Solving Tiling Enumeration Problems by Tensor Network Contractions

Kai Liang

Submitted: Aug 16, 2025; Accepted: Oct 20, 2026; Published: Nov 6, 2026
© The author. Released under the CC BY license (International 4.0).

Abstract

This paper presents an algorithm for contracting two-dimensional tensor networks on a square lattice; and we combine it with solving congruence equations to compute the exact enumeration (including weighted enumeration) of Wang tilings. Based on this, the paper demonstrates how to transform other tiling enumeration problems (such as those of polyominoes) into Wang tiling enumeration problems, thereby solving them using this algorithm.

Our algorithm extends the sequence length records for dozens of sequences defined by polyomino tiling enumeration on chessboards on the OEIS website, covering numerous of different polyomino sets, including I-polyominoes, tetrominoes, pentominoes, etc. This demonstrates the high efficiency and strong universality of the algorithm for solving exact tiling enumeration problems.

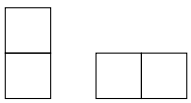
In addition, the theory and techniques used in the algorithm establish a bridge between tensor network contractions and tiling enumeration, where the former provides a theoretical foundation for solving problems in the latter, while the latter offers an intuitive combinatorial interpretation of the former.

Mathematics Subject Classifications: 05B45, 05A15, 05B50, 15A69

1 Introduction

Tiling enumeration is a classic problem in combinatorics [1, 2, 3].

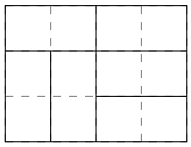
Taking the most classic *domino tiling* ([4, 5, 6, 7, 8]) as an example: given a *rectangle chessboard* of size $l \times h$ (i.e., with width l and height h), the task is to calculate the number of all possible ways to cover the entire chessboard by tiling *dominoes* (as shown below, 1×2 tiles that can be rotated), distinguishing between distinct ways that are symmetric.



School of Mathematics and Statistics, Xidian University, Xi'an, China (24071213197@stu.xidian.edu.cn).

 https://doi.org/10.37236/14510

For example, the following is one of the tilings using dominoes in a 4×3 chessboard:



When the shape of the chessboard is $l \times h$, the number of different ways of tiling (also reffered to as *tiling enumeration*) can be calculated using the following formula [4, 5]:

$$\left(\prod_{i=1}^{l}\prod_{j=1}^{h}\left(4\cos^2\frac{i\pi}{l+1}+4\cos^2\frac{j\pi}{h+1}\right)\right)^{\frac{1}{4}}.$$

However, most other tiling enumeration problems do not have specific formulas applicable to the general case like this, with at most formulas for special cases where the width l is fixed and small [9, 10]. In most cases, such enumeration problems requires complex computation, and the time and space complexity of the algorithms are both exponential [11, 12].

The algorithm presented in this paper is designed for the exact enumeration of Wang tilings on various shapes of chessboards, and can be generalized to other tiling problems (such as polyomino tilings). Compared to previous algorithms, it effectively saves space and improves efficiency. The website OEIS (The On-Line Encyclopedia of Integer Sequences) records hundreds of sequences defined by similar tiling enumeration in chessboards, Our algorithm has extended the known sequence lengths for dozens of these cases beyond previous record.

Section 2 of this paper introduces the preliminary knowledge involved in the algorithm, including tiling enumeration and tensor network contractions.

Section 3 explains the specific principles of the algorithm for the exact enumeration of Wang tilings and analyzes its related parameters.

Section 4 discusses how to transform domino tilings into Wang tilings, thereby enabling exact enumeration using this algorithm.

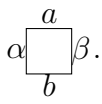
Section 5 provides a summary of some further generalizations of the algorithm, and enumerates several unresolved open problems.

2 Preliminary

2.1 Tiling enumeration

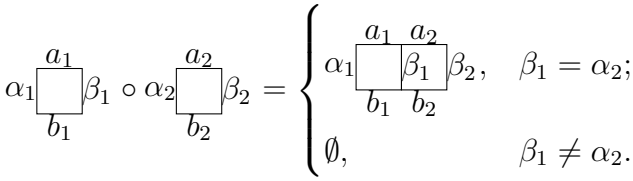
2.1.1 Wang tiling

Wang tiles ([13]) are unit square tiles, each with a character on its four edges, such as



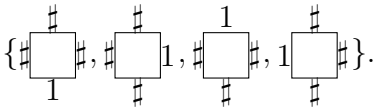
.

In general, the characters on the left and right edges of a tile are taken from a character set, which we call the *horizontal character set*; while the characters on the top and bottom edges are taken from another character set, which we call the *vertical character set*. Wang tiles can be joined edge-to-edge, provided that the characters on adjacent edges are identical. For example, the horizontal concatenation of two Wang tiles is

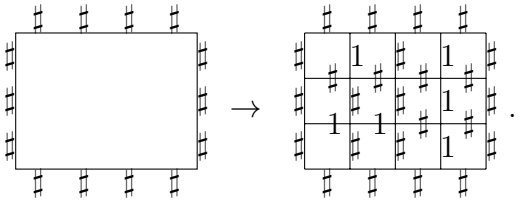


A *Wang tiling* requires covering a specific region with Wang tiles that match edge-to-edge. Under the standard rule, Wang tiles cannot be rotated or flipped, and each position on the edge of the given region is assigned a fixed character (called the *boundary character*, usually denoted as ♯), requiring the tiles on the edge to match these characters. Various variants of Wang tilings in finite regions can be reduced to this standard form.

For instance, we select the Wang tile set as



Then, one of the tilings in a 4×3 chessboard is



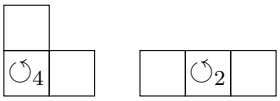
When the tiling region is the entire plane (or an entire quadrant), the problem of whether a tiling exists to cover the entire plane (i.e., the tiling existence problem) is undecidable [13]. In fact, a significant portion of problems regarding Wang tilings on the plane are undecidable [14].

When the tiling region is finite, the problem becomes decidable (since at least all possible tilings can be enumerated), but it remains computationally complex. For example, the existence problem for Wang tilings on a $l \times h$ chessboard (whether or not boundary characters are restricted or tile rotation is allowed) is always NP, while the enumeration problem is always #P [11, 12]. Therefore, only approximate enumeration algorithms can be provided for large-scale general Wang tilings.

2.1.2 Polyomino tiling

n-ominoes are the natural generalization of dominoes, referring to the tiles composed of n unit squares joined edge-to-edge [15, 16].

3-ominoes are also called *trominoes*, with 2 shapes (as shown below): the *right tromino* and the *straight tromino* [17].



The notation $\circlearrowleft_n$ on a tile indicates that the polyomino can be rotated into n different orientations.

Similarly, $4, 5, \ldots$-ominoes have special names: *tetrominoes*, *pentominoes*, ...

[18] provides a method for simulating Wang tilings using polyomino tilings, proving that the problem of whether general domino tilings can cover the entire plane is undecidable, and that the existence and enumeration problems for tilings on general chessboards are NP and #P, respectively.

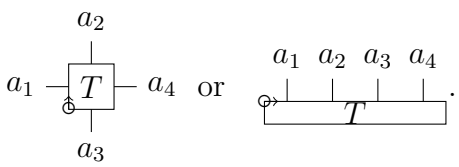
Currently, for problems of this type, backtracking algorithms or heuristic searches are commonly employed to exhaustively enumerate all possible tilings [19, 20, 21, 22]. However, due to the exponential growth of computational cost with increasing problem size, such computations may require supercomputers or distributed computing systems, and in some cases, may even become entirely infeasible.

The algorithm presented in this paper is an optimized *dynamic programming* approach, which does not require the exhaustive enumeration of all possible tilings. We store the states by using multi-dimensional tensors (referred to as *state tensors*), and state transitions are implemented through specific tensor operations, which can significantly save space and improving efficiency. Furthermore, we employ modular enumeration and solving congruence equations instead of direct enumeration to avoid data overflow and further improve computational efficiency.

2.2 Tensor contractions

This paper uses $\mathbb{K}$ to denote a general commutative (semi-)ring. a_i denotes a free index from the index set Σ_i (also referred to as the *vertical alphabet*), and σ_i denotes its size; α_i denotes a free index from the index set Θ_i (also referred to as the *horizontal alphabet*), and θ_i denotes its size.

A $\mathbb{K}$-*tensor* refers to a tensor whose entries are elements of a (semi-)ring $\mathbb{K}$. Since tensor contraction involves only addition and multiplication, such a generalization is natural. This paper uses *string diagrams* to represent tensors and their operations [23, 24]. Specifically, we use closed shapes to represent tensors and pins to represent tensor indices. For example, a tensor $T_{a_1a_2a_3a_4}$ with 4 indices is represented as



The order of the indices (from left to right) is determined by starting at the hollow points (referred to as *initial points*) on the string diagram and following the direction of the arrows.

Computers typically store the entries of a tensor in a fixed linear order. In programming languages such as C and Python, multi-dimensional arrays are stored in *row-major* order, i.e. the *last* index varies fastest and the *first* index varies slowest. For a tensor $T_{a_1a_2a_3a_4}$, the entries are stored in the order

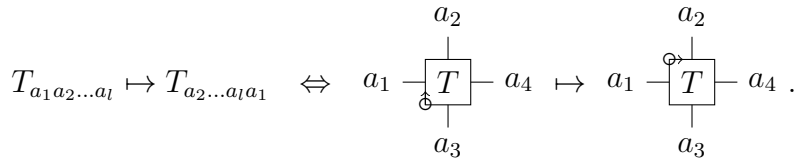
$$T_{0000}, T_{0001}, T_{0002}, \ldots, T_{0010}, T_{0011}, \ldots, T_{0020}, \ldots, T_{0100}, \ldots, T_{1000}, \ldots,$$

and are accessed in code as T[a_1][a_2][a_3][a_4]. Thus the memory order of the index sequence is exactly the *reverse* of the order in which we write the indices: the first index a_1 varies slowest while the last one a_4 varies fastest. The shape parameter of the tensor (containing the sizes of the index sets for each index) is also reversed in this way.

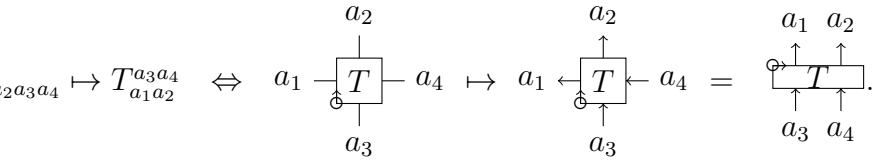
Additionally, we define the *rotation of indices* as

$$T_{a_1a_2\ldots a_l} \mapsto T_{a_2\ldots a_la_1},$$

which means rotating the first index to the last position. If we specify the order of indices by assigning an initial point and an initial direction on the string diagram of the tensor, such a rotation can be achieved by moving the initial point forward and jumping over an index, like

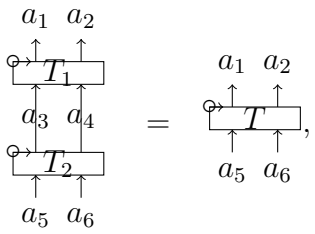
$T_{a_1a_2\ldots a_l} \mapsto T_{a_2\ldots a_la_1} \quad \Leftrightarrow$  .

