An Algorithm for Creating Geometric Dissection Puzzles

Yahan Zhou, yhzhou@cs.umass.edu
Rui Wang, ruiwang@cs.umass.edu
Department of Computer Science, UMass Amherst

Abstract

Geometric dissection is a popular category of puzzles. Given two planar figures of equal area, a dissection seeks to partition one figure into pieces that can be reassembled to construct the other figure. In this paper, we present a computational method for creating lattice-based geometric dissection puzzles. Our method starts by representing the input figures on a discrete grid, such as a square or triangular lattice. Our goal is then to partition both figures into the smallest number of clusters (pieces) such that there is a *one-to-one* and *congruent* matching between the two sets of clusters. Solving this problem directly is intractable with a brute-force approach. We propose a hierarchical clustering method that can efficiently find near-optimal solutions by iteratively minimizing an objective function. In addition, we modify the objective function to include an area-based term, which directs the solution towards pieces with more balanced sizes. Finally, we show extensions of our algorithm for dissecting 3D shapes of equal volume.

1 Introduction

Geometric dissection is a mathematical problem that has enjoyed for a long time great popularity in recreational math and art [4,7]. In its basic form, the dissection question asks whether any two shapes of the same area are equi-decomposable, that is, if they can be cut into a finite number of congruent polygonal pieces [4]. Such problems have a rich history originating from the exploration of geometry by the ancient Greeks. One of the earliest examples is a visual proof of the Pythagorean theorem. In Arabic-Islamic mathematics and art, dissection figures are frequently used to construct intriguing patterns ornamenting architectural moments [9]. Dissection figures also provide a popular way to create puzzles and games. The Tangram, which is a dissection puzzle invented in ancient China, consists of 7 pieces cut from a square and then rearranged to form a repertoire of other shapes. A closely related subject to geometric dissections is tiling, which seeks to find a collection of figures that can fill the plane infinitely with no overlaps or gaps. The use of tiling is ubiquitous in the design of patterns throughout the history of art, especially in the drawings of M.C. Escher.

Mathematically, it has long been known that a dissection always exists for 2D polygons, due to the Bolyai-Gerwien theorem [2,5,8]. Although the theorem provided a general solution to find dissection solutions, the upper bound on the number of pieces is quite high. In practice, many dissections can be achieved with far fewer pieces (such as the example shown in Figure 2). Therefore, much recent work has focused on the challenge of finding optimal dissections using as few pieces as possible, and this has inspired extensive research in the mathematical and computational geometry literature [1, 3, 4, 6, 7]. While many ingenious analytic solutions have been discovered for shapes such as equilateral triangles, squares and other regular polygons, finding the optimal solution for general shapes remains a difficult open research problem.

Our goal is to find a computational algorithm for solving geometric dissections. We focus on a special class of the problem where shapes are represented on discrete square or triangular lattice. We introduce an optimization-based algorithm to iteratively minimize an objective function. The algorithm can be extended to support area balancing or create 3D dissection puzzles. Figure 1 shows two examples.

2 Algorithms and Implementation

Assumptions and Overview. Given two input figures $\mathcal{A}$ and $\mathcal{B}$ of equal area, our goal is to find the minimum set of pieces to dissect $\mathcal{A}$ and $\mathcal{B}$. To formulate it as an optimization problem, we require both

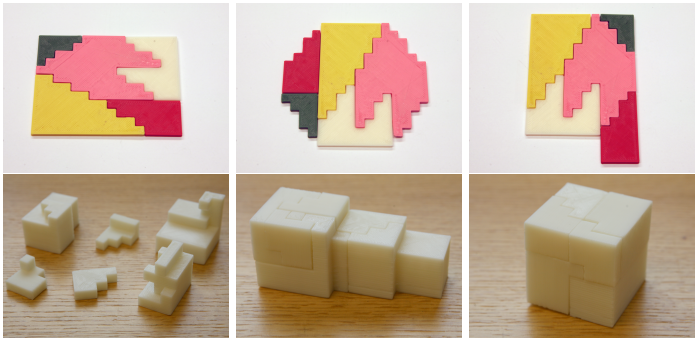


Figure 1: Dissection puzzles created using our algorithm. The top row shows a 2D example, using five pieces to construct three shapes. The bottom row shows a 3D example with six pieces. The two shapes constructed from them demonstrates $3^2+4^2+5^2=6^2$.

