs.

Figure 2(a) shows the puzzle pieces of an image in Cho et al.'s image database [4]. Also shown in the figure are the neighbor relationships incorrectly identified by a metric - A red marking on the edge of a piece indicates that the neighbor adjacent to the edge was incorrectly identified. This is a convenient way of visualizing correct and incorrect neighbor relationships and also provides insight into the nature of mistakes made by the compatibility metric. The strengths of the vertical links $(x,N(x,t),t)$ for all pieces x are shown in Figures 2(b), 2(c), 2(d) where the lengths of the cycles considered are at most 2,4 and 6 respectively. We observe how the different cycle lengths provide varied degrees of information. In a region of the puzzle where all neighboring relationships are correctly identified the strength of the links close to the center of the region are high. The strengths gradually diminish as we move towards parts with incorrect neighbors.

3.3 Cost Refinement

Consider a puzzle piece x with the following property : There is high ambiguity involved in finding the left neighbor of x, but the compatibility metric is able to confidently identify the correct left neighbor of the correct top/bottom neighbors of x (Other analogous situations are easily interpreted). This procedure attempts to use this information to distinguish the correct left neighbor of x from others.

Let x_l,x_t be respectively the correct left and top neighbors of x. Suppose all vertical neighboring relationships in the puzzle have been correctly identified by the metric. If the metric is able to confidently identify the correct left neighbor of x_t, then $C(x_t,N(x_t,l),l) << C(x_t,i,l)\ \forall i \neq N(x_t,l)$. We try the following modification to the cost values $C(x,i,l)\forall i$: $C(x,i,l)$ += $min[C(N(x,t),N(i,t),l),C(N(x,b),N(i,b),l)]$ in the expectation that the aforementioned property will help disambiguate between the candidates for the left neighbor of x. Similar updates are done to other piece pair configurations as well.

Although we made the assumption of vertical neighboring relationships being correct, most of them are incorrect in reality, and the above modification might adversely affect cost values that suggested correct neighbors previously. To prevent this situation, following the cost modification we check if the strength of each link $(x,N(x,d),d)$ has improved and revert to the old values of $C(x,y,d)\ \forall y$ if not. This refinement procedure is carried out iteratively until link strengths do not improve significantly.

(a) Independent paths (b) Paths sharing a common link

Figure 3: Direction sequences with a common *dest* value

3.4 Neighbor Refinement

A neighbor matrix is populated using our beliefs about the best piece candidate for the neighboring positions of a given piece. Whereas the compatibility metric assigns numerical values to each piece candidate that may appear as the neighbor, entries of the neighbor matrix are point estimates of the neighbors of each piece. Through this neighbor refinement technique we attempt to correct erroneous entries in the neighbor matrix using correct ones.

Consider direction sequences $\mathbf{d}$ which have a common *dest* value $dest(\mathbf{d}) = k$ (For instance, notice that $dest(l,t,r) = dest(r,t,l) = dest(t)$). These direction sequences can be visually thought of as leading to the same point when starting from a particular point and traversing along the directions as illustrated in Figure 3(a). For a given puzzle piece x, these direction sequences suggest candidates $destp(x,\mathbf{d})$ for the piece located at position k relative to x. Intuition suggests that the number of direction sequences which suggest the same piece at this location is correlated with the confidence that the piece is the correct candidate. But the following issue needs to be considered - paths produced by such direction sequences with all links correct except for one single incorrect link that is shared by all of them will suggest the same, albeit incorrect piece (Figure 3(b)). Hence there is a possibility of high beliefs being assigned to incorrect placements in this approach. This motivates us to look at independent paths (paths which do not share any links in common) that suggest the same piece for a particular location. There can be a maximum of 4 such paths since the paths need to contain either of the four links $(x,d); d \in \mathcal{D}$.