By dividing the indices into row indices (the subscripts of the tensor, represented by outward arrows in the string diagram) and column indices (the superscripts, represented by inward arrows), a tensor can be flattened into a matrix, like

$T_{a_1a_2a_3a_4} \mapsto T^{a_3a_4}_{a_1a_2} \quad \Leftrightarrow$  .

For programming languages where the matrix storage format is C-style (row-major, such as C and Python), if the flattened row and column indices retain their original order, and the row indices precede the column indices (as in the example above), the flattening operation does not modify the data storing the tensor or matrix, only the shape parameters, so the time cost is negligible. Note that some languages use F-style matrix storage (column-major, such as Matlab).

After flattening both tensors into matrices, the contraction of tensor indices can be represented as matrix multiplication, like

$$(T_1)^{a_3a_4}_{a_1a_2}(T_2)^{a_5a_6}_{a_3a_4} = T^{a_5a_6}_{a_1a_2} \quad \Leftrightarrow \quad$$  $$=$$,

where a_3, a_4 are the contracted indices.

3 Wang tiling

3.1 Tensor representation

In this subsection, we will present a close correspondence between the enumeration of Wang tilings and tensor contraction operations.

First, we generalize the classical Wang tile set to a $\mathbb{K}$-*Wang tile set*, where each tile carries a value from $\mathbb{K}$ as its *weight*, and the weights are multiplied during tiling. A weight of 0 means that it does not appear in the set. For example, the horizontal concatenation of two $\mathbb{K}$-Wang tiles is

$$(k_1 \cdot \alpha_1 \underset{b_1}{\overset{a_1}{\square}} \beta_1) \circ (k_2 \cdot \alpha_2 \underset{b_2}{\overset{a_2}{\square}} \beta_2) = \begin{cases} k_1k_2 \cdot \alpha_1 \underset{b_1}{\overset{a_1}{\square}} \overset{a_2}{\underset{b_2}{\square}} \beta_2, & \beta_1 = \alpha_2; \\ \emptyset, & \beta_1 \neq \alpha_2. \end{cases} \tag{1}$$

We refer to any combination of ($\mathbb{K}$-)Wang tiles obtained by concatenating them in this manner in arbitrary directions and an arbitrary number of times as a *($\mathbb{K}$-)Wang tiling*. Note that we only consider the characters on the boundary of a ($\mathbb{K}$-)Wang tiling; the characters in its interior (such as β_1 in the expression above) are ignored, as they play no role in subsequent tilings.

For any two $\mathbb{K}$-Wang tile sets, their concatenation is the sum over all pairwise concatenations of their respective tiles according to this rule. It can be observed that there is an obvious correspondence between $\mathbb{K}$-Wang tile sets and $\mathbb{K}$-tensors with 4 indices, where the characters on the four edges are treated as indices, and the weight of the tile is treated as the value in the tensor corresponding to those indices, like

$$k \cdot \alpha \underset{b}{\overset{a}{\square}} \beta \mapsto T_{b\beta\alpha a} = k.$$

This yields the *tensor representation* of a $\mathbb{K}$-Wang tile set: it is a fourth-order tensor, each entry of which is given by the weight of the corresponding tile according to the above rule. Thus, we can uniformly express the $\mathbb{K}$-Wang tile set and its tensor representation using

an expression of the form

$$\sum_{b\in\Sigma}\sum_{\beta\in\Theta}\sum_{\alpha\in\Theta}\sum_{a\in\Sigma} T_{b\beta\alpha a}\cdot \alpha\begin{array}{|c|}\hline a\\ \hline b\\ \hline\end{array}\beta.$$

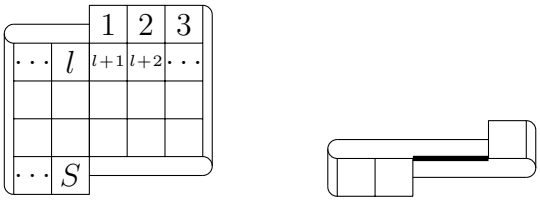
Building on this, one can see that the concatenation rule for $\mathbb{K}$-Wang tile set is exactly consistent with the contraction rule for tensors: concatenation (contraction) is allowed only when the characters (indices) match, and the weight of the result is the sum of the products of the weights of the two composable parts. Thus, horizontal and vertical concatenations of the $\mathbb{K}$-Wang tile sets can be respectively represented as horizontal and vertical index contractions of the corresponding tensors. Such a correspondence can obviously be extended from the $\mathbb{K}$-Wang tile set to the $\mathbb{K}$-Wang tiling set as well.

When $\mathbb{K}$ is taken as $\mathbb{N}$ or $\mathbb{Z}$, the weight of a tile, or a tiling, can be understood as the enumeration. A tile (or tiling) having weight k means that there are k distinct such tiles (or tilings) with the same edge characters. In this case, calculating the total weight after the tensor contraction under a given order is equivalent to enumerating the tilings under this order.

3.2 In helicoids

We first discuss the enumeration problem of Wang tilings in a *helicoid* (which is called a *twisted cylinder* in [16]), as its properties are more convenient for demonstrating the principles of our algorithm and are sufficiently general.

The construction of the helicoid is as follows: take a strip composed of S squares connected horizontally, and wrap it around a cylinder with circumference l, such that the square with index $i+l$ is placed below the square with index i (as shown below, the numbers in the squares represent their indices).

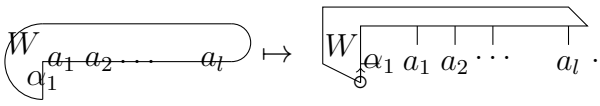


It is not difficult to see that when $S \geqslant l$ (as shown in the left), the top and bottom edges of the helicoid always have the same shape: a twisted line of length l, connected at both ends by a unit vertical line. For the degenerate case where $S < l$ (as shown in the right), the boundary of the strip can be extended (i.e., adding the bold lines in the figure) to maintain this edge shape.

If tiles are placed in the helicoid in the order of the original horizontal strip, the remaining region will still be a helicoid with circumference l, and the shape of the edges will not change.

We assign a character from the vertical alphabet Σ to each horizontal edge of the helicoid's top edge (ignoring the bottom edge for now), and a character from the horizontal

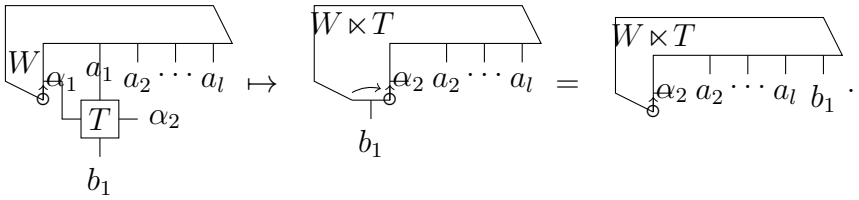
alphabet Θ to each vertical edge, and always treat *the lower endpoint of the vertical edge as the starting point*. The characters on the vertical edge (as indices) are arranged *counterclockwise* to form a word (character sequence) of length $l+1$, taken from $\Theta\Sigma^l$. The combination of such words with weights from $\mathbb{K}$ (we refer to them as $\mathbb{K}$-*word sets*) can be represented as tensors in the same way as $\mathbb{K}$-tile sets, like



Next, we take $\mathbb{K}$ as $\mathbb{Z}$ and discuss the enumeration algorithm for such tilings. Consider (the tensor representation of) the $\mathbb{K}$-word set of the word on the top edge of the helicoid at a certain moment as $W_{\alpha_1 a_1 a_2 \dots a_l}$. At this time, place a Wang tile $T_{b_1\alpha_2\alpha_1 a_1}$ at the first position of the helicoid region (that is, the Γ-shaped bend on the top edge). The enumeration of the new word on the edge (note that the starting point has moved one unit to the right) is

$$(W \ltimes T)_{\beta a_2 \dots a_l b_1} \coloneqq \sum_{\alpha a_1 \in \Theta\Sigma} W_{\alpha a_1 a_2 \dots a_l} T_{b_1\alpha_2\alpha_1 a_1},$$

and the string diagram is



That is, the indices $\alpha_1 a_1$ on the seam are contracted, and then the index b_1 corresponding to the character below the tile is rotated to the end (corresponding to the movement of the starting point). Without changing the encoding, the above can also be expressed in the form of matrix operations, such as

$$(W \ltimes T)^{a_3 \dots a_l b_1}_{\alpha_2 a_2} \coloneqq W^{a_2 \dots a_l}_{\alpha_1 a_1} T^{\alpha_1 a_1}_{b_1 \alpha_2}.$$

However, directly multiplying the two matrices on the right would result in a matrix with indices $(WT)^{a_2 \dots a_l}_{b_1 \alpha_2}$, which differs from our defined $W \ltimes T$ by an index rotation. The reason for defining $W \ltimes T$ in this way is to ensure that in each matrix multiplication, the two contracted indices $\alpha_1 a_1$ of the left operand are always its first two indices, thus ensuring that the index contractions can be converted into matrix multiplications.

To save the computational cost of rotating indices, we transform the above operations into equivalent batched matrix multiplications.

Note that if the free index b_1 is fixed to $b^* \in \Sigma$ (i.e., taking a slice of the tensor along this index), then

$$(W \ltimes T)_{\alpha_2 a_2 \dots a_l b^*} = W^{a_2 \dots a_l}_{\alpha_1 a_1} T^{\alpha_1 a_1}_{b^* \alpha_2},$$

where the encoding of $T^{\alpha_1 a_1}_{b^* \alpha_2}$ equals to a slice of the tensor $T_{b_1 \alpha_2 \alpha_1 a_1}$. Let $\Sigma = \{0, 1, \ldots, (\sigma - 1)\}$, according to the tensor encoding rules, the encoding of $(W \ltimes T)_{\alpha_2 a_2 \ldots a_l b_1}$ equals to the concatenated encoding of

$$(W \ltimes T)_{\alpha_2 a_2 \ldots a_l 0}, (W \ltimes T)_{\alpha_2 a_2 \ldots a_l 1}, \ldots, (W \ltimes T)_{\alpha_2 a_2 \ldots a_l (\sigma-1)}.$$

Therefore, to obtain the encoding of $(W \ltimes T)$, we only need to multiply $W^{a_2 \ldots a_l}_{\alpha_1 a_1}$ sequentially with

$$T^{\alpha_1 a_1}_{0 \alpha_2}, T^{\alpha_1 a_1}_{1 \alpha_2}, \ldots, T^{\alpha_1 a_1}_{(\sigma-1) \alpha_2},$$

and concatenate the resulting matrices vertically (this concatenation does not consume time, as it only requires connecting the output memory spaces).