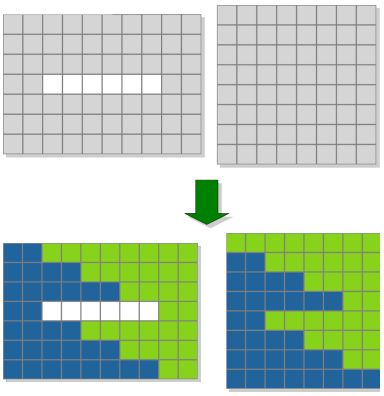


Figure 2: Classic example of dissection using only two pieces.

input figures to be represented on a discrete grid. The simplest choice is a square lattice. Note that after discretization, the area (number of pixels) covered by each figure must remain the same. This can be ensured either by the design of the input figures, or by using a graphical interface to touch up the rasterized figures. In the following, we use symbols $\mathcal{A}$ and $\mathcal{B}$ to denote the two rasterized figures of equal area.

Given the input, we formulate the dissection problem into a cluster optimization problem. Specifically, our goal is to partition each figure into the smallest number of clusters (each cluster being a connected piece) such that there is a one-to-one and congruent matching between the two cluster sets. Here congruency means two pieces must match exactly under isometric transformations, including translation, rotation, and flipping. Since the solution space is discrete, all possible transformations are also discrete. For example, on a square lattice with grid size 1, all translations must have integer values, and there are only 4 possible rotations, namely, multiples of 90°. Thus excluding translation, two congruent pieces must match under at least one of the 8 combinations of rotation and flipping.

To solve the problem, we introduce a hierarchical clustering algorithm that progressively minimizes an objective function until a solution is found. We start the search from a random initial condition, and apply refinement steps to iteratively reduce the objective function value. We use random exploration to keep the algorithm from getting stuck in local minima. Figure 3 provides a graphical overview of the algorithm, which is explained in detail in the next sections.

2.1 Dissecting Two Figures on a Square Lattice

Distance metric. Given two pieces (i.e. clusters) from each figure: $\mathbf{a} \subset \mathcal{A}$, $\mathbf{b} \subset \mathcal{B}$, we define a distance metric D that measures the bidirectional mismatches between them under the best possible alignment:

$$D(\mathbf{a},\mathbf{b}) = \min_{T_{\mathbf{a},\mathbf{b}}} \left(\left\| \{ p \,|\, p \in \mathbf{a} \text{ and } (T_{\mathbf{a},\mathbf{b}} \times p) \notin \mathbf{b} \} \right\| + \left\| \{ p \,|\, p \in \mathbf{b} \text{ and } (T_{\mathbf{a},\mathbf{b}}^{-1} \times p) \notin \mathbf{a} \} \right\| \right) \tag{1}$$

where $T_{\mathbf{a},\mathbf{b}}$ is an isometric transformation from piece $\mathbf{a}$ to $\mathbf{b}$, $T_{\mathbf{a},\mathbf{b}}^{-1}$ is the reverse transformation, p counts the number of pixels that are in one piece but not the other (i.e. it measures bidirectional mismatches). As D measures the minimum mismatches under all possible $T_{\mathbf{a},\mathbf{b}}$, it will be 0 if the two pieces are congruent.

To simplify the calculation of D, we first set the translation to align the (bounding box) centers of $\mathbf{a}$ and $\mathbf{b}$, then search among all valid rotations and flippings to obtain D. While this does not consider all possible translations, we found it to work well as a heuristic, and it guarantees to be equal to zero when two pieces are congruent. Note that if the center of a piece does not lie exactly on a grid point, we need to align it to the 4 nearby grid points and calculate D for each; the smallest among them is returned as the distance.

