Supplementary Material for *Inverse Tiling of 2D Finite Domains*

This supplementary material is composed of seven parts. The first part presents a proof of NP-hardness of our inverse tiling problem. The second part analyses the search space of our prototile set construction method. The third part presents the implementation details of the prototile set construction method. The fourth part provides additional inverse tiling results. The fifth part provides details about the experiment that evaluates scalability of our inverse tiling approach. The sixth part provides details about the experiment for evaluating K minimization of our inverse tiling approach. The last part provides details about the user study for the puzzle application.

1 Proof of NP-hardness of Inverse Tiling Problem

We present a proof for NP-hardness of the inverse tiling problem formally defined in Section 3 of the paper. To this end, we denote the inverse tiling problem as INVERSE_TILING and the decision version of INVERSE_TILING problem as INVERSE_TILING_DECISION. Furthermore, we choose a decision version of the forward tiling problem denoted as FORWARD_TILING_DECISION, which is proved to be a NP-complete problem [Horiyama et al. 2017]. To prove NP-hardness of INVERSE_TILING problem, we first prove that INVERSE_TILING_DECISION problem is NP-complete (see Section 1.1), by polynomially reducing the FORWARD_TILING_DECISION problem to it. Next, we prove NP-hardness of INVERSE_TILING problem by showing that it is an optimization problem that is harder than INVERSE_TILING_DECISION problem (see Section 1.2).

1.1 NP-completeness of INVERSE_TILING_DECISION

The FORWARD_TILING_DECISION problem can be defined as:

$$\begin{gathered}\text{FORWARD\_TILING\_DECISION} = \{(D, N, P) \mid \\ \text{there exists } N \text{ tiles that exactly cover domain } D, \\ \text{where each tile is an instance of a prototile in } P\}.\end{gathered}$$

where D is the given finite domain that needs to be tiled, N is the given number of tiles, P is the prescribed prototile set. The FORWARD_TILING_DECISION problem has been proved to be NP-complete [Horiyama et al. 2017]. Next, we define INVERSE_TILING_DECISION problem as:

$$\begin{gathered}\text{INVERSE\_TILING\_DECISION} = \{(D, N, K_{\max}, C) \mid \\ \text{there exists } N \text{ tiles that exactly cover domain } D, \\ \text{such that the number of prototiles } K \text{ is smaller than } K_{\max}, \\ \text{and each tile satisfies prototile size/shape constraints in C}\}.\end{gathered}$$

where D is the given finite domain that needs to be tiled, N is the given number of tiles, $K_{\max}$ is the maximally allowed number of prototiles, and C is a set of constraints on prototile size/shape, e.g., constraints on the number of grid cells occupied by each prototile, bounding box size, shape connectivity and convexity, etc.

Polynomial-time Reduction from FORWARD_TILING_DECISION to INVERSE_TILING_DECISION. For each instance of the FORWARD_TILING_DECISION problem with input (D', N', P'), we can construct a corresponding instance of the INVERSE_TILING_DECISION problem with input $(D, N, K_{\max}, C)$, where the input domain $D = D'$ and the number of tiles $N = N'$ remain unchanged. The upper bound on the number of prototiles, $K_{\max}$, can be set to $N + 1$, as the FORWARD_TILING_DECISION problem does not impose constraints on the number of prototiles used. The constraint set C in INVERSE_TILING_DECISION can be defined to require that all prototiles must exactly match one of the tile shapes in the given set P'. It is straightforward to construct D, N, $K_{\max}$, and C from the input (D', N', P') in polynomial time. In this way, any valid solution to the INVERSE_TILING_DECISION problem that satisfies these constraints corresponds to a valid solution to the FORWARD_TILING_DECISION problem, establishing a polynomial-time reduction.

1.2 NP-hardness of INVERSE_TILING

The objective of our INVERSE_TILING problem is to minimize the number (K) of prototiles needed to exactly cover the domain D, subject to the constraints C on prototile size/shape, and the number of tiles N. Our INVERSE_TILING problem is an optimization version of INVERSE_TILING_DECISION problem since it aims to minimize the number (K) of prototiles instead of determining if the number (K) of prototiles is smaller than a predefined number $K_{\max}$. According to [Garey and Johnson 1979], an optimization problem is NP-hard if its decision version is NP-complete. Hence, our INVERSE_TILING problem is NP-hard.

2 Analysis of Our Approach

In this part, we first provide derivation on the search space size of our prototile set construction method in our approach and a trivial method based on partitioning the input domain into N pieces (i.e., tiles). Next, we experimentally show that the search space of our method is significantly smaller than that of the partitioning-based method.