We use this idea to identify and correct some of the incorrect neighbor relationships. For each piece x and direction d, we consider direction sequences $\mathbf{d}$ for which $dest(\mathbf{d}) = d$ (E.g. For $d = left$, we may consider direction sequences such as $(l),(t,l,b),(b,l,t),(b,l,l,t,r)$... etc). The best neighbor in direction d is decided based on the highest number of tuples (pairs, triples, quads) of independent paths which suggest the same piece. As in the case of cost refinement, after updating the N matrix, for each link we check if its strength has improved and revert to the old value if not.

3.5 Combining Compatibility Metrics

Different compatibility metrics may use different image features as key to discriminate correct piece pairs from others. Although one metric may perform better than another when considering the performance on all pieces as a whole, each has its own strengths and performs well on pieces with certain properties. We computed compatibility values using several metrics and found that the number of correct piece pairs identified by them taken together (considering a pair to have been correctly identified if atleast one of the metrics correctly identified it) was considerably more than the number of pairs correctly identified by any particular one of them for a given puzzle. The difference is more pronounced in the case of smaller piece size.

We intend to use the idea of cycles to exploit the strengths of multiple compatibility metrics. We consider multiple metrics and compute the neighbor matrix for each. For each link

(x, x_d^m, d) suggested by each compatibility metric m, we compute its link strength. During this strength computation, all the links suggested by all the metrics are taken into account. For instance, two metrics m_1, m_2 may disagree on the neighbor for a particular piece, in which case both the links suggested by them $(x, x_d^{m_1}, d), (x, x_d^{m_2}, d)$ are used in the strength computation. We found this to be more effective compared to computing the strength values of individual metrics separately. A new neighbor matrix is constructed as follows : for each piece and direction, the neighbor suggested by either of the metrics with the largest strength is taken to be the neighbor.

We considered the following four metrics in experimenting with the metric combining procedure :

- SSD
- MGC
- a metric that penalizes change in the laplacian instead of the gradient in Equation 2
- a metric that penalizes change in both the horizontal and vertical directions at the boundary instead of only the gradients in the direction perpendicular to the boundary in Equation 2

The motivation behind considering these metrics is that they penalize change in different image features (color intensity, gradients, laplacian).

3.6 Assembly

To demonstrate the performance of our techniques on puzzle assembly, we use the tree based assembly procedure of Gallagher [7] with the following modifications : Instead of picking piece pairs based on the normalized cost matrix, we use link strengths. For a given neighbor matrix, we calculate the link strengths and pick piece pairs in the order of higest to lowest link strengths during assembly. The normalized scores are used to break ties. This is done until the strength of a chosen pair falls below a threshold. After this procedure we obtain beliefs for the placement of the remaining parts in free locations using paths that originate at pieces that have already been placed. Based on these beliefs we place more pieces until the largest number of paths that suggest a particular placement fall below a threshold. Finally, the trimming and hole filling procedures of Gallagher are used to obtain rectangular puzzle assemblies.

4 Experimental Results

We use the database of 20 images each of dimensions 672x504 provided by Cho et al. [4] for our experimentation. We consider the two piece sizes previous work have considered : 28x28 [7, 8, 9] and the more challenging 14x14 [7]. For these piece sizes, the puzzle sizes are 432 and 1728 respectively. We shall refer to these two types of puzzles as type(a) and type(b) puzzles respectively.

4.1 Neighbor identification accuracy

Neighbor identification accuracy is a commonly used metric to estimate the performance of a compatibility metric (E.g. : Similarity performance in [7]). It is the percentage of correctly identified neighbors of pieces in a puzzle. Table 1 shows the neighbor identification performance of a metric (raw scores) and the cumulative benefits of applying our cost refinement and neighbor refinement algorithms. Cost refinement is first applied to the cost

Table 1: Mean neighbor accuracy values of (i) Scores suggested by compatibility metric (ii) Scores after *cost refinement* (iii) Neighbor matrix after *neighbor refinement*.