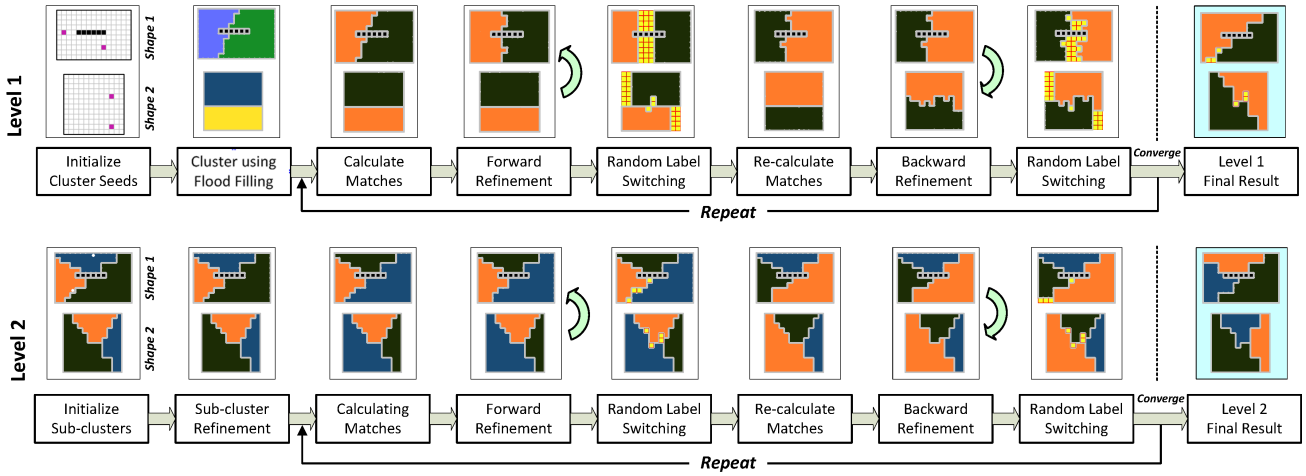


Figure 3: An overview of our hierarchical optimization method for dissecting two figures: one is a 10×15 rectangle with a 1×6 off-center hole, the other is a 12×12 square. At each level, we visualize the steps of computing one candidate solution. Matched clusters are shown in the same color. Mismatched pixels are shown with a yellow color and red outline.This example requires 3 pieces to dissect, which was found by the algorithm at the end of level 2.

Matching. Assume $\mathcal{A}$ and $\mathcal{B}$ are both partitioned into k clusters $\{\mathbf{a}_i\}$ and $\{\mathbf{b}_j\}$, we now need to pair up each element in $\{\mathbf{a}_i\}$ with one in $\{\mathbf{b}_j\}$ such that the sum of distance between every matched pair is minimized. We call this a *matching* between the two sets, denoted as M. Mathematically,

$$M = \underset{m \in \{\{\mathbf{a}_i\} \to \{\mathbf{b}_j\}\}}{\arg\min} \sum_{(\mathbf{a}_i, \mathbf{b}_j) \in m} D(\mathbf{a}_i, \mathbf{b}_j) \tag{2}$$

where $\{\mathbf{a}_i\} \to \{\mathbf{b}_j\}$ denotes a bijection from $\{\mathbf{a}_i\}$ to $\{\mathbf{b}_j\}$. Basically we are seeking among all possible bijections the one that gives rise to the minimum total distance. This is known as the assignment problem in graph theory, which is well-studied and can be solved by a maximum weighted bipartite matching. Specifically, we create a weighted bipartite graph between the two sets $\{\mathbf{a}_i\}$ and $\{\mathbf{b}_j\}$: every element $\mathbf{a}_i$ in $\{\mathbf{a}_i\}$ is connected to every element $\mathbf{b}_j$ in $\{\mathbf{b}_j\}$ by an edge, whose weight is equal to the distance D between the two elements. The goal is to find a bijection whose total edge weight is minimal. A common solution is based on a modified shortest path search, for which we use an optimized Bellman-Ford algorithm [10]. It guarantees to yield the best matching in $O(k^3)$ time, where k is the number of clusters.

We call the total pair distance under M the *matching score*, denoted as $E_M = \sum_{(\mathbf{a}_i, \mathbf{b}_j) \in M} D(\mathbf{a}_i, \mathbf{b}_j)$. Note that if $E_M = 0$ then M is a dissection solution. Thus the smaller E_M is, the closer we are to a solution.