When using NumPy, we can represent T as a 3-dimensional tensor with shape $(\sigma, \sigma\theta, \theta)$, which is obtained by stacking σ (corresponding to the first value of the shape parameter) matrices of size $\sigma\theta \times \theta$ (corresponding to the last two values)

$$T^{\alpha_1 a_1}_{0 \alpha_2}, T^{\alpha_1 a_1}_{1 \alpha_2}, \ldots, T^{\alpha_1 a_1}_{(\sigma-1) \alpha_2},$$

along the third dimension. Thus, when computing $W \ltimes T$, we only need to reshape W to $(\sigma^{l-1}, \sigma\theta)$ and perform NumPy's matrix multiplication. According to its broadcasting mechanism, since W lacks one dimension, it will first be expanded by adding a dimension and broadcasted (copied into σ copies and stacked), resulting in a 3-dimensional tensor with shape $(\sigma, \sigma^{l-1}, \sigma\theta)$. Then, the last two dimensions will be treated as the number of rows and columns for matrix multiplication, yielding a 3-dimensional tensor with shape $(\sigma, \sigma^{l-1}, \theta)$, which is exactly the tensor obtained by stacking the matrices

$$(W \ltimes T)^{a_2 \ldots a_l 0}_{\alpha_2}, (W \ltimes T)^{a_2 \ldots a_l 1}_{\alpha_2}, \ldots, (W \ltimes T)^{a_2 \ldots a_l (\sigma-1)}_{\alpha_2},$$

along the third dimension. For languages without such broadcasting mechanisms, the above operations are also straightforward to implement.

Starting from the initial top-edge word set that only contain $\sharp\sharp \ldots \sharp$ (with a weight 1), using this algorithm to place tiles sequentially and summing the results, we can calculate the number N of the word on the bottom edge after tiling the entire helicoid. Different positions can also specify different tile sets, i.e., defining

$$W_{i+1} = (W_i \ltimes T_i),$$

where T_i is the $\mathbb{Z}$ tile set at the position with index i.

However, when the word enumeration becomes too large, matrix operations are prone to data overflow. To address this issue, we select multiple prime numbers p (discussed later), and modify the above operations to modulo p operations, i.e., taking the result modulo p after each matrix operation. Thus, the tiling enumeration becomes a modulo p enumeration, projecting the ring $\mathbb{Z}$ of the tensor onto $\mathbb{Z}_p$, and the $\mathbb{Z}$-tensor is projected onto a $\mathbb{Z}_p$-tensor. It is easy to prove that such a projection preserves all the tensor operations used above.

According to the Chinese Remainder Theorem, if $\gcd(p, P) = 1$, then the system of congruences for R' (where $[R']_p$ denotes the congruence class of R' modulo p)

$$\begin{cases} [R']_p = [r]_p; \\ [R']_P = [R]_P \end{cases}$$

has the solution

$$[R']_{pP} = [P^- Pr + p^- pR]_{pP},$$

where $P^- \in [P]_p^{-1}$ and $p^- \in [p]_P^{-1}$ satisfy

$$P^- P + p^- p = 1.$$

Such P^- and p^- can be obtained using the Euclidean algorithm on (p, P).

Thus, the following steps can be used to compute the enumeration in the general sense through modular enumeration: First, estimate the lower and upper bounds of the desired enumeration, such as

$$\underline{N} \leqslant N < \overline{N}.$$

In the worst case, we can always take the lower bound as

$$\underline{N} = 0$$

(the final enumeration in the applications of this paper is always non-negative) and the exponential upper bound as

$$\overline{N} = \|T\|^{lh},$$

where $\|T\|$ denotes the ℓ_1-norm of the tile combination, i.e. the sum of the *absolute values* of all its entries. This bound remains valid when the weights may be negative: the weighted enumeration N is a sum of products of lh tile weights, so $|N|$ is at most the same sum with every weight replaced by its absolute value, which is in turn at most $\left(\sum |w|\right)^{lh} = \|T\|^{lh}$. (If negative weights are allowed and N itself may be negative, one can replace the interval $[\underline{N}, \overline{N})$ by the symmetric one $(-\overline{N}, \overline{N})$.) Depending on the problem, $\underline{N}$ and $\overline{N}$ may also be chosen in different ways to achieve tighter bounds. However, we will not further discuss specific choices here, because the *halting mechanism* introduced later will make the algorithm independent of concrete upper or lower bounds.

Then, assume N lies in the congruence class $[R]_p$. Starting with $R = 0$ and $P = 1$ (where $[0]_1 = \mathbb{Z}$), select sufficiently many prime numbers (this ensures $\gcd(p, P) = 1$) p for modular enumeration, and replace the original $[R]_P$ with the obtained $[R']_{pP}$, repeating the process as follows:

$$\begin{cases} P \mapsto pP; \\ R \mapsto (P^- Pr + p^- pR) \ \% \ pP. \end{cases}$$

Here, % denotes the modulo operation, ensuring $R < P$. After repeating this process a sufficient number of times until $P > \overline{N} - \underline{N}$, there will be only one number in $[R]_P$ that falls within $[\underline{N}, \overline{N})$, which is the desired enumeration. This number can be obtained using

$$N = \underline{N} + (R - \underline{N}) \ \% \ P.$$

When the product P of the accumulated primes is less than $(\overline{N} - \underline{N})$, the above formula may not yield the desired enumeration; however, when $P \geqslant (\overline{N} - \underline{N})$, N will be the required enumeration and will remain fixed, with no further changes to R or P affecting its value.

In this case, to make the algorithm stop faster, a *halting mechanism* can be added: given an integer $\tau_{\text{halt}} > 0$, stop when the above formula yields the same N for τ_{halt} consecutive rounds of solving the congruence equations, and output this N as the result. Note that R does not change if and only if

$$r \equiv R \pmod{p} \quad \Longleftrightarrow \quad N \equiv R \pmod{p} \quad \Longleftrightarrow \quad p \mid (N - R),$$

i.e., the new residue r coincides with the current residue R modulo p, which for a "random" N has a probability (though not strictly mathematical) of approximately $\frac{1}{p}$.

Therefore, under the above setup, the probability of misjudgment is approximately the reciprocal of the product of the last τ_{halt} primes used. At the same time, if a misjudgment occurs, the deviation is at least the product of these τ_{halt} primes. As long as the probability of misjudgment is acceptable, or the deviation is large enough to be detected (if an error is detected, increase τ_{halt} and recompute), the above setup can be used to stop the computation more promptly.

Next, we discuss the selection of prime numbers p. If p is chosen too large, it may still cause data overflow, but under the premise of avoiding overflow, the larger p is, the more efficient the algorithm becomes. Therefore, we need to determine the upper bound $\overline{p}$ for the selection of p, and search for primes starting from this number downward to use as p.

Note that each matrix multiplication in the algorithm takes the form

$$W^{\cdots}_{\alpha_1 a} T^{\alpha_1 a}_{b^* \alpha_2},$$

i.e., the rows of $W^{\cdots}_{\alpha_1 a}$ are multiplied by the columns of $T^{\alpha_1 a}_{b^* \alpha_2}$. Here, the entries of $W^{\cdots}_{\alpha_1 a}$ are uncertain, but after taking modulo p, they do not exceed $p - 1$. There are two cases to consider:

(1) The terms of T are all non-negative.

In this case, assume the data type is a 32-bit unsigned integer, with a range from 0 to $2^{32} - 1$. In a tiling enumeration where the tile sets at each position are T_i, we define

$$S_{\text{col}} \coloneqq \max_i \max_{\alpha_2} \sum_{a \alpha_1} (T_i)^{\alpha_1 a}_{b^* \alpha_2},$$

as the maximum column sum of $(T_i)^{\alpha_1 a_1}_{0 \alpha_2}$ during matrix multiplication. Then, the largest possible number obtained from row-column multiplication should satisfy

$$(p - 1) S_{\text{col}} \leqslant 2^{32} - 1.$$

Thus, p only needs to be no larger than the upper bound

$$\overline{p} = \left\lfloor \frac{2^{32} - 1}{S_{\text{col}}} \right\rfloor + 1$$

to ensure no data overflow occurs.

(2) T has negative terms.

In this case, assume the data type is a 32-bit signed integer, with a range from -2^{31} to $2^{31}-1$. We need to ensure that the data does not overflow in either the positive or negative direction.

Let T_i^+ denote the tensor obtained by retaining only the positive entries of T_i (setting all other entries to 0), and let T_i^- denote the tensor obtained by retaining only the negative entries. Define

$$S_{\text{col}}^+ :- \max_i \max_{\alpha_2} \sum_{a\alpha_1} (T_i^+)_{b^*\alpha_2}^{\alpha_1 a};$$
$$S_{\text{col}}^- :- \min_i \min_{\alpha_2} \sum_{a\alpha_1} (T_i^-)_{b^*\alpha_2}^{\alpha_1 a},$$

where S_{col}^- is thus the *most negative* column sum (the entries of T_i^- are non-positive). Then, the largest and smallest possible numbers obtained from row-column multiplication should satisfy

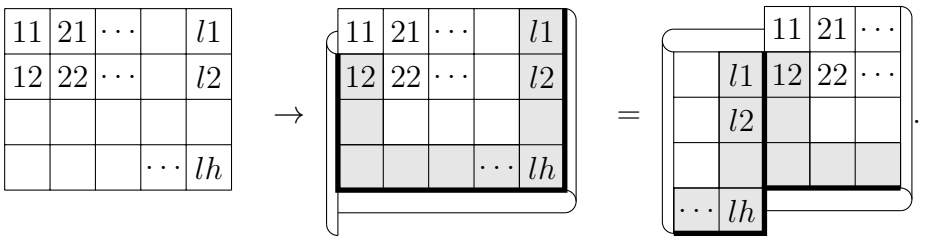
$$(p-1)S_{\text{col}}^+ \leqslant 2^{31}-1;$$
$$(p-1)S_{\text{col}}^- \geqslant -2^{31}.$$

Thus, the upper bound for p should be

$$\overline{p} = \left\lfloor \min\left(\frac{2^{31}-1}{S_{\text{col}}^+}, -\frac{2^{31}}{S_{\text{col}}^-}\right)\right\rfloor + 1.$$

3.3 In rectangle chessboards

In this section, we consider the following Wang tiling in a chessboard: Tiling a $l \times h$ (width and height, respectively) chessboard with Wang tiles, where each position on the boundary of the region has the character ♯ (no longer shown in the figures). Note that a rectangle can always be rolled into a helicoid by horizontally joining the pair of positions with horizontal and vertical indices (l, j) and $(1, j+1)$ together, like



The Wang tiling in the chessboard can be viewed as Wang tiling in a helicoid with the initial top edge marked by ♯, with the additional restriction that the characters on the bold lines in the figure must be ♯. This can be achieved by modifying the tile sets at the boundary positions (shaded regions in the figure). For example, for all tile sets on the

right edge, only tiles with the right edge marked by ♯ are retained. Apply this processing operation separately to the left, right, and bottom edges.

If we fix l and want to quickly provide the enumeration for each height h, we can apply the above processing only to the left and right edges, and then perform tiling layer by layer. After each layer is fully tiled, note that the weight of the word $\sharp\sharp\ldots\sharp$ in the resulting $\mathbb{Z}_p$-word set is exactly a specific term of the state tensor at that moment. If ♯ is the first character in the alphabet, then according to the encoding order, its weight is always the first term of the tensor. We only need to extract this term as the modulo enumeration of tilings satisfying the bottom edge condition at the current height.