Search space of our prototile set construction method. The search space of our prototile set construction method is $O(\binom{M}{N}) \cdot O(N \cdot \prod_{k=1}^{C_{\max}-1}(2k+2))$, where $C_{\max}$ is the user-specified maximal allowable number of grid cells of each tile. In our method, we first select N seeds from M grid cells, which corresponds to a search space of $O(\binom{M}{N})$. After that, we enlarge these N tiles by iteratively including their neighboring unoccupied grid cells, starting with initial tiles that occupy only a single grid cell. For each tile with k grid cells, there exist at most $(2k + 2)$ unoccupied neighboring grid cells [Redelmeier 1981] that can be included to enlarge the tile. Therefore, when $C_{\max}$ is given, the total number of tile enlargement variations is bounded by $\prod_{k=1}^{C_{\max}-1}(2k+2)$. For N tiles, there are at most $(N \cdot \prod_{k=1}^{C_{\max}-1}(2k+2))$ possible enlargement options. By default, $C_{\max}$ is set to $\lceil \frac{M}{N} \rceil + 1$. Hence, the search space of our method is $O(\binom{M}{N})) \cdot O(N \cdot \prod_{k=1}^{\lceil \frac{M}{N} \rceil}(2k+2))$.

Search space of the partitioning-based method. The search space of the partitioning-based method is $O(N^M)$. This is because there are N^M possible ways to partition a domain with M grid cells into N pieces (i.e., tiles) since each of the M grid cells can be assigned to any of the N tiles.

Comparison with the partitioning-based method. We compare the search space of our prototile set construction method with that of the partitioning-based method under two different experimental settings.

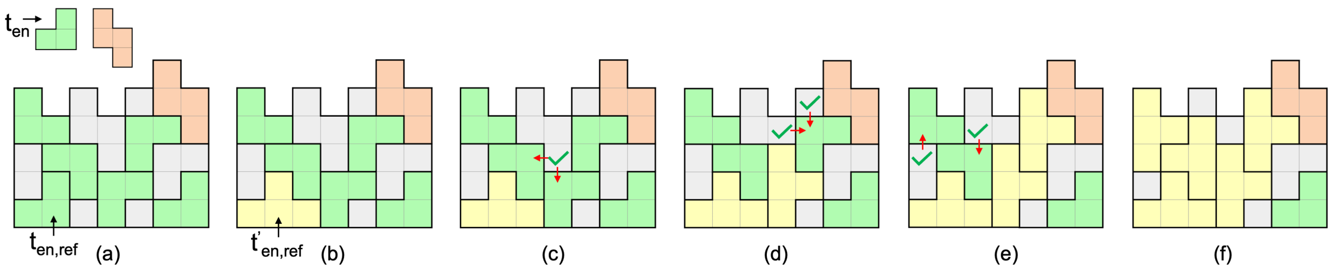
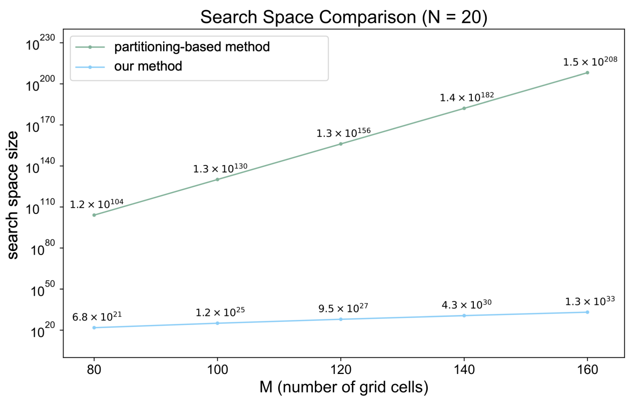


Figure 1: *Enlarging part of the tiles congruently for a tiling state. We first (a) select prototile t_{en} (in green color) from existing prototiles $\{t_k\}$, and then select prototile instance $t_{en,ref}$ as the reference tile. Next, we (b) enlarge the reference tile $t_{en,ref}$ to form the target tile $t'_{en,ref}$ (in yellow color) by including an adjacent uncovered grid cell. We further (c-f) enlarge each of the remaining instances of the prototile t_{en}, such that it has the same shape as the target tile $t'_{en,ref}$.*