Objective function. Since the minimum number of pieces to achieve a dissection is unknown in advance, we propose a hierarchical approach that solves the problem in multiple levels. Each level ℓ partitions the two input figures into $\ell + 1$ clusters, and outputs a set of best candidates. Our objective function is simply the matching score E_M. Specifically, let's denote with $C_k = (\{\mathbf{a}_i\}_k, \{\mathbf{b}_j\}_k)$ a candidate solution where $\{\mathbf{a}_i\}_k$ and $\{\mathbf{b}_j\}_k$ are two given k-piece clusterings of $\mathcal{A}$ and $\mathcal{B}$ respectively. Then the objective function $E_k(C)$ is:

$$E(C_k) = E_M(\{\mathbf{a}_i\}_k, \{\mathbf{b}_j\}_k) \tag{3}$$

At the end of each level ℓ, we select a set of N_b best candidate solutions $\{S_\ell\}$ which give the smallest values according to Equation 3, and use the set for initialization in the next level. The algorithm will terminate when a solution is found such that $E(S_\ell) = 0$. In the following we will describe our algorithms for the first and each subsequent level. Refer to Figure 3 for a graphical illustration.

2.2 Level 1 Optimization

Seeding. At the first level, our goal is to compute the best set of 2-piece clusterings to approximate the dissection. To begin, we split $\mathcal{A}$ into two random clusters $\mathbf{a}_1$ and $\mathbf{a}_2$. This is done by starting from two random seeds, and growing each into a cluster using flood fill (also known as seed fill), which ensures that each cluster is a connected component. We do the same for $\mathcal{B}$, resulting in two random clusters $\mathbf{b}_1$ and $\mathbf{b}_2$.

Matching. Now that we have the initial clusters $\{\mathbf{a}_i\}_2$ and $\{\mathbf{b}_j\}_2$, we invoke Equation 2 to compute the matching M between them. In Figure 3 we use the same color to indicate a matched pair. Note that there is no particular ordering of the clusters, so the colors may flip depending on the matching algorithm.

Forward copy-paste. Our next step is to refine the clusters. As the solution space is large, randomly modifying each cluster individually is unlikely to give a better matching score. To actively force their shapes to become similar, we use a more explicit approach that copies and pastes a cluster $\mathbf{a}_i$ to its matched cluster $\mathbf{b}_j$. This is called a *forward copy-paste*. To do so, we apply the transformation that yields the minimum distance between $\mathbf{a}_i$ and $\mathbf{b}_j$ (Eq. 1) to $\mathbf{a}_i$, and pastes the result to $\mathcal{B}$. Note that if the two matched clusters are not yet congruent, the pasting may overwrite some pixels that belong to neighboring clusters. This is allowed, but we randomize the order of clusters for copy-paste to avoid bias. Pixels pasted outside the figure boundary are ignored. Following this step, some pixels in $\mathcal{B}$ may have received no pasted pixels from $\mathcal{A}$, thus they become holes. We use a random flood-fill to eliminate the holes. Specifically, we randomly select already pasted pixels and grow them outward to fill the hole.

Random label switching. As mentioned above, during copy-paste, some clusters may overlap with each other, resulting in conflicts. Our next step is to reduce such conflicts by modifying the cluster assignments for some pixels at the boundary of two clusters. To do so, we first recompute the matching between the current two sets of clusters, then simulate a copy-paste in the backward direction, i.e. from $\mathcal{B}$ to $\mathcal{A}$. During this process we record the pixels that would have overlapped after pasting. For each such pixel x, we randomly relabel it to the cluster of one of its four neighboring pixels. This is called *random label switching*. Note that if x is surrounded completely by pixels of its own cluster, its label will remain the same. Thus only pixels on the boundary of a cluster can potentially be switched to a different label.

Intuitively, the motivation of the forward copy-paste is to encourage the clusters in $\mathcal{B}$ to be shaped similarly to the clusters in $\mathcal{A}$, and the random label switching is to modify the cluster boundaries in $\mathcal{B}$ to reduce cluster conflicts/overlaps. The two steps combined is called a **forward refinement** step.

Backward refinement. The backward refinement performs exactly the same steps as the forward refinement, except in the reverse direction (i.e. a copy-paste from $\mathcal{B}$ to $\mathcal{A}$, followed by a random labeling switching in $\mathcal{A}$). At this point, we have completed one iteration of back-and-forth refinement.