Piece Size	Puzzle Size	Raw Scores	*Cost refinement*	*Neighbor refinement*
MGC				
28	432	90.2%	93.1%	94.9%
14	1728	72.5%	81.8%	86.5%
SSD				
28	432	78.9%	85.6%	89.8%
14	1728	57.8%	64.0%	67.2%

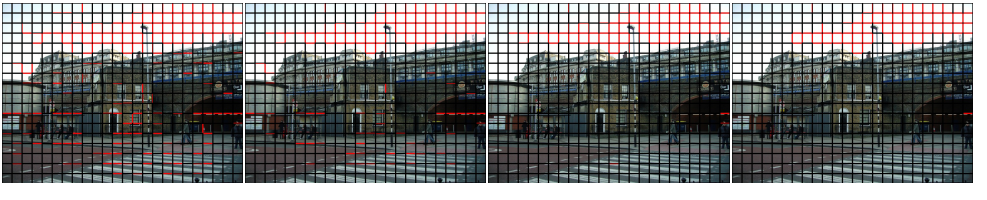
values produced by the metric. Neighbor accuracy values corresponding to the best candidates suggested by the refined scores are shown in column 2. The neighbor matrix suggested by the refined scores is then processed by the neighbor refinement procedure to obtain a new neighbor matrix, whose neighbor accuracy values are shown in column 3. These values have been averaged across the 20 images. The SSD and MGC compatibility metrics have been considered.

Cost Refinement and Neighbor Refinement : For type(a) puzzles we observe a 4.7% increase for MGC scores and a 10.8% improvement for SSD scores. The raw MGC scores perform well, leaving less room for improvement. On type(b) puzzles the raw MGC scores perform not as well, and a 14% improvement is observed. The corresponding improvement for SSD is less (9.4%) in this case because the raw scores perform poorly and the large number of incorrect relationships limit the ability of paths and cycles in making correct inferences. In general, the improvement for a given metric is significant when the piece size is smaller as there is more room for improvement.

Combining Compatibility Metrics : The four metrics discussed in Section 3.5 were first processed with score refinement and neighbor refinement and then the metric combining procedure was applied. Mean improvements of 1% and 4% were observed respectively for type(a) and type(b) puzzles compared to MGC alone. A peak improvement of 11% was observed for a particular type(b) puzzle.

Aggregate Performance : The (mean,peak) aggregated improvement in neighbor accuracy due to all the proposed techniques are (5.7% , 13.7%) and (18.4% , 26%) respectively for piece sizes 28 and 14. Notice the significant improvement for type(b) puzzles. Moreover, perfect(100%) neighbor identification accuracy is achieved for 14 type(a) puzzles and 1 type(b) puzzle. For these puzzles a simple greedy procedure which picks piece pairs in the order of decreasing link strengths would produce perfect reconstructions.

Figure 4 illustrates the result of applying each improvement procedure on a particular type(a) puzzle. Corresponding neighbor accuracy figures are mentioned in the captions. We



(a) Raw MGC Scores 77% (b) cost refinement 83% (c) neighbor refinement 87% (d) combining metrics 88%

Figure 4: Incorrectly estimated neighbors (indicated by a red marking on piece edges) after application of each improvement procedure - Image 1 of Cho et al.'s database [4] (piece size = 28)

Table 2: Comparison of puzzle assembly on Cho et al.'s image database [2] for piece size = 28

	Direct	Neighbor	Component	Perfect Rec.
Pomeranz et al. [8]	94%	95%	-	13
Andalo et al. [1]	96%	95.6%	-	13
Gallagher [7]	95.3%	95.1%	95.3%	12
Sholomon et al. [9]	86.2%	96.2%	-	-
Ours	96.3%	96.4%	96.2%	15

Table 3: Comparison of puzzle assembly on Cho et al.'s image database [2] for piece size = 14

	Direct	Neighbor	Component	Perfect Rec.
Pomeranz et al.	29.3%	45.8%	35.3%	0
Gallagher	50.6%	73.9%	63.0%	0
Ours	70.3%	90.0%	87.0%	3

observe that majority of the incorrectly identified neighbors after applying all techniques are in the constant sky region of the image. This can be expected because of the high ambiguity involved in finding correct neighbors in this region, and we can claim that ideal performance has been achieved for this particular puzzle.