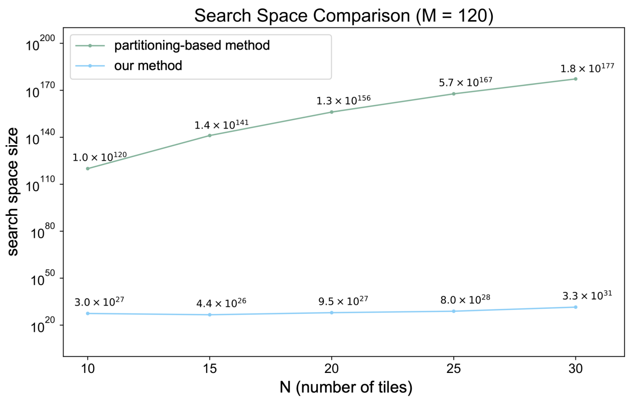


Figure 2: *Comparing the search space size between our prototile set construction method and a partitioning-based method: (top) search space size with respect to M while fixing $N = 20$; (bottom) search space size with respect to N while fixing $M = 120$. Note that the vertical axis (i.e., search space size) is in logarithmic scale.*

First, we fix the number of tiles $N = 20$ and increase the number of grid cells M to analyze the growth of the search space size with respect to M. Next, we fix the number of grid cells $M = 120$ and increase the number of tiles N to analyze the growth of the search space size with respect to N. Figure 2 shows that as M (N) increases, the search space of our method is significantly smaller and grows slower than that of the partitioning-based method. One possible reason to explain this is that the tile iterative enlargement strategy in our method naturally ensures that each tile satisfies the prototile size/shape constraints, which cannot be achieved by the partitioning-based method. This experiment demonstrates the effectiveness of our prototile set construction method to reduce the search space of the inverse tiling problem.

3 Implementation Details for Enlarging Tiles

In this part, we provide implementation details of the approach for generating candidates of the next tiling state in Section 5.1 of the paper. This approach consists of four steps to enlarge tiles.

(i) *Selecting a prototile t_{en}.* We select a prototile, say t_{en}, from the current set of prototiles $\{t_k\}$ for enlarging its tiles (prototile instances) based on three criteria (see Figure 1(a)):

- *Prototile size.* First, we prioritize enlarging small prototiles to meet the minimum allowed number ($C_{\min}$) of grid cells. If a prototile's size equals $C_{\max}$, it should not be selected.
- *Number of prototile instances.* Second, we prioritize enlarging a prototile with a large number of instances, aiming to cover more grid cells in the input domain D.
- *Enlargeability.* Third, we prioritize enlarging a prototile with a low enlargeability value, as some of its instances may not be enlargeable after a few tile enlargement operations.

We normalize each term to the range $[0, 1]$, with a higher value indicating a higher chance of choosing the prototile. Empirically, the weights for the three terms are set as 0.6, 0.2, and 0.2, respectively, to balance the impact of the terms.

(ii) *Selecting a prototile instance $t_{en,\ ref}$.* After choosing prototile t_{en}, we further pick one of its instances, say $t_{\text{en, ref}}$, to form a reference tile; see Figure 1(a). In detail, the prototile instance $t_{\text{en, ref}}$ will be enlarged by including an adjacent uncovered grid cell, and the enlarged shape will serve as a reference to guide the enlargement of the remaining prototile instances in $\{t_{\text{en, }j}\}$. We prioritize selecting $t_{\text{en, ref}}$ as a prototile instance with a low enlargeability value for two reasons. First, such a prototile instance may not be enlargeable if we choose to enlarge some other prototile instances first. Second, a prototile instance with a low enlargeability value has less flexibility to be congruently enlarged, so enlarging it first increases the chance of congruently enlarging more prototile instances.

(iii) *Determining the target tile $t'_{en,\ ref}$.* After choosing prototile instance $t_{\text{en, ref}}$, we next have to choose which of its adjacent uncovered grid cells to take for enlarging $t_{\text{en, ref}}$ to form a new tile denoted as $t'_{\text{en, ref}}$; see Figure 1(b). We determine the shape of target tile $t'_{\text{en, ref}}$ using the following approach, guided by the congruent tiles requirement:

- *Match an existing prototile.* We first identify uncovered grid cells adjacent to tile $t_{\text{en, ref}}$. Then, we try to extend $t_{\text{en, ref}}$ to include each of the uncovered grid cells and check if the enlarged tile has the same shape as one of the existing prototiles. In case the reference tile $t_{\text{en, ref}}$ can be enlarged