Convergence. We repeat the back-and-forth refinement iteration for R times (the default value of R is 100). This typically reaches convergence very quickly, upon which we obtain a candidate solution C_2, whose associated objective function value is $E(C_2)$.

Random seed exploration. The refinement process can be seen as a way to find local minima from the initial seeds. Small changes to the initial seeds do not significantly affect the converged result. In order to seek a global minimum, we apply random exploration, where we re-compute the candidate solution N times (the default value of N is 400), each time with a different set of initial seeds. After random exploration, the best $N_b = 30$ candidate solutions (i.e. those with the smallest objective function values) are selected and output as the level 1 final results, denoted as $\{S_1\}$.

At this point, if there exists a candidate solution whose matching score is 0, we have found a complete dissection solution. Otherwise we continue with subsequent levels. The top portion of Figure 3 illustrates all steps in level 1. Note how the candidate solution refines following each step. The red outlines on some pixels indicate unmatched pixels between a pair of clusters which are not yet congruent.

2.3 Level ℓ Optimization

In each subsequent level ℓ, we start from one of the candidate solutions S_ℓ from the last level. Our goal is to insert a new cluster to S_ℓ, and then search for the best $(\ell+1)$-piece approximation using the same back-and-forth refinement process as in level 1. Intuitively, as the candidates from the previous level are some of the closest ℓ-piece approximations to the dissection, they serve as excellent starting points for the current level.

The main difference between level ℓ and level 1 is in creating the initial clusters. Note that we cannot use completely random initialization as in level 1, as doing so will abandon the results discovered in previous levels, and hence will not reduce the problem's complexity. Instead, we introduce two heuristics for initialization by exploiting the previous results, and we consider them both during the random exploration.

Splitting an existing cluster. In the first heuristic, we select a pair of pieces $\{\mathbf{a}_i, \mathbf{b}_j\}$ from S_ℓ with the largest (worst) matching score, and split each into two sub-clusters. We refer to the pair as the parent clusters. The splitting introduces an additional cluster for each figure; the other clusters that were not split remain unchanged for now. To decide how to split, a straightforward way is random splitting, but as the parent clusters are not well matched, a random split can create difficulties for convergence in subsequent steps. Therefore we need to optimize the splitting to create better matched sub-clusters.

It turns out that we can optimize the splitting by using the same approach as level 1 optimization. To do so, we treat the parent clusters $\mathbf{a}_i$ and $\mathbf{b}_j$ as two input figures, and apply level 1 optimization to obtain the best 2-piece dissection between them. This results in sub-clusters that are much better matches than random splitting. Our experiments show that this can significantly improve the quality of the subsequent refinement.

Creating new clusters from mismatched pixels. Our second heuristic is to create a new cluster from the mismatched pixels. For example, assume $\{\mathbf{a}_i, \mathbf{b}_j\}$ are a matched pair but not yet congruent, then transforming $\mathbf{a}_i$ to $\mathbf{b}_j$ will likely result in some pixels that are not contained in $\mathbf{b}_j$. These pixels will be marked in $\mathbf{a}_i$ as mismatched pixels. In Figure 3 the mismatched pixels are shown with a yellow color and red outline. After marking all mismatched pixels in $\mathcal{A}$ and $\mathcal{B}$, we randomly select a seed from them and perform a flood fill to grow the seed into a cluster, which then becomes a new cluster to be inserted to the current level.

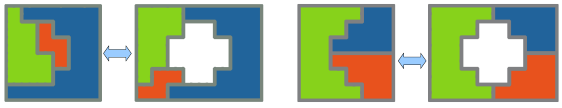
Comparing the two heuristics. The rationale behind the first heuristic is that priority should be given to splitting the worst matched pair, as this will likely reduce the matching score. The rationale behind the second heuristic is that when a candidate solution is very close to reaching a complete dissection, priority should be given to the few pixels that remain unmatched. In practice, we account for both of them during our random exploration: among the N random tries, 75% will use the first heuristic to initialize the sub-clusters, and 25% will use the second heristic. This way we can combine the advantages of both.

Global refinement and random exploration. Once the sub-clusters are created, we perform the same back-and-forth refinement process as in level 1. Now all clusters will participate in the refinement, therefore we call this step global refinement. Upon convergence, we obtain a candidate solution $C_{\ell+1}$.