It is worth noting that since our algorithm does not require a common tile set for all positions, it also does not require the character sets at each position to be the same, only that the character sets on adjacent edges match. Therefore, after processing the tile sets on the edges as described above, we can retain the original character set on the edge or directly change the character set on that edge to $\{\sharp\}$. However, for simplicity, we usually fix the horizontal and vertical character sets to sizes θ and σ.

Next, we discuss the complexity of the algorithm. At each moment, the $\mathbb{Z}_p$-word set of the twisted ring is taken from $\Theta\Sigma^l$, so the state tensor (representing this $\mathbb{Z}_p$-word set) has a size of

$$\sigma^l\theta.$$

The state tensor constitutes the bulk of the algorithm's space cost. During program execution, two tensors need to be maintained (one before tiling and one after), so the *space cost* of the algorithm is

$$\Theta(\sigma^l\theta).$$

Estimating the time cost of the algorithm, however, is considerably more difficult.

First, consider the time cost of a single tiling operation performed at a given position. Each tiling operation performs σ matrix multiplications of size $\sigma^{l-1}\times\sigma\theta$ and $\sigma\theta\times\theta$. Using the naive matrix multiplication algorithm, the total complexity of these σ multiplications is which requires traversing the state tensor once, resulting in a time complexity of

$$\Theta(\sigma^{l+1}\theta^2).$$

Assuming we use n_p primes in total and each prime requires tiling operations at $S = lh$ positions, the total time cost of the algorithm is:

$$\Theta(n_p lh\sigma^{l+1}\theta^2).$$

The problem is that n_p here is not fixed. It depends not only on the chosen Wang decomposition, but also on the growth rate of the enumeration we are computing. We can only give a rough estimate of it.

If the desired enumeration is $N(l, h)$, the distance between the upper and lower bounds is sufficiently large, and the automatic halting mechanism mentioned in the previous section is used to determine the number of primes selected, the algorithm will stop shortly

after the product of the used primes exceeds $N(l,h)$. Since the primes used at this point are all sufficiently close to $\overline{p}$, the number of primes used can be estimated as

$$n_p \approx \frac{\ln N(l,h)}{\ln \overline{p}},$$

where the constant $\overline{p}$ is determined by the selected $\mathbb{Z}$-tile set (see the end of Section 3.2).

For most tiling enumeration problems, the value of $N(l,h)$ grows exponentially with lh (see the end of Section 5.3):

$$N(l,h) \sim \lambda^{lh},$$

which allows us to make the estimate

$$n_p \approx \frac{lh \ln \lambda}{\ln \overline{p}}.$$

In this case, the time cost of the algorithm can be roughly estimated as

$$\Theta\left(\frac{\ln \lambda}{\ln \overline{p}} \cdot l^2 h^2 \sigma^{l+1} \theta^2\right).$$

Since given that estimating the time cost is rather complicated and its growth rate does not differ greatly from that of the space cost, in this paper we generally use the space cost, i.e., the size of the state tensor, to measure the complexity of the algorithm and to discuss its optimization.

For example, when

$$\sigma^l \theta > \theta^h \sigma,$$

that is, when the chessboard is sufficiently wide:

$$\frac{l-1}{h-1} > \frac{\ln \theta}{\ln \sigma},$$

the entire tiling region and tile sets should be transposed (thereby swapping l and h) before enumerating to reduce the size of the state tensor, thereby improving efficiency.

3.4 In regions of other shapes

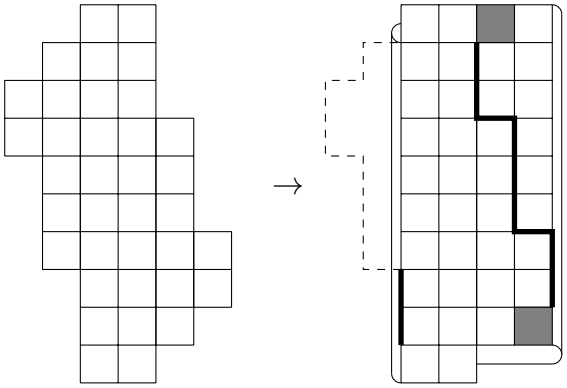
Our algorithm can also be further applied to the enumeration of Wang tilings in finite regions of other shapes. The approach remains to reduce the problem to Wang tilings on a helicoid. We will provide three examples.

Example 1. In irregular chessboards.

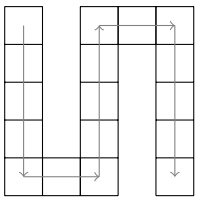
For an irregular chessboard, it can be viewed as a helicoid with certain blocks removed,

which is equivalent to that each of these positions is occupied by a Wang tile  . For

example, consider the 3*rd order* 3-*pillow* (from [3]) shown below: restrict the characters on bold lines to be ♯, and assign  as the (𝕂-)Wang tile set to each of the gray positions, so the tiling enumeration in this area is converted into the tiling enumeration in a helicoid.



In fact, the enumeration of Wang tilings in irregular regions is equivalent to the contraction of tensors with irregular connectivity. Such calculation can often be optimized by adjusting the order of the tensor contraction, potentially leading to higher efficiency than our algorithm. For instance, consider the snake-shaped region shown below: if embedded in a helicoid, the size of the state tensor reaches a maximum of $\theta\sigma^5$. However, if the tensors are contracted in the order indicated by the arrows, the maximum size is only $\max(\theta, \sigma)$.



Nevertheless, finding the most efficient contraction order is a complex problem [24], and we will not delve further into this discussion here.

Example 2. In $l \times h$ cylinders.

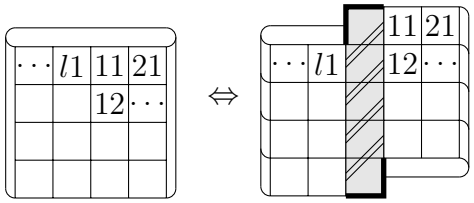
We have two different methods for handling tilings on a cylinder. Although both can handle arbitrary cylinders, they differ in computational efficiency, and one should choose the more efficient one depending on the specific situation.

For the first method, consider a *vertically oriented* cylinder, where l is the circumference and h is the height. This is equivalent to removing the offset during the assembly of the helicoid or applying a one-step offset in the opposite direction. We can achieve this

effect using the following Wang tile set:

$$\sum_{\alpha\in\Theta}\sum_{\beta\in\Theta}\begin{array}{ccc}&\alpha&\\ \alpha&\square&\beta\\ &\beta&\end{array}.$$

Note that we always consider the Wang tile set as a $\mathbb{K}$-Wang tile set with weights assigned as either 0 or 1, thereby enabling the summation over it. By adding a column of positions at the seam (the gray regions in the figure below) where the $l\times h$ chessboard is rolled and joined into a helicoid, assigning this tile to these positions, and restricting the characters on the boundary (the bold lines) to be $\sharp$, the resulting tilings correspond one-to-one with Wang tilings on the $l\times h$ cylinder. The long equal sign in the figure shows the path of these Wang tiles passing horizontal characters.



Note that the characters on the top and bottom edges of this tile set are horizontal characters, which serve to record and transmit horizontal characters. In this case, the size of the state tensor is $\theta^2\sigma^l$, which is only a factor of θ larger than that of the $l\times h$ chessboard.

And for the another method, for convenience, we now consider a *horizontally oriented* cylinder, where l is the height of the cylinder (but not the height of the chessboard) and h is the circumference. This is equivalent to gluing the top and bottom edges of the chessboard.

First, replace each Wang tile in the ($\mathbb{K}$-)tile set as follows (keeping the weights unchanged):

$$\begin{array}{ccc}&a&\\ \alpha&\square&\beta\\ &b&\end{array}\mapsto\sum_{c\in\Sigma}\begin{array}{ccc}&ac&\\ \alpha&\square&\beta\\ &bc&\end{array}.$$

Here, Σ is the vertical alphabet of the top edge of the column where the tile set is located. That is, a new vertical character is appended to the vertical characters on the top and bottom edges, which records the character of the top edge of the column and transmits it downward during the tiling process.

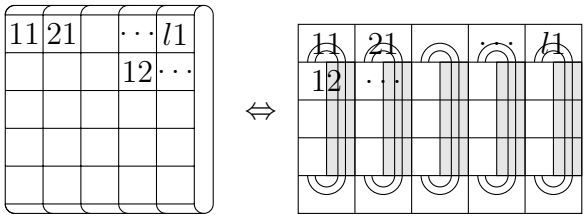
Thus, when tiling such Wang tiles on the $l\times h$ chessboard, the tiles at the top edge are replaced as follows:

$$\begin{array}{ccc}&ac&\\ \alpha&\square&\beta\\ &bc&\end{array}\mapsto\begin{cases}\begin{array}{ccc}&\alpha&\\ \alpha&\square&\beta\\ &ba&\end{array}, & a=c;\\ 0, & a\neq c.\end{cases}$$

That is, only the cases where the top two characters are equal are retained under the usual constraints (thus, the character c records the top character of this column). Similarly, at the bottom edge, we impose the constraint:

$$\alpha\begin{array}{|c|}\hline \\ \\ \hline \end{array}\beta \begin{array}{c}ac\\ \\ bc\end{array} \mapsto \begin{cases} \alpha\begin{array}{|c|}\hline \\ \\ \hline \end{array}\beta \begin{array}{c}ab\\ \\ \end{array}, & b = c; \\ 0, & b \neq c. \end{cases}$$

This is equivalent to requiring that the bottom and top characters of the column must both be c, effectively gluing the top and bottom edges of the chessboard, as shown in the figure below.



Since each vertical character is doubled, the size of the state tensor for the torus case increases from $\theta\sigma^l$ (for the rectangle chessboard case) to $\theta\sigma^{2l}$. If such a cylinder is transposed and processed using the first method, the tensor size (note the interchange of horizontal and vertical directions) becomes $\theta^h\sigma^2$. Therefore, the second method is more efficient if and only if

$$\sigma^{2l-2} < \theta^{h-1}$$

that is,

$$\frac{l-1}{h-1} < \frac{\ln\theta}{2\ln\sigma}$$

It can be seen that for fixed σ, θ, the second method is more suitable when the cylinder is sufficiently "thick" (i.e., the circumference h is much larger than the height l of the cylinder).

Example 3. In $l \times h$ torus.

This is equivalent to applying the second method of Example 2 to the vertically oriented cylinder case (processed using the first method of Example 2) to join the top and bottom edges together. In this case, the size of the state tensor becomes $\theta^2\sigma^{2l}$. The complexity of the torus case is inevitably much higher than that of all the aforementioned regions with the same l and h, which is related to the topological structure of the torus: the torus has no boundary, so regardless of the order of tiling, information about all positions on the edges of the already tiled stack must be recorded.

Note that the topological structure of torus, like the rectangle chessboard, is transpose-symmetric. When $\sigma^{2l-2} > \theta^{2h-2}$, the entire structure can be transposed (the state tensor size becomes $\theta^{2h}\sigma^2$) to enhance computational efficiency.