4.2 Puzzle Assembly

We evaluate puzzle assembly in terms of the four evaluation metrics used by previous researchers :

- *Direct Comparison* : Correct absolute placements
- *Neighbor Comparison* : Correct pairwise adjacencies in assembly
- *Largest Component* : Largest component with correct pairwise adjacencies
- *Perfect Reconstruction* : Number of perfect reconstructions

We compare the assembly performance of our approach with several recent algorithms using the afore-mentioned metrics in Tables 2, 3. Citations indicate values obtained from the respective publications. Metric values not provided by the authors are left blank.

We were unable to perform comparisons with [1], [9] on type(b) puzzles as scores were not reported for this type of puzzles and implementations haven't been made available by the authors to the best of our knowledge. Gallagher and Pomeranz et al. have made their implementations publicly available and we were able to perform runs on these puzzles. Because of the random component in Pomeranz et al.'s algorithm, we ran it 10 times with a random seed on each image and chose the assembly that produced that highest best buddies score.

We observe that the scores of our approach are consistently higher than that of state of the art. The difference is more pronounced in the case of smaller piece size. Whereas state of the art techniques achieve a maximum of 13 perfect reconstructions for type(a) puzzles, we achieve 15 perfect reconstructions. For type(b) puzzles, while the state of the art is not capable of perfectly reconstructing any of the puzzles, we achieve 3 perfect reconstructions. A visual comparison of some type(b) puzzles are presented in Figure 5. Our assemblies are observed to have significantly better visual quality.

We further experimented with the 20 images of the McGill database [8] considering puzzles with piece-size of 14 (2160 pieces) and observed improvements of (12%,7%,14%) in (direct, neighbor, component) scores relative to Gallagher's assemblies. Further detailed numerical and visual results on individual images are presented in our supplementary document.

Experiments were performed on a Xeon 2.5GHz 4-core workstation with 30GB memory. Our solver took 15min, 30min on average for a type(a) and type(b) puzzle respectively (approximately double the time taken by Gallagher's implementation on the same machine). The lengths of paths and cycles considered in all algorithms were at most 10. While this limits the ability of the algorithms, the complexity of finding paths and cycles increases exponentially with length. We hope to consider the design of efficient algorithms that exploit paths of greater lengths in future.

5 Conclusion

In this paper we propose the idea of paths and cycles in jigsaw puzzles and demonstrate its ability in extracting useful information from compatibility metrics. The proposed cost refinement and neighbor refinement algorithms improve the neighbor identification accuracy of a given compatibility metric. Our metric combining technique utilizes the strengths of different compatibility metrics to estimate accurate piece pair relationships. The proposed techniques together produce perfect neighbor identification for a higher number of puzzles than the state of the art assembly algorithms are able to perfectly reconstruct. A greedy solver guided by our measure of link strengths beats state of the art performance, doing so by a large margin for puzzles with smaller piece size.