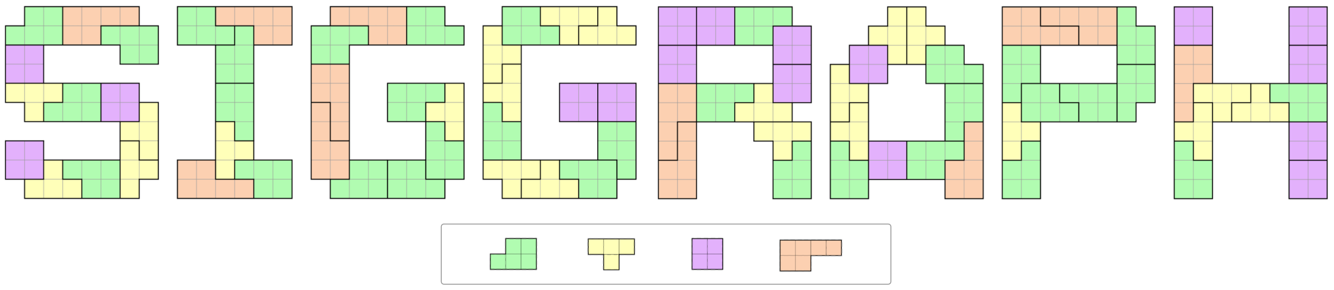


Figure 3: *A tiling result produced by our approach on an input domain* SIGGRAPH *with 8 disjoint letters.*

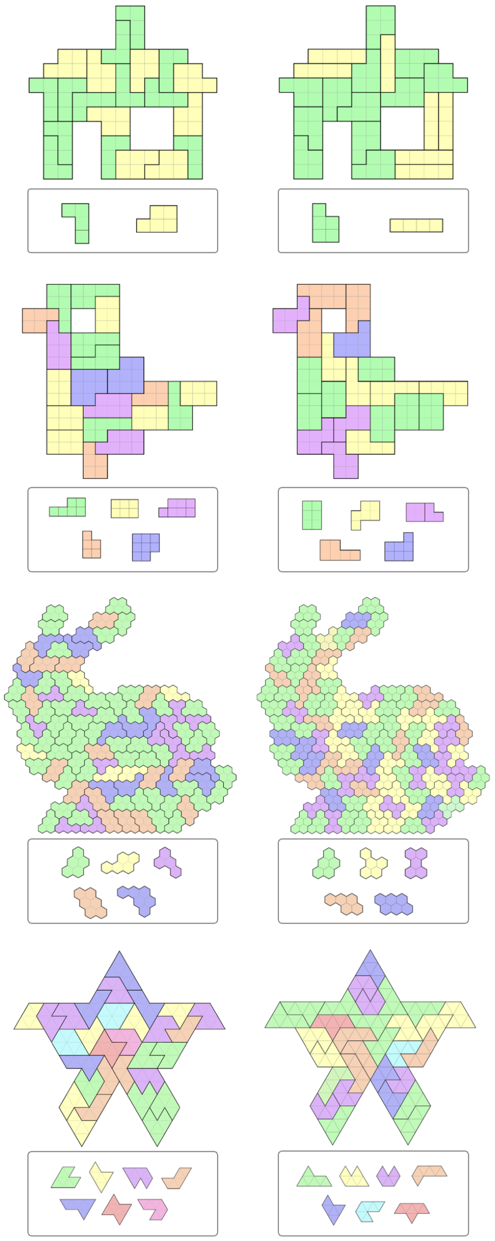


Figure 4: *Our approach allows generating different tiling results with the same K for the same input domain. For each input domain, we show two different tiling results with the same K.*

in different ways to match multiple existing prototiles, we choose the existing prototile as the target tile $t'_{\text{en, ref}}$, for which the associated uncovered grid cell has a low blockability value. By doing so, we have a better chance of reducing the number of prototiles by one.

- *Match part of an existing prototile.* In case the reference tile $t_{\text{en, ref}}$ cannot be enlarged to match any existing prototile, we choose the target tile $t'_{\text{en, ref}}$ whose shape matches part of an existing prototile using the same method as above. By doing so, the reference tile $t_{\text{en, ref}}$ will have a chance of matching an existing prototile after a few more iterations of tile enlargement.