4 Polyomino tiling

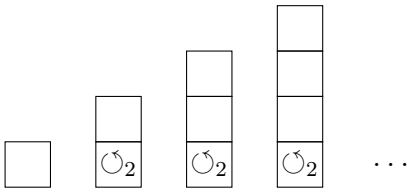
In this section, we will demonstrate how to transform polyomino tilings in a rectangle chessboard (which can be generalized to any area we discussed in the previous chapter) into Wang tilings, thereby enabling the use of our algorithm for enumeration.

All computational results in this chapter are saved at https://github.com/liang-kai-ruqi, and many of them have also been recorded in the corresponding sequences in OEIS.

Note that we always assume that polyominoes can be rotated, while Wang tiles cannot.

4.1 I-shaped polyominoes

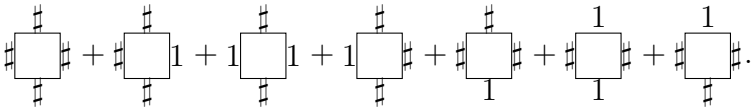
We first consider the problem of tiling enumeration of I-shaped polyominoes in a rectangle chessboard. I-shaped polyominoes refer to rectangular tiles of size $1 \times i$, as shown in the figure below.



We shall denote the first three by $\square, \boxminus, \begin{smallmatrix}\square\\\square\end{smallmatrix}$ respectively.

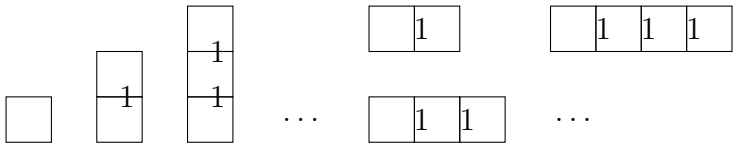
Example 4. Enumerate the tilings in a $l \times h$ chessboard using I-shaped polyominoes.

We select the horizontal and vertical alphabets as $\{\sharp, 1\}$, where $\sharp$ is the boundary character. The Wang tile set (also treated as a $\mathbb{Z}$-tile set with weights 0 or 1) is defined as



Here, $\Sigma = \Theta = \{\sharp, 1\}$, $\sigma = \theta = 2$. Hereafter, we shall always assume that the horizontal and vertical alphabets of the Wang tile set consist of all the symbols that appear in the horizontal and vertical directions, respectively.

Observed that the tiles from the tile set above can and can only form polyominoes with *edges consisting only of boundary characters* as shown below (boundary characters $\sharp$ are no longer drawn).



This exactly matches the polyominoes required by the problem, and each polyomino corresponds to a *unique* decomposition into the Wang tile set we provided. We call such a Wang tile set a *Wang decomposition* of the polyomino. Thus, if a polyomino in any region has a Wang decomposition, its tiling enumeration is equal to the tiling enumeration of its Wang decomposition, which our algorithm can be used for.

For $\mathbb{Z}$-Wang tile sets, the above *one-to-one correspondence* can naturally be relaxed to *enumeration correspondence*. We will provide two examples in Section 4.3 and Section 4.4.

We will provide a more detailed demonstration of the calculation process for this example based on this Wang decomposition, as the processes for other examples in this paper are similar.

In this paper, we always select the data type of integer variables as signed or unsigned 32-bit integer (with a size of 4B). Based on the tile set in this example, to ensure no data overflow occurs, the numbers in the state tensor should not exceed 1431655765. Starting from this number, we select 100 primes downward, obtaining the sequence

$$1431655751, 1431655747, 1431655739, \ldots, 1431653609, 1431653599.$$

The product of these primes is approximately 3.8355×10^{915}, ensuring it is greater than the number we need to compute. This shows that although primes become sparser as numbers grow larger, they are more than sufficient for our algorithm.

We set the halting mechanism to halt enumerating when the remainder R for each height remains unchanged for 2 consecutive iterations. The estimated probability of misjudgment is less than

$$\frac{1}{1431653609 \times 1431653599} \approx 4.88 \times 10^{-19},$$

which is entirely acceptable. Moreover, based on the growth trend of the computed arrays, misjudgments can be easily detected.

We compute all cases for $l \leqslant 27$ for each $h \leqslant 27$, and a part of the results are shown in the Table 1:

It can be seen that, whether for fixed width or when width equals height, the number of tiling solutions roughly exhibits exponential growth as the height increases. In fact, most polyomino tiling enumeration in a rectangle chessboard follow this pattern, while a few exhibit periodicity on top of this, such as periodic fluctuations or periodic occurrences of enumeration being 0.

Additionally, the computation time and primes number n_p used for each width are shown in the Table 2:

The total computation time for the above was 45175s, with 53.667% of the time spent on computing the case with the largest width. It can be observed that, for a fixed maximum height, the computation time increases roughly exponentially with the width l of the tiling region: Roughly speaking, each increase in width by 1 doubles the computation time. This highlights the significant impact of chessboard width on algorithm complexity.

h	$l=1$	$l=2$	$l=3$	$l=h$
1	1	2	4	1
2	2	7	29	7
3	4	29	257	257
4	8	124	2408	50128
5	16	533	22873	50796983
6	32	2293	217969	264719566561
7	64	9866	2078716	7063448084710944
8	128	42451	19827701	963204439792722969647
9	256	182657	189133073	670733745303300958404439297
10	512	785932	1804125632	2384351527902618144856749327661056
...	...	...	...	...
				112141233
				917823545347805132535903225917610 47
				826210096197536299672146260608017 82
				865917341524452757526933755090388 06
				862903059369327478680078594815619 49
	67	46678	808678	279587429973704839084035900445193 71
	108	068726	8352164238	848791223722921757369873374171021 46
27	864	464909	5015065537	708459364321443285468879091314955 53

Table 1: Some results of I-shaped polyominoes tiling enumeration in rectangle chessboards

l	time	n_p	l	time	n_p	l	time	n_p
1	0.002s	3	10	0.037s	6	19	51.42s	21
2	0.003s	4	11	0.075s	13	20	112.5s	22
3	0.002s	5	12	0.149s	14	21	236.5s	23
4	0.003s	6	13	0.564s	15	22	508.9s	24
5	0.003s	7	14	0.757s	16	23	1101s	25
6	0.005s	8	15	1.561s	17	24	2415s	26
7	0.006s	9	16	3.823s	18	25	5218s	27
8	0.011s	10	17	10.66s	19	26	11427s	28
9	0.021s	12	18	22.90s	20	27	24244s	29

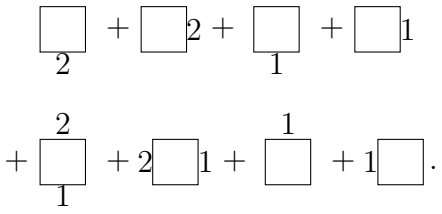
Table 2: Calculation time and used primes numbers of I-shaped polyominoes tiling enumeration in rectangle chessboards

Furthermore, the most time-consuming step of the algorithm is always matrix operations (multiplication and modulo), while the time spent on other steps (such as solving congruence equations) is negligible.

Special I-shaped polyominoes can also be decomposed into Wang tiles. For example:

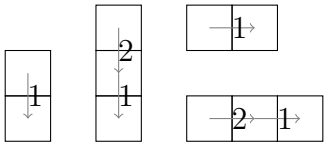
Example 5. Enumerate the tilings in a $l \times h$ chessboard using $\square + \square$.

These two polyominoes have Wang decompositions (hereafter, characters not shown are all $\sharp$):

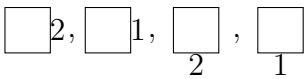


Here, $\sigma = \theta = 3$.

The tiles in the first row are *root tiles*, while the others, called *branch tiles*, are used for *growing* the tiles. The numbers on each tile indicate the number of tiles needed to grow to the right or downward from that position (as shown in the figure below).

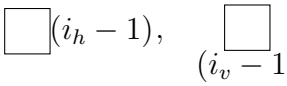


This approach of *tiles growing* is a general method for designing Wang decompositions of I-shaped polyominoes. We can see that the *root tiles*

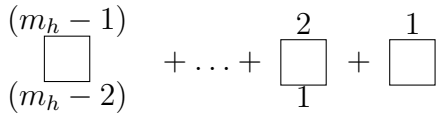


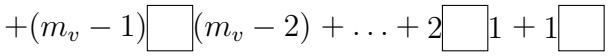
determine the total number of tiles (equal to the character on each of them added 1) and the direction (the orientation of the character), while the remaining tiles (which we call *branch tiles*) grow sequentially in that direction to produce a given number of tiles, thereby forming the desired polyominoes.

For other I-shaped polyomino sets, we only need to replace the root tiles with



where i_h and i_v range over the total numbers of tiles of the horizontal and vertical polyominoes in the set, respectively; and add a sufficient number of branch tiles of the form



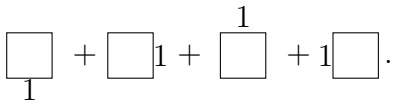


where m_h and m_v are the maximum total numbers of tiles among the horizontal and vertical polyominoes, respectively, which are also equal to the number of horizontal and vertical characters in this case.

For two simpler examples:

Example 6. Enumerate the tilings in a $l \times h$ chessboard using ⊟ (dominoes).

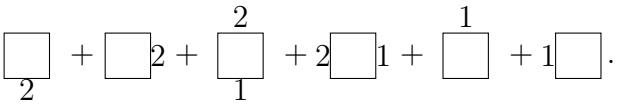
It has a Wang decomposition:



Here, $\sigma = \theta = 2$. However, this case has a specific formula for the general situation (See Introduction), so we will not compute it further.

Example 7. Enumerate the tilings in a $l \times h$ chessboard using ⊟ (straight trominoes).

It has a Wang decomposition:



Here, $\sigma = \theta = 3$.

4.2 Small polyominoes

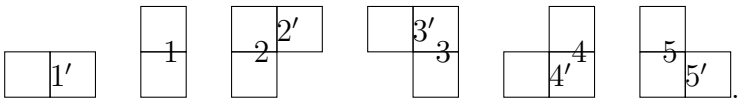
For any polyomino that admits a suitable Wang decomposition, its tiling enumeration problem (in regions of any shape) can be transformed into a tensor network contraction, thereby allowing our algorithm to be applied. Therefore, how to find a Wang decomposition becomes a key issue.

For a polyomino set of *finite area*, defined as a set whose total area is finite, we have:

Proposition 8. *Any finite-area polyomino set must have a Wang decomposition.*

Proof. This is because, in the worst case, one can enumerate every possible orientation of each polyomino, add distinct non-boundary characters to each internal edge, and then separate the tiles. It is easy to see that the resulting Wang tile set is a Wang decomposition.

For example, the Wang decomposition for $\square\!\square + \square\!\square$ obtained in this way is



□

However, the above proposition only guarantees the existence of a Wang decomposition. An excessive number of characters (especially vertical characters) can significantly reduce the efficiency of the algorithm. Therefore, for polyomino sets with more regular structures, it is best to employ some techniques to minimize the number of characters when searching for a Wang decomposition.