Similar to level 1, we perform random exploration for N times, the goal of which is to seek a global minimum. Each exploration starts from a randomly selected candidate S_ℓ from the previous level, applies one of the two heuristics to insert a new cluster, and computes refinement. Again, after random exploration, the best N_b candidate solutions are output as the results $\{S_\ell\}$ of level ℓ. Figure 3 shows an example of level 2 optimization. For this example, our algorithm discovered a complete dissection at the end of level 2, thus the program terminates with a 3-piece dissection.

2.4 Extensions

Area-based Term. So far we have described an algorithm for finding the minimum dissection of two figures. However, there is no constraint on the size of each resulting piece. A solution may contain pieces that are significantly larger than others. This is often undesirable, both for aesthetic reasons and for reducing the



without area term (3 pcs) with area term (3 pcs)

Figure 4: Comparing solutions computed with and without area-based term. Note how the area-based term leads to pieces with more balanced size.

(a) Heart to Key (6 pcs) (b) H to House (6 pcs)

Figure 5: Dissections using a triangular lattice.

difficulty of the puzzles (i.e. large pieces are easier to identify and place on a target figure). For better control of the solution, we introduce an area-based term into our objective function to favor a solution where the size of each piece is more balanced. To do so, we incorporate the area-based term E_α into Equation 3:

$$E = E_M(\{\mathbf{a}_i\}_k, \{\mathbf{b}_j\}_k) + \lambda \cdot [E_\alpha(\{\mathbf{a}_i\}_k) + E_\alpha(\{\mathbf{b}_j\}_k)] \tag{4}$$

where λ is a weight adjusting the relative importance of the two terms. Here E_α is the total area penalty. It is defined by summing up the area penalty $\alpha(\mathbf{a}_i)$ of each piece, which is calculated as:

$$\alpha(\mathbf{a}_i) = \begin{cases} A(\mathbf{a}_i)/\bar{A} - 1 & \text{if } A(\mathbf{a}_i) > 2\bar{A} \\ \bar{A}/A(\mathbf{a}_i) - 1 & \text{if } A(\mathbf{a}_i) < \bar{A}/2 \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

where A denotes the area (i.e. number of pixels) of a piece, and $\bar{A}$ denotes the average area (i.e. the total area divided by the number of pieces). Essentially $\alpha(\mathbf{a}_i)$ penalizes a piece if it is either more than twice the average area, or less than half of the average area; otherwise we consider a piece to be within the normal range of size variations and assign a zero penalty.

Note that the preference towards balanced area and the preference towards smaller number of pieces are often conflicting goals. For example, if the area weight factor λ is set too large, the solution will be heavily biased towards area uniformity, and will deviate from the goal of seeking the smallest number of pieces. To address this issue, we gradually decrease λ as the level ℓ increases. This will reduce the effect of area penalty over levels, encouraging the solver to focus more on finding the minimum solution as the level increases. Our current implementation sets $\lambda = \frac{1}{2}\,0.8^{\ell-1}$.

Avoiding split pieces. Another improvement we made to the objective function is to include a term that penalizes split pieces. A split piece is one that contains disconnected components. While these components transform together in the same way, they are not physically implementable. Thus we simply add a large penalty to such pieces to help eliminate them during best candidate selection. Note that we do not explicitly prevent them because there are cases where split pieces are temporarily unavoidable, such as during the first several levels of processing when the input figures themselves are disconnected (see Figure 7 (c)-(e)).

Extension to the Triangular Lattice Besides using a square lattice, our method can also be extended to other lattices, such as the right triangular lattice, which is constructed by splitting each grid on a square lattice to four isosceles triangles along the diagonals (see Figure 5). Using this lattice, we can represent input figures with both rectilinear edges as well as 45° angled edges. This makes the lattice representation more expressive and introduces more variations. Figure 5 shows two examples.