- *Choose an uncovered grid cell.* In case the reference tile $t_{\text{en, ref}}$ cannot be enlarged to match any existing prototile or part of any existing prototile, we prioritize to choose an uncovered grid cell adjacent to the reference tile $t_{\text{en, ref}}$ with a low blockability value, and assign the cell to $t_{\text{en, ref}}$ to form a target tile $t'_{\text{en, ref}}$; see Figure 1(b). If the target tile $t'_{\text{en, ref}}$ does not satisfy the prototile shape constraints (i.e., the bounding box size $W_{\text{bbox}} \times H_{\text{bbox}}$ and the convexity threshold τ_{conv}), we discard the assignment and choose another uncovered grid cell to enlarge the reference tile $t_{\text{en, ref}}$.

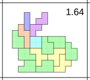
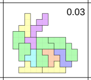
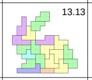
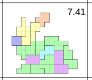
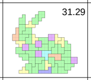
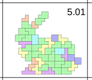
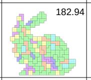
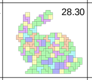
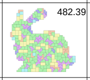
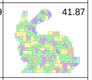
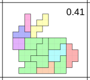
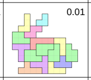
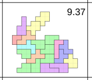
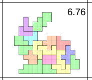
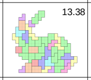
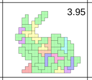
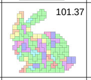
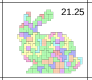
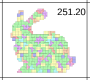
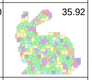
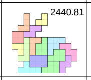
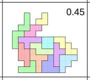
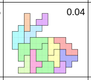
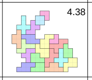
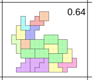
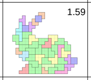
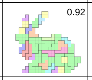
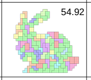
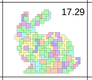
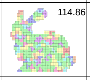
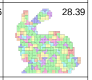
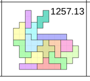
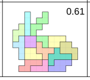
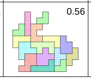
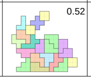
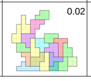
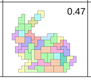
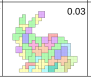
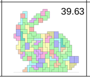
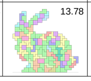
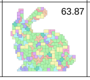
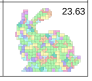
(iv) *Enlarging the prototile instances $\{t_{en,\ j}\}$.* Once the target tile shape $t'_{\text{en,ref}}$ is determined, we aim to enlarge each of the remaining prototile instances in $\{t_{\text{en, }j}\}$ to make it have exactly the same shape as $t'_{\text{en, ref}}$, to meet the congruent tiles requirement. This can be classified into three cases. In the first case, one uncovered grid cell can be assigned to multiple adjacent prototile instances to make each of them match $t'_{\text{en, ref}}$. In this case, we assign the cell to the adjacent prototile instance with a small enlargeability value; see Figure 1(c→d). In the second case, a prototile instance has multiple choices of including an adjacent uncovered grid cell to match $t'_{\text{en, ref}}$. In this case, we assign the adjacent uncovered cell with a small blockability value to the prototile instance; see Figure 1(d→e). In the last case, one uncovered grid cell can be assigned to a single adjacent prototile instance to make it match $t'_{\text{en, ref}}$. In this case, we directly make the assignment; see Figure 1(e→f).

4 Additional Inverse Tiling Results

Inverse tiling results on disconnected domains. Our inverse tiling approach is able to produce tiling results on disconnected domains. Figure 3 shows that our approach is able to produce a tiling result from an input domain with disjoint regions (i.e., eight alphabet letters).

Inverse tiling results for the same K. Thanks to the randomness in our prototile construction process, our inverse tiling approach allows

Table 1: *K-hedral tiling results of* Bunny *with different resolutions generated in the experiment to evaluate scalability of our approach by comparing it with two baseline approaches. The computational time of each result is in minutes.*

	$[C_{min}, C_{max}] = [3, 6]$														
	M = 68, N = 15			M = 107, N = 20			M = 216, N = 45			M = 484, N = 100			M = 1000, N = 200		
	baseline #1	baseline #2	our approach	baseline #1	baseline #2	our approach	baseline #1	baseline #2	our approach	baseline #1	baseline #2	our approach	baseline #1	baseline #2	our approach
K=6	/	1.64 	0.03 	/	13.13 	7.41 	/	31.29 	5.01 	/	182.94 	28.30 	/	482.39 	41.87 
K=8	/	0.41 	0.01 	/	9.37 	6.76 	/	13.38 	3.95 	/	101.37 	21.25 	/	251.20 	35.92 
K=10	2440.81 	0.45 	0.04 	/	4.38 	0.64 	/	1.59 	0.92 	/	54.92 	17.29 	/	114.86 	28.39 
K=12	1257.13 	0.61 	0.56 	/	0.52 	0.02 	/	0.47 	0.03 	/	39.63 	13.78 	/	63.87 	23.63 