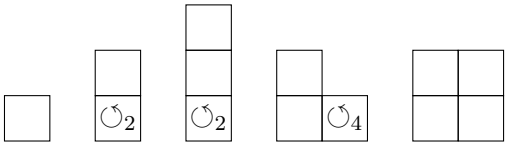
Based on our previous discussion, the size of the state tensor increases exponentially with the number of vertical characters, while it increases only linearly with the number of horizontal characters. Therefore, in order to minimize the size of the state tensor as much as possible, we can define the *optimal Wang decomposition* as follows:

(1) The number of vertical characters should be as small as possible.

(2) On the basis of (1), the number of horizontal characters should be as small as possible.

In fact, we have not yet found a sufficiently general method to obtain the optimal Wang decomposition for a given polyomino set. This appears to be a rather complex computational problem. For different polyomino sets, we can only provide the corresponding Wang decomposition on a case-by-case basis according to their specific characteristics and verify them manually. Such an approach does not guarantee that the Wang decomposition found is optimal. See Section 5.3 for further discussion.

In this subsection, we list the Wang decompositions for all non-empty sets of polyominoes that chosen from the 5 shapes below.



We shall denote them by $\square, \square\!\square, \begin{smallmatrix}\square\\\square\end{smallmatrix}, \square\!\square\!\square, \square\!\square\\\square\!\square$ respectively. Thus, there are $2^5 - 1 = 31$ different non-empty polyomino sets in total. To reduce the number of cases that need to be considered, we have the following:

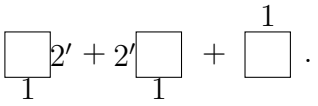
Proposition 9. *If a ($\mathbb{K}$)-Wang tile set $\mathcal{T}$ is a Wang decomposition of a polyomino set $\mathcal{M}$, then $\mathcal{T} + \square$ is a Wang decomposition of $\mathcal{M} + \square$.*

Proof. Obviously,  cannot participate in forming any polyomino other than □ (otherwise edge characters would appear in the interior); it can only form □ by itself, in isolation. Therefore, after adding  , the set of polyominoes that can be formed is exactly the original set plus □. □

Adding  does not change the original number of horizontal or vertical characters either, so in this subsection we will no longer list polyomino sets that include □.

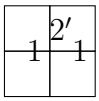
Example 10. Enumerate the tilings in a $l \times h$ chessboard using ⊞.

It has a Wang decomposition:



Here, $\sigma = \theta = 2$.

Clearly, these three Wang tiles can only form the shape of ⊞ in the way shown below.

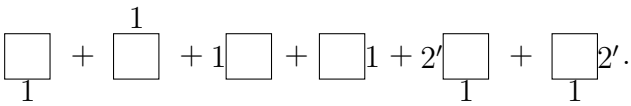


Although there is a vertical line inside the polyomino with a boundary character, it does not matter.

However, the enumeration of ⊞ tilings on a rectangle chessboard is obvious: it is 1 if both l and h are even, and 0 otherwise. Thus, we only need to use the above Wang decomposition and Proposition 9 to compute the case of □ + ⊞.

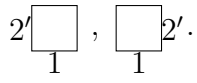
Example 11. Enumerate the tilings in a $l \times h$ chessboard using ⊟ + ⊞.

It has a Wang decomposition:



Here, $\sigma = 2$, $\theta = 3$.

That is, based on the Wang decomposition of ⊟ in Example 6, we add



These, together with $\overset{1}{\square}$, exactly form $\boxplus$ (as in the Example 10).

By analogy with this example, we've observed that:

Proposition 12. *If a ($\mathbb{K}$-)Wang tile set $\mathcal{T}$ is a Wang decomposition of a polyomino set $\mathcal{M}$, and it satisfies the following conditions:*

(1) It includes $\overset{1}{\square}$ *(with weight* 1*), and it's the only tile in $\mathcal{T}$ whose top label is* 1*.*

(2) It does not include the horizontal character $2'$*.*
Then,

$$\mathcal{T} + \underset{1}{\square}2' + 2'\underset{1}{\square}$$

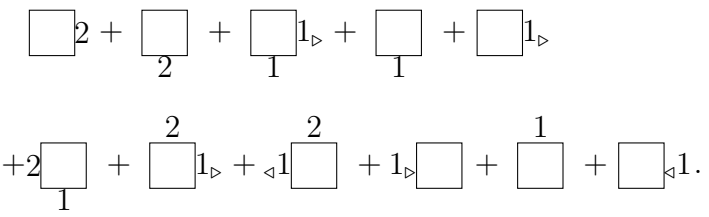
is a Wang decomposition of $\mathcal{M} + \boxplus$.

Proof. The principle is similar to that of 9. By condition (2), the two added tiles $\underset{1}{\square}2'$ and $2'\underset{1}{\square}$ can match the character $2'$ only with each other, so whenever one of them is used the other must be placed immediately to its right, and the resulting horizontal pair has both bottom labels equal to 1. By condition (3), each of these bottom labels can be matched only by $\overset{1}{\square}$ placed directly below. Since $\overset{1}{\square}$ has no other labels, these four tiles are forced to close up into exactly $\boxplus$(as in Example 10), and they cannot participate in forming any other polyomino. □

This operation on $\mathcal{T}$ increases the number of horizontal characters by one and does not change the number of vertical characters, having little impact on the algorithm's efficiency. Since all tile sets listed below in this subsection satisfy the above conditions, we will not list polyomino sets that include $\boxplus$ either.

Example 13. Enumerate the tilings in a $l \times h$ chessboard using $\boxminus + \boxplus$.

It has a Wang decomposition:



Here, $\sigma = 3$, $\theta = 4$.

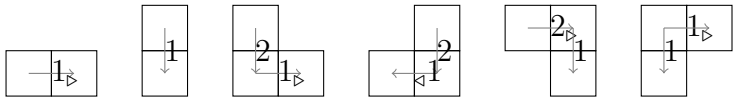
This is equivalent to selecting the tiles in the first row as *root tiles* for *tree-like growth* (which allows branching), with the following rules:

(1) The growth direction of the root tiles can only be downward or to the right; the growth direction of other tiles can only be downward, to the left, or to the right.

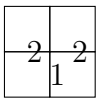
(2) A total of 2 or 3 tiles are grown (i.e., the sum of the numbers on the root tiles is 1 or 2).

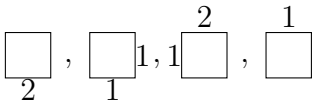
(3) The second growth direction must differ from the first. In other words, the corresponding branch tiles (all in the second row) must be *turning*.

It is easy to verify that this can uniquely obtain each orientation of $\square + \square\!\square$ under rotation (as shown below).



Note that horizontal growth has two distinct directions, so $_{\triangleleft}1$ and $1_{\triangleright}$ are used to distinguish them. Otherwise, an additional polyomino (as shown below) would appear.



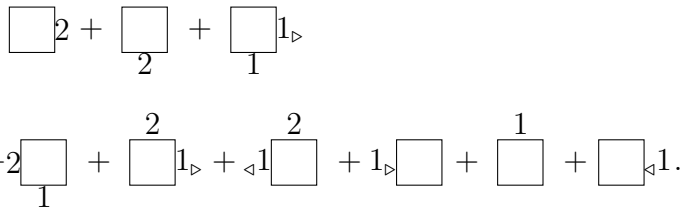
In view of this, one might try to obtain a Wang decomposition of $\square + \square\!\square + \square\!\square$ by merging the two characters $_{\triangleleft}1$ and $1_{\triangleright}$ into a single character 1; however, such a merge does *not* yield a Wang decomposition, because in addition to $\square\!\square$ it produces spurious assemblies (for instance, the merged tiles  assemble into a 4-cell zigzag shape that is neither a domino, a right tromino, nor a square). Therefore, the Wang decomposition of $\square + \square\!\square + \square\!\square$ is obtained by applying Proposition 12 to the decomposition above instead.

Example 14. Enumerate the tilings in a $l \times h$ chessboard using $\square\!\square$ (right trominoes).

By restricting the number of tiles to 3 in Example 13, i.e., removing the root tiles with 2 tiles



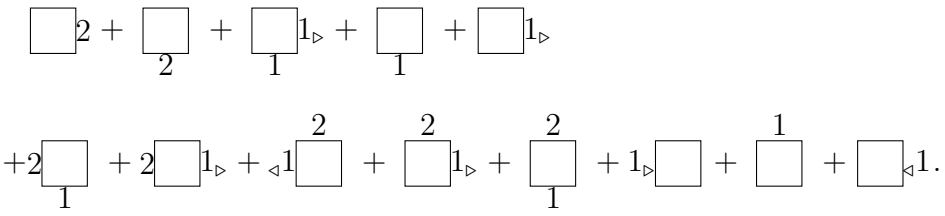
the resulting Wang decomposition is



Here, $\sigma = 3$, $\theta = 4$.

Example 15. Enumerate the tilings in a $l \times h$ chessboard using $\square + \begin{smallmatrix}\square\\\square\end{smallmatrix} + \square\!\square$ (dominoes and all trominoes).

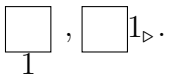
Simply remove rule (3) from Example 13, i.e., allow tiles to grow without turning. The resulting Wang decomposition is



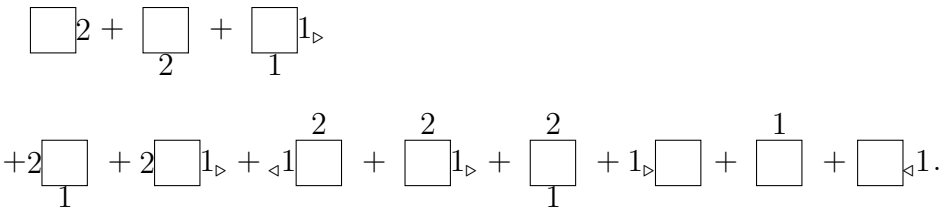
Here, $\sigma = 3$, $\theta = 4$.

Example 16. Enumerate the tilings in a $l \times h$ chessboard using $\begin{smallmatrix}\square\\\square\end{smallmatrix} + \square\!\square$ (all trominoes).

Simply restrict the number of tiles in rule (2) of Example 15 to 3, i.e., remove the corresponding root tiles



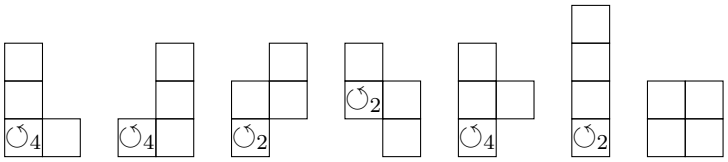
The resulting Wang decomposition is



Here, $\sigma = 3$, $\theta = 4$.

4.3 Tetrominoes

Tetrominoes, also known as 4-ominoes or *Tetris pieces*, have 7 different shapes (as shown below).