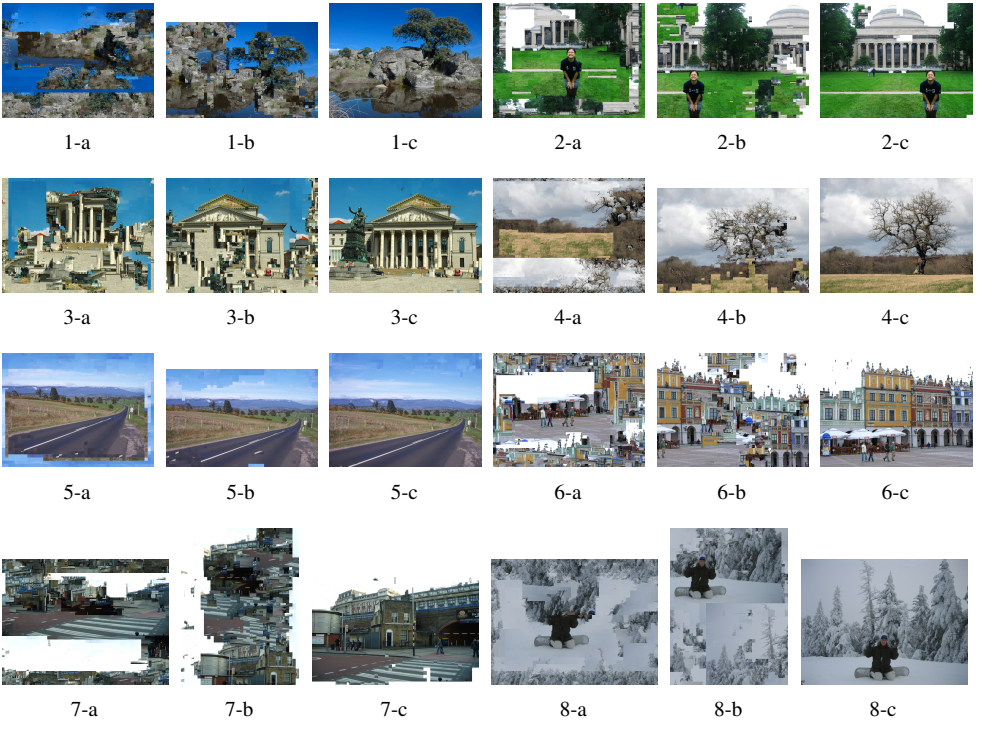


Figure 5: Visual comparison of 14 piece-size puzzle assembly. Puzzles belong to Cho et al.'s database [2]. In each set of 3 images a,b,c indicate the puzzle assemblies of Pomeranz et al., Gallagher and ours respectively.

Acknowledgements

I am thankful to Ranga Rodrigo for his support and helpful feedback. I also thank Andrew C. Gallagher and Dolev Pomeranz for making their code publicly available.

References

[1] Naif Alajlan. Solving square jigsaw puzzles using dynamic programming and the hungarian procedure. pages 1941–1947. American Journal of Applied Sciences, 2009.

[2] Fernanda A. Andalo, Gabriel Taubin, and Siome Goldenstein. Solving image puzzles with a simple quadratic programming formulation. In *SIBGRAPI*, pages 63–70. IEEE Computer Society, 2012.

[3] Taeg Sang Cho, Moshe Butman, Shai Avidan, and William T. Freeman. The patch transform and its applications to image editing. In *CVPR*. IEEE Computer Society, 2008.

[4] Taeg Sang Cho, Shai Avidan, and William T. Freeman. A probabilistic image jigsaw puzzle solver. In *CVPR*, pages 183–190. IEEE, 2010.

[5] Erik Demaine and Martin Demaine. Jigsaw puzzles, edge matching, and polyomino packing: Connections and complexity. *Graphs and Combinatorics*, 23(0):195–208, 2007.

[6] H. Freeman and L. Gardner. Apictorial jigsaw puzzles: the computer solution of a problem in pattern recognition. pages 118–127. Electronic Computers 13, 1964.

[7] Andrew C. Gallagher. Jigsaw puzzles with pieces of unknown orientation. In *CVPR*, pages 382–389. IEEE, 2012.

[8] Dolev Pomeranz, Michal Shemesh, and Ohad Ben-Shahar. A fully automated greedy square jigsaw puzzle solver. In *CVPR*, pages 9–16. IEEE, 2011.

[9] Dror Sholomon, Omid David, and Nathan S. Netanyahu. A genetic algorithm-based solver for very large jigsaw puzzles. In *CVPR*, pages 1767–1774. IEEE, 2013.

[10] Xingwei Yang, Nagesh Adluru, and Longin Jan Latecki. Particle filter with state permutations for solving image jigsaw puzzles. In *CVPR*, pages 2873–2880. IEEE, 2011.