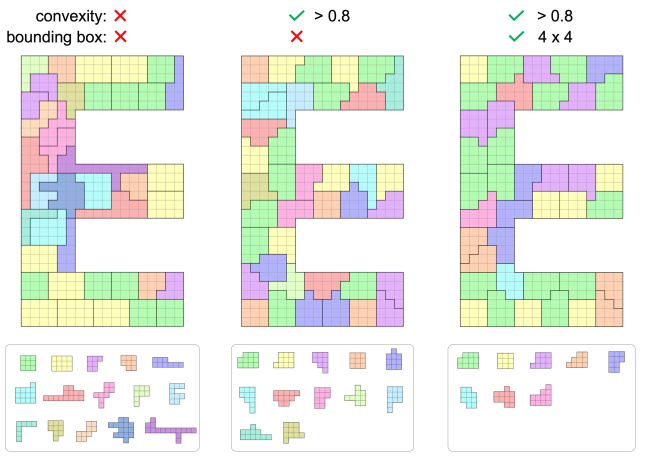


Figure 5: *Our approach allows generating tiling results with constraints on the prototile shape, which are: (i) minimum convexity 0.8; and (ii) maximum bounding box 4×4. Tiling results generated (left) without the two constraints, (middle) with the first constraint, and (right) with both constraints.*

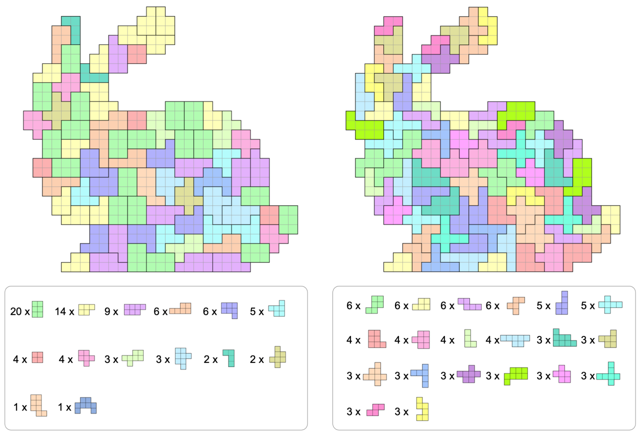


Figure 6: *Our approach allows controlling balance of the numbers of prototile instances $\{n_k\}$: (left) less balanced $\{n_k\}$ where $n_k \in [1, 20]$, and (right) more balanced $\{n_k\}$ where $n_k \in [3, 6]$.*

generating different tiling results with the same K for the same input domain; see Figure 4 for examples. In practice, we can generate multiple tiling results with the same K for the same input domain and one can choose a tiling result according to his/her preference.

Inverse tiling results with constraints on the prototile shapes. Figure 5 shows how constraints on the prototile shapes affect the tiling result. In particular, the prototile convexity constraint (convexity > 0.8) avoids structurally weak tiles for fabrication; compare Figure 5 (left) and (middle). The prototile bounding box size constraint (4×4) further makes each tile more compact; compare Figure 5 (middle) and (right).

Inverse tiling results for a more balanced prototile set. Figure 6 shows that our approach allows controlling balance of the numbers of prototile instances. This is achieved by prioritizing choosing the next tiling state candidate where the numbers of prototile instances, $\{n_k\}$, are more balanced in the computational framework.

5 Evaluating Scalability of Our Approach

In Section 6 of the paper, we conducted an experiment to evaluate the scalability of our inverse tiling approach by comparing it with two baseline approaches. The first baseline is a randomized search of all possible partitions of the input domain to find a K-hedral tiling result. The second baseline is a randomized search within our computational framework, where the strategies (i.e., blockability and enlargeability) are disabled. In this part, we provide more details about these two comparisons.

Comparison with baseline #1. Table 1 shows that baseline #1 is only able to generate a tiling result when both the number (M) of grid cells in the input domain and the number (N) of tiles are small (i.e., when $M = 68$ and $N = 15$). In contrast, our approach is able to generate a tiling result even when M and N become larger (e.g., when $M = 1000$ and $N = 200$). The better scalability performance of our approach can be explained by the smaller search space enabled by our computational framework; see again Figure 2.