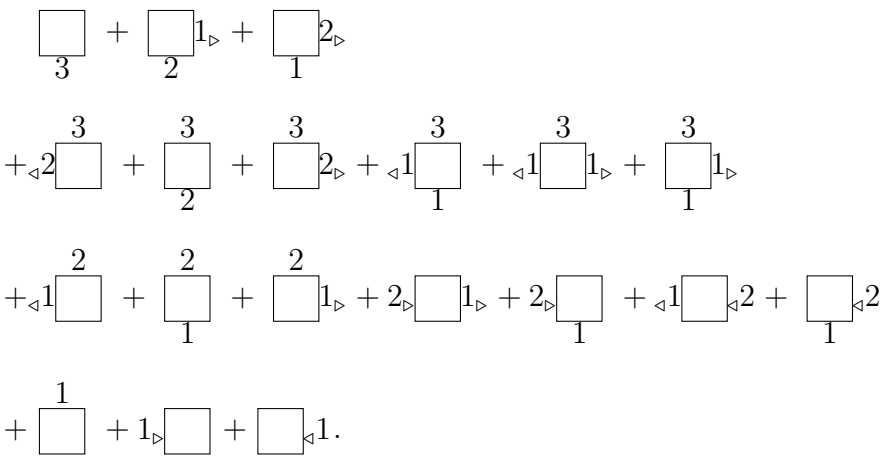
Example 17. Enumerate the tilings in a $l \times h$ chessboard using all tetrominoes.

Based on the growth rules in Example 16 (trominoes), we add one more to the total number of tiles by one, i.e.,

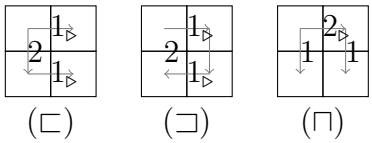
(1) The growth direction of the root tiles can only be downward or to the right; the growth direction of other tiles can only be downward, to the left, or to the right. Multiple directions can grow simultaneously.

(2) A total of 4 tiles are grown (i.e., the sum of the numbers on the root tiles is 3).

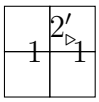
This results in the following tile set:



The only difference between this Wang tile set and a Wang decomposition of all tetrominoes is that it has 3 different ways of growth into $\boxplus$ (as shown below), causing repeated enumeration of $\boxplus$ three times.

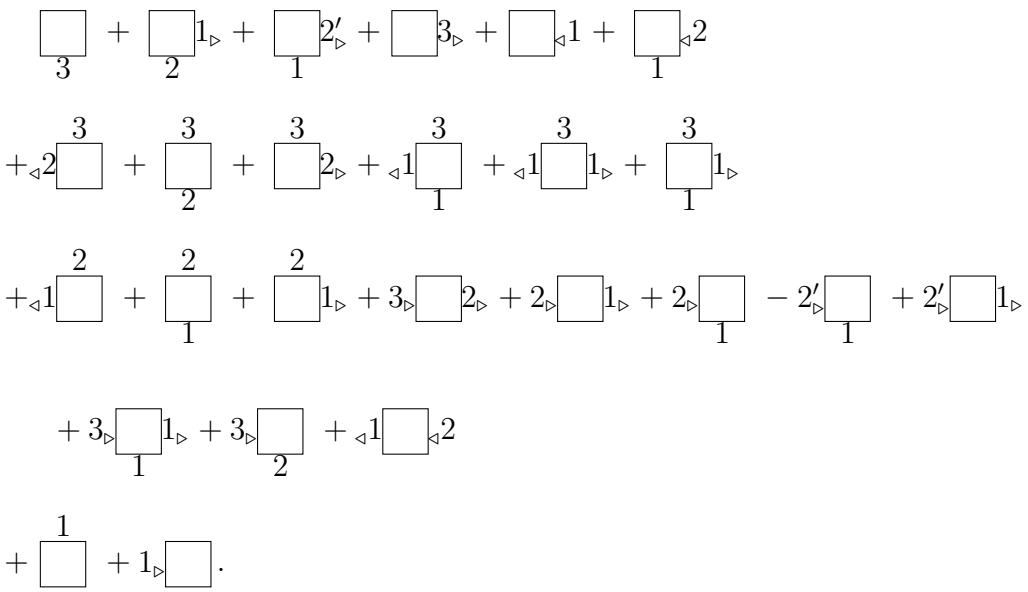


To resolve this issue, we split the horizontal character $2_{\triangleright}$ into $2_{\triangleright}$ and $2'_{\triangleright}$, where $2'_{\triangleright}$ indicates that the upstream tile grows both to the right and downward, and $2_{\triangleright}$ represents other cases. If the tile downstream of $2'_{\triangleright}$ grows downward, it forms a $\sqcap$-shape. We modify the weight of the downstream tile $2'_{\triangleright}$ (1) to -1, so that the weight of $\sqcap$-shaped grown $\boxplus$



is -1, which exactly cancels out the weights of the other two growth methods to yield 1, thereby eliminating the repeated enumeration of $\boxplus$.

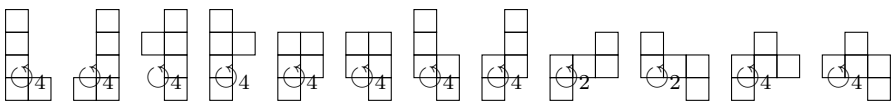
The resulting Wang decomposition of tetrominoes (as a $\mathbb{Z}$-tile set) is

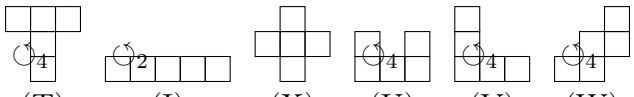


Here, $\sigma = 4$, $\theta = 7$.

4.4 Pentominoes

Pentominoes, also known as 5-ominoes, have 18 distinct shapes under rotation (as shown below, note that flipped symmetric pentominoes share the same letter as their label):

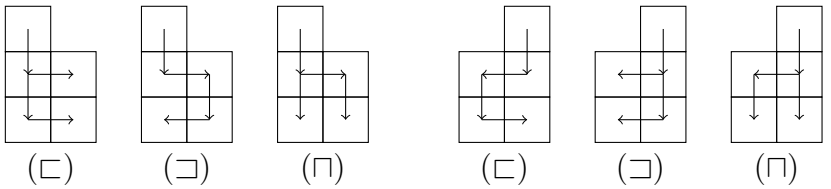




Example 18. Enumerate the tilings in a $l \times h$ chessboard using all pentominoes.

We continue using the approach in the previous example, add one more to the total number of tiles.

On this basis, note that any pentomino containing ⊞ (i.e., the two P-pentominoes, each with 4 orientations) has exactly 3 different ways of growth, such as



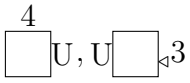
.

We need to modify the weight of each polyomino containing a ⊓-shaped growth to -1. This requires splitting the three characters $2_{\triangleright}$, $3_{\triangleright}$, and ${}_{\triangleleft}2$ into two characters each, as in the previous example, and then changing the weight of their downstream (either right or left) Wang tiles to -1.

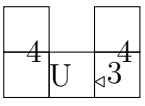
Additionally, since we prohibit upward growth, this tile set cannot form U-shaped pentomino with an upward opening (as shown below).



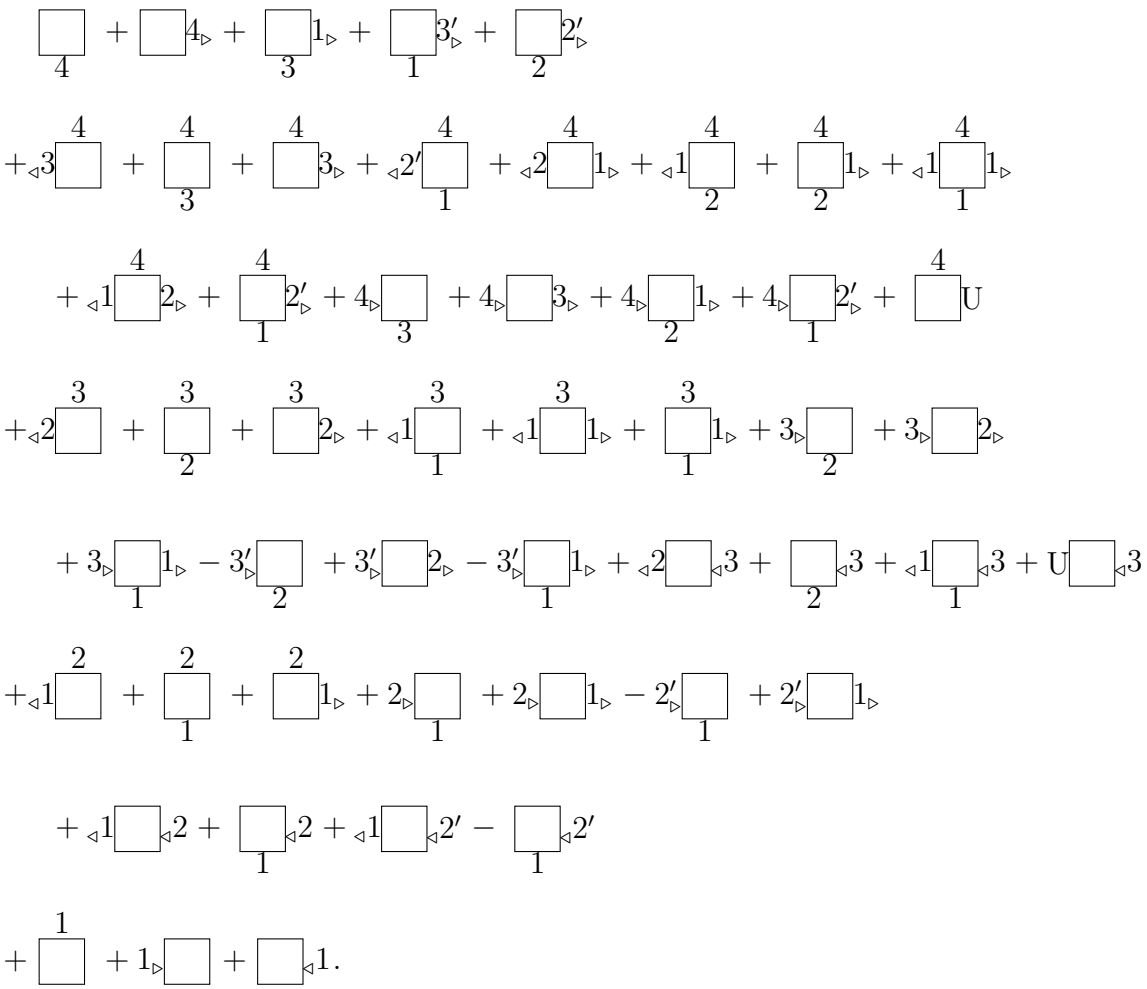
Therefore, we add the horizontal character U and the special tiles



which are specifically used to form the missing pentomino, as shown below.



This results in the Wang decomposition for all pentominoes:



Here, $\sigma = 5$, $\theta = 12$.

Clearly, in this case, the enumeration is positive only when at least one of l or h (without loss of generality, assume it's l) is divisible by 5. Table 3 below presents the calculation results for the cases where $l = 5, 10$ (while h ranges over $1, 2, \ldots, 20$), with computation times of 0.4228s and 3789s. Our algorithm achieved the first successful computation for the cases of $l = 10$ with large h.

4.5 Related OEIS sequences

Table 4 lists all the polyomino tiling enumeration presented in this paper that have corresponding sequences recorded in the OEIS, as well as the precious range and the range of results calculated by our algorithm (utilizing transpose symmetry, as well as certain special cases where the enumeration is 0). The device used for calculation is a conventional computer (using a single core of AMD ryzen 7 7840hs CPU).