Extension to 3D Shape Dissection We can also extend our algorithm to dissecting 3D shapes that are represented onto a cubic grid. In this case, each grid cell has six neighbors, and the transformation of pieces considers 24 different 3D rotations. However, unlike 2D, a piece is not allowed to be mirrored (which is analogous to flipping in 2D), because in general mirroring can not be physically implemented in 3D. The

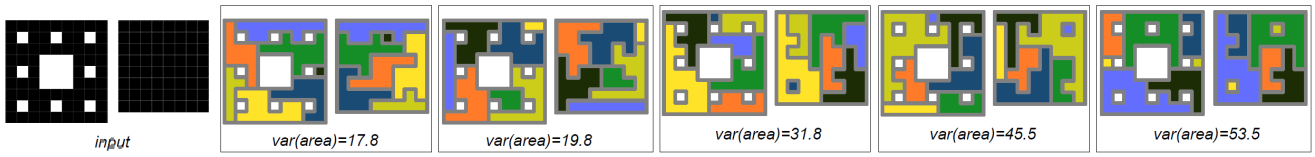


Figure 6: An example where our algorithm found multiple solutions with the same number of pieces. The input is a 9×9 Serpenski carpet and a 8×8 square. Five selected solutions are shown, each of which has 7 pieces. We calculate the area variance for each. Smaller variance corresponds to a more uniform/balanced size, which is more preferable.

area-based term is now modified to a volume-based term. A 3D dissection sample is shown in Figure 1. The video on the accompanying CD contains another example.

To physically realize the 3D puzzles, the solution must be free from interlocks, i.e. it must be possible to take the pieces apart from each other without breaking them. To do so, we introduce an interlock detector to filter out unsuitable solutions. Note that interlock detection for the general case is non-trivial. Here we apply a simple detection algorithm by attempting to move one piece at a time entirely out from the assembly. If at any step we cannot find such a piece, we report interlock. While this is only a sufficient condition, we found it to work well as our algorithm does not tend to produce pieces with complex structures.

3 Results

Optimality. To examine the optimality of our algorithm, we generated solutions for several representative dissection problems described in Frederickson's book entitled *Dissections: Plane and Fancy* [4]. These examples are demonstrated in Figure 7. As shown in the left column, all input figures are rectilinear polygons that can be represented exactly on a square lattice. The reference solutions from [4] are shown in the right column of the figure. The middle column shows our solutions. For all examples we obtain the same number of pieces as the dissections in [4]. Note also that for many of them our solution differs from those in [4] in terms of the shapes of the pieces.

2D dissections. Figures 1, 3, 4, 6, 7 all demonstrate 2D dissections using a square lattice. Figure 5 shows two-figure dissections computed using the triangular lattice.

Area-based term. In Figure 4 we have shown that enabling the area-based term leads to results where the sizes of the pieces are more balanced. In Figure 6 we show another example where our algorithm found multiple solutions at the same level. We typically select the best solution as the one that gives rise to the smallest area variance. But other criteria can also be used to define the best solution as well.

3D dissections. The bottom row of Figure 1 shows a 3D dissection that demonstrates $3^3 + 4^3 + 5^3 = 6^3$. Other 3D dissection examples can be found in the video.

4 Limitations and Future Work

One major limitation of our method is that due to discretization, many figures cannot be exactly represented using a discrete lattice grid. They have to be rasterized, resulting in approximate shapes. Therefore our method is not meant to substitute analytic approaches to many dissection problems. Nonetheless, for the purpose of creating puzzles, we have found the approximate shapes to be sufficient in many cases.

Another limitation is that the user is given little control over the algorithm (except for adjusting the area-based term), and so it is difficult to constrain the solution to have certain desirable properties. One example is the symmetry of the pieces, which is often desirable from an aesthetic point of view. Currently we do not consider such properties but believe it is possible to enforce user-defined constraints by modifying the objective function and including new terms.