Comparison with baseline #2. Although the search space has been significantly reduced by using our computational framework, it remains large, making a randomized search within the computational framework (baseline #2) inefficient for finding a tiling result on a large input domain; e.g., baseline #2 took 8.04 hours to generate a $K = 6$ tiling result for a Bunny with 1000 grid cells and 200 tiles. Hence, we introduce two strategies to guide the search process: blockability and enlargeability; see Section 5.1 in the paper. By incorporating the two strategies, the computation time of our approach is significantly reduced compared to baseline #2 that does not use them. Table 1 shows that our approach is more efficient than baseline #2 for generating a K-hedral tilting result for input

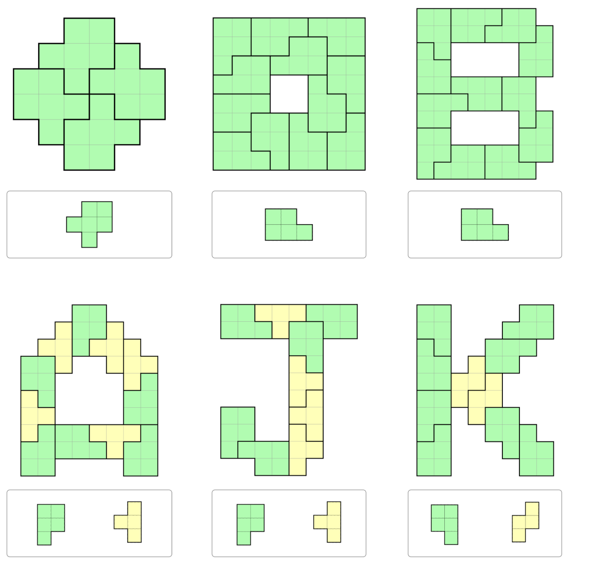


Figure 7: *Tiling results for six different input domains produced by our inverse tiling approach. For each of them, the minimized K is the same as the ground truth.*

domains with difference sizes (Ms) and different given number of tiles (Ns). This improvement demonstrates the effectiveness of our strategies in guiding the search of desired inverse tiling results within our computational framework.

6 Evaluating K Minimization of Our Approach

In Section 6 of the paper, we validated the proximity between the minimized K computed by our approach and the actual minimal K by comparing our approach with the ground truth for tiling the same 2D finite domain. Here we use dancing links (DLX) [Knuth 2000] to find the minimal K. DLX is a technique for efficiently implementing backtracking algorithm, enabling to efficiently enumerate all the possible tiling results for a given set of allowable tile shape. In this experiment, we use tetrominoes (4 squares) and/or pentominoes (5 squares) to tile the COIN, HOUSE and TEAPOT, since tetrominoes and pentominoes are the most commonly used tiles in polyomino tiling problems. There are 5 free tetrominoes without holes and 12 free pentominoes without holes [Redelmeier 1981], so we have 17 possible tile shapes in total. Hence, there are $\binom{17}{1} = 17$ possible sets with a single shape, $\binom{17}{2} = 136$ possible sets with two shapes, and $\binom{17}{3} = 680$ possible sets with three shapes. To find the minimal K, we iterate through all sets of allowable tile shapes, starting with sets with a single shape ($K = 1$), then moving to sets with two shapes ($K = 2$), sets with three shapes ($K = 3$), and so on. For each set of allowable tile shapes, we employ DLX to identify if there exists a feasible tiling result. As we aim to find a minimal K rather than enumerating all the tiling results, we stop the iteration once a feasible tiling result is found and record the number of tile shapes in the current set as the ground truth (minimal K). The minimal K for the COIN, HOUSE and TEAPOT is found as 1, 2 and 3, respectively.

Next, we run our inverse tiling algorithm to tile the three input domains, where the tile size constraint is set as $[C_{\min}, C_{\max}] = [4, 5]$. Specifically, the number of tiles N falls within the range of $[\lceil 48/5 \rceil = 10, \lfloor 48/4 \rfloor = 12]$ for the COIN, $[\lceil 88/5 \rceil = 18, \lfloor 88/4 \rfloor = 22]$ for the HOUSE and $[\lceil 213/4 \rceil = 54, \lfloor 213/5 \rfloor = 42]$ for the TEAPOT, based on the tile size constraint and the number of grid cells in the input domain (48 grid cells in the COIN, 88 grid cells in the HOUSE and 213 grid cells in the TEAPOT). We parallelly run our approach for each possible N to find the tiling result with the smallest K for at most 12 hours. As a result, our approach achieved the minimal K when tiling the COIN (0.35 minutes) and HOUSE (15.35 minutes) while generating a tiling result with $K = 4$ for the TEAPOT (331.86 miuntes), which is close to the ground truth ($K = 3$). This experimental result shows that our inverse tiling approach is able to efficiently find a minimized K that is equal or close to the ground truth.

In addition to the three results shown in Figure 11 of the paper, we also use the same experiment settings to evaluate the minimization of K for another six input domains; see Figure 7. For each of the six input domains, our approach is able to efficiently find a tiling result with the minimal K (ground truth) within 1 minute.

Table 2: *Statistics of our user study. The 2nd column to 6th column present the time of solving each puzzle in minutes. Specifically, slash indicates the user cannot solve the given puzzle in 60 minutes. The last row provides the average time ($\bar{t}$) for solving each puzzle in minutes.*

	Bird N=20, K=5	Bird N=20, K=10	Bird N=10, K=5	Heart (squ) N=20, K=4	Heart (hex) N=20, K=4
User 1	10.55	/	21.34	36.41	/
User 2	13.42	26.47	6.5	14.95	39.71
User 3	18.94	37.54	3.41	21.96	17.56
User 4	22.84	/	17.46	57.51	/
User 5	19.62	/	5.61	17.35	45.78
User 6	22.45	41.28	10.54	27.46	52.68
$\bar{t}$	17.97	35.10	10.81	29.27	38.93

7 User Study of Puzzle Application

In Section 6 of the paper, we conducted a user study to evaluate the 2D assembly puzzles designed by our inverse tiling approach. In our user study, we recruited 6 participants who had no prior knowledge of our 2D assembly puzzles to study factors influencing the level of difficulty of our designed puzzles. For each participant, we briefly introduced the concept of K-hedral tiling of 2D finite domains and then informed him/her the task of assembling a set of puzzle pieces to form a 2D target shape within 60 minutes. During the assembly of puzzle pieces, users were allowed to rotate, translate and flip the pieces (tiles) to find a feasible assembly result. We presented each participant with 5 assembly puzzles in the following order: BIRD with $N = 10, K = 5$, BIRD with $N = 20, K = 10$, BIRD with $N = 20, K = 5$, HEART (square grid) with $N = 20, K = 4$, and HEART (hexagon grid) with $N = 20, K = 4$. After each participant attempted to solve the puzzle (i.e., assemble the puzzle pieces to form each target shape), we recorded whether they successfully solved the puzzle within the regulated time period and the time taken to solve the puzzle if successful.

Result. The statistics of our user study is presented in Table 2. The slash in Table 2 indicates the user could not solve the given puzzle within 60 minutes. Most of puzzles were solved between 10 to 50 minutes, demonstrating the varying levels of difficulty across the puzzles. We observed that a puzzle's level of difficulty is related to the number of pieces (N), the number of distinct pieces (K), and grid types. This is because a large N, a large K, and a complex grid type (i.e. hexagon grid) increases the number of possible combinations in which two pieces can be put together with edge-to-edge contact, making the puzzles harder to solve for players. For example, participants took longer to solve BIRD with $N = 20, K = 5$ (17.97

minutes on average) compared to BIRD with $N = 10, K = 5$ (10.81 minutes on average), confirming the relation between the number (N) of tiles and the puzzle's level of difficulty. Moreover, three of our participants failed to solve BIRD with $N = 20, K = 10$, demonstrating that the puzzle's level of difficulty increases for a large K. In terms of level of difficulty caused by different grid types, we observe that two of our participants failed to solve HEART (hexagon grid) with $N = 20, K = 4$ within 60 minutes while all the participants successfully solved HEART (square grid) with $N = 20, K = 4$, validating the puzzle's level of difficulty increases for a grid pattern where the grid cells have more edges. Thanks to our inverse tiling approach, users are able to design a puzzle whose level of difficulty matches his/her ability by specifying the number (N) of tiles and the grid type and choosing tiling results with a desired number (K) of prototiles.

References

GAREY, M. R., AND JOHNSON, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company.

HORIYAMA, T., ITO, T., NAKATSUKA, K., SUZUKI, A., AND UEHARA, R. 2017. Complexity of tiling a polygon with trominoes or bars. *Discrete and Computational Geometry 58*, 3, 686–704.

KNUTH, D. E. 2000. Dancing links. In *Millennial Perspectives in Computer Science*, 187–214.

REDELMEIER, D. H. 1981. Counting polyominoes: Yet another attack. *Discrete Mathematics 36*, 2, 191–203.