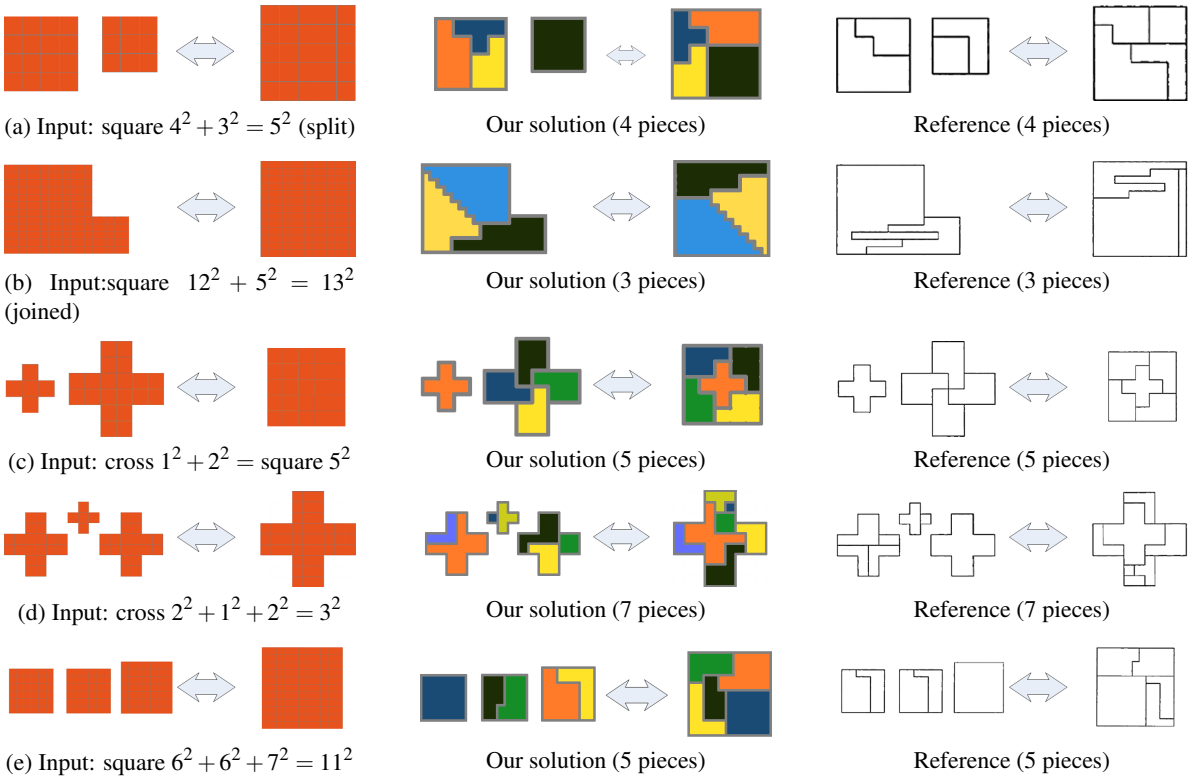


Figure 7: A comparison of our solutions with reference solutions shown in [4]. Some of these examples are visualizations of the Pythagorean triple numbers. The left column shows the input, the middle shows our solution, and the right shows solution in [4]. For all examples we achieve the same number of pieces as shown in [4].

References

[1] Jin Akiyama, Gisaku Nakamura, Akihiro Nozaki, Kenichi Ozawa, and Toshinori Sakai. The optimality of a certain purely recursive dissection for a sequentially n-divisible square. *Comput. Geom. Theory Appl.*, 24(1):27–39, 2003.

[2] F. Bolyai. *Tentamen juventutem. Typis Collegii Reformatorum per Josephum et Simeonem Kali*. Maros Vasarhelyini, 1832.

[3] M. J. Cohn. Economical triangle-square dissection. *Geometriae Dedicata*, 3(4):447–467, 1975.

[4] G. N. Frederickson. *Dissections: Plane and Fancy*. Cambridge University Press, 1997.

[5] P. Gerwien. Zerschneidung jeder beliebigen anzahl von gleichen geradlinigen figuren in dieselben stücke. *Journal für die reine und angewandte Mathematik (Crelle's Journal)*, 10:228–234, 1833.

[6] Evangelos Kranakis, Danny Krizanc, and Jorge Urrutia. Efficient regular polygon dissections. *Geometriae Dedicata*, 80(1):247–262, 2000.

[7] H. Lindgren. *Recreational Problems in Geometric Dissections and How to Solve Them*. Dover Publications, 1972.

[8] M. Lowry. Solution to question 269, [proposed] by Mr. W. Wallace. *Leybourn, T. (ed.) Mathematical Repository*, 3(1):44–46, 1814.

[9] Alpay Özdural. Mathematics and arts: Connections between theory and practice in the medieval islamic world. *Historia Mathematica*, 27(2):171–201, 2000.

[10] Douglas B. West. *Introduction to Graph Theory (2nd Edition)*. Prentice Hall, 2000.