$h=$	$l=5$	$l=10$
1	1	1
2	5	45
3	56	7670
4	501	890989
5	4006	101698212
6	27950	7845888732
7	214689	756605877809
8	1696781	75996685446347
9	13205354	7470920047174798
10	101698212	729748655181974778
11	782267786	70521242596066128006
12	6048166230	6882943628424155149082
13	46799177380	672858933871350579734838
14	361683136647	65670854176387745944044415
15	2793722300087	6406383348267533424844337077
16	21583392631817	624874119278590450628206097405
17	166790059833039	609781469454435553110940302063 23
18	1288885349447958	595071124448617043162690211995 7082
19	9959188643348952	580653334431250399171093742069162662
20	76953117224941654	566572849158404680394050157132257585 36

Table 3: Some results of pentominoes tiling enumeration in rectangle chessboards

Below the total calculation time is for the case of calculating the maximum width, and the ratio of it to the total time consumption can roughly reflect the rate at which the total time consumption increases with the width. The rightmost column shows the approximate value of the largest enumeration in the calculation results (although we calculate the exact value).

It can be observed that, compared to the traditional algorithms used in the original records, our algorithm improves computational efficiency in the vast majority of cases, thereby significantly expanding the computable range, especially for the cases with larger enumeration and more complex polyomino sets.

5 Further generalizations and open problems

5.1 Variants of tile shape

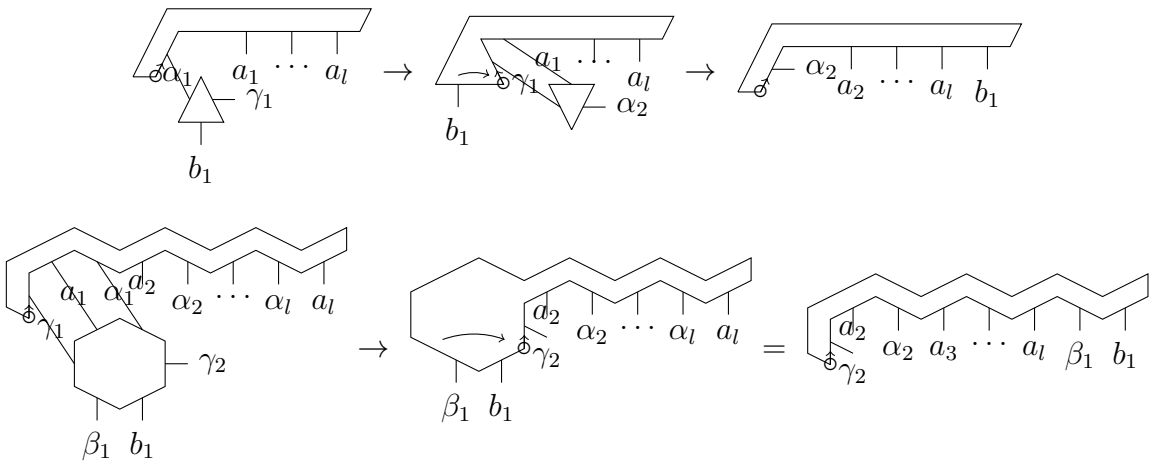
Our algorithm can also be generalized to the enumeration of tilings of more types. For example, *polyiamonds* and *polyhexes* are variants of polyominoes, where the unit tiles are equilateral triangles and regular hexagons, respectively, instead of squares. All the shapes of 4-iamonds (3 different shapes) and 3-hexes (3 different shapes) are shown in the figures below.

polyomino set	OEIS sequences and precious range	calculation range	time	maximum enumeration
I-shaped	A254414: $l+h \leqslant 30$ A254127: $l=h \leqslant 15$	$l+h \leqslant 55$	total: 53795s $l=27$: 28489s	3.306×10^{262} $(l=27, h=28)$
	A219866: $l+h \leqslant 22$ A219874: $l=h \leqslant 11$	$l+h \leqslant 35$	total: 9468s $l=17$: 6362s	7.757×10^{59} $(l=17, h=18)$
	A351322: $l+\frac{h}{3} \leqslant 31$	$l+\frac{h}{3} \leqslant 35$	total: 62954s $l=17$: 42719s	3.307×10^{94} $(l=17, h=54)$
	A220054: $l+h \leqslant 29$ A220061: $l=h \leqslant 16$	$l+h \leqslant 35$	total: 17955s $l=17$: 12106s	3.046×10^{77} $(l=17, h=18)$
	A219987: $l+h \leqslant 21$ A219994: $l=h \leqslant 17$	$l+h \leqslant 35$	total: 16380s $l=17$: 10987s	1.167×10^{68} $(l=17, h=18)$
	A353877: $l,h \leqslant 9$ A353934: $l=h \leqslant 16$	$l+h \leqslant 35$	total: 22518s $l=17$: 15107s	1.813×10^{107} $(l=17, h=18)$
	A233320: $l+h \leqslant 28$	$l+h \leqslant 35$	total: 13299s $l=17$: 8928s	2.173×10^{49} $(l=17, h=18)$
	A270061: $l+h \leqslant 25$ A270071: $l=h \leqslant 12$	$l+h \leqslant 35$	total: 19344s $l=17$: 13017s	6.584×10^{89} $(l=17, h=18)$
	A364457: $l+h \leqslant 25$ A364504: $l=h \leqslant 12$	$l+h \leqslant 35$	total: 18331s $l=17$: 12279s	4.358×10^{79} $(l=17, h=18)$
	A245013: $l+h \leqslant 45$ A063443: $l=h \leqslant 40$	$l+h \leqslant 55$	total: 19667s $l=27$: 9769s	8.952×10^{92} $(l=27, h=28)$
	A352431: $l+\frac{h}{2} \leqslant 40$	$l+\frac{h}{2} \leqslant 53$	total: 81727s $l=26$: 41408s	1.644×10^{231} $(l=26, h=54)$
	A352589: $l,h \leqslant 9$ A353777,$l=h \leqslant 17$	$l+h \leqslant 55$	total: 75583s $l=27$: 38039s	1.217×10^{224} $(l=27, h=28)$
	A219967: $l+h \leqslant 27$ A219975: $l=h \leqslant 12$	$l+h \leqslant 35$	total: 8339s $l=17$: 5589s	9.109×10^{27} $(l=17, h=18)$
	A219946: $l+h \leqslant 35$ A219952: $l=h \leqslant 17$	$l+h \leqslant 35$	total: 14942s $l=17$: 10017s	1.467×10^{36} $(l=17, h=18)$
	A353963: $l,h \leqslant 9$ A354067: $l=h \leqslant 16$	$l+h \leqslant 35$	total: 26400s $l=17$: 17712s	1.103×10^{81} $(l=17, h=18)$
tetrominoes	A230031: $l+h \leqslant 20$ A263425: $l=h \leqslant 10$	$l+h \leqslant 27$	total: 8879s $l=12$: 6494s	8.980×10^{31} $(l=12, h=15)$
pentominoes	A233427: $l+h \leqslant 17$	$l+h \leqslant 26$	total: 16814s $l=11$: 13595s	7.977×10^{30} $(l=11, h=15)$
 + pentominoes	A278657: $l+h \leqslant 15$	$l+h \leqslant 23$	total: 17578s $l=11$: 14482s	4.576×10^{38} $(l=11, h=12)$

Table 4: OEIS sequences related to polyomino tiling enumeration and calculation of our algorithm



The enumeration problems for these variants can also be transformed into similar tensor operations using the techniques of *helicoids* and *Wang decomposition* introduced in this paper, and then computed using modular enumeration for their (weighted) enumeration. The only difference is that these variants introduce an additional directional alphabet, and the rules for index rotation in matrix operations need to be redesigned to ensure that the first few indices always participate in the multiplication. The details are straightforward and can be inferred from the movement of the initial point defining the index order, as shown in the figures below.



Similarly, our algorithm can be generalized and applied to any variant of Wang tiles that exhibits periodicity, as well as the tile sets that can be decomposed into such Wang tiles.

5.2 Tilings with restrictions

In certain tiling enumeration problems, constraints may be imposed on the number of occurrences of specific tiles or require the tilings to possess certain symmetries. We provide two sufficiently general examples corresponding to these two types of constraints, respectively, to demonstrate how our algorithm can be applied to handle these restricted cases.

Example 19. Enumerate the ways to place $\boxminus$ and $\boxplus$ in a $l \times h$ chessboard, such that the total number of tiles is n.

Clearly, the placement of $\boxminus + \boxplus$ in a chessboard corresponds one-to-one with the tiling of $\square + \boxminus + \boxplus$ on the same chessboard, as it suffices to fill all the remaining empty spaces using $\square$. As for the restriction on the number of tiles, it is noteworthy that during tiling in a helicoid, the horizontal characters precisely scan all positions. Therefore, the horizontal characters can be utilized to record information related to the number of tiles.

We use $\{0,1,\ldots,\nu-1\}\times\Theta$ to replace the original horizontal alphabet Θ, where ν is greater than all the values of n that we need to calculate. In this specific case, since each Wang decomposition of $\square$ or $\begin{smallmatrix}\square\\\square\end{smallmatrix}$ has exactly one root tile, the number of tiles on the chessboard is precisely equal to the number of root tiles. Therefore, we perform the following substitution:

$$\alpha\begin{smallmatrix}&a&\\b&&\end{smallmatrix}\beta \mapsto \begin{cases} \displaystyle\sum_{i=0}^{\nu-2} i\alpha\begin{smallmatrix}&a&\\b&&\end{smallmatrix}(i+1)\beta, & \text{root tile;}\\ \displaystyle\sum_{i=0}^{\nu-1} i\alpha\begin{smallmatrix}&a&\\b&&\end{smallmatrix}i\beta, & \text{not root tile.}\end{cases}$$

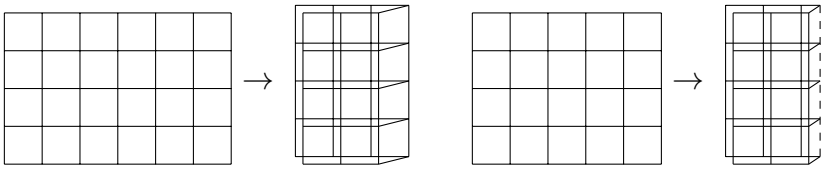
Then, by replacing $\sharp\sharp\ldots\sharp$ with $0\sharp\sharp\ldots\sharp$ as the initial word for tiling, the resulting tensor will encompass the enumeration of all tilings that restrict the number of tiles with $n = 0,1,\ldots,\nu-1$, as the weight of $0\sharp\sharp\ldots\sharp$, $1\sharp\sharp\ldots\sharp$, …, $(\nu-1)\sharp\sharp\ldots\sharp$, precisely corresponding to the leading ν terms of the tensor. Compared to the case without the number restriction, the space complexity increased by a factor of ν in this case.

Furthermore, if the number restriction is modulo ν with remainder n, simply modify the substitution described above as follows:

$$\alpha\begin{smallmatrix}&a&\\b&&\end{smallmatrix}\beta \mapsto \begin{cases} (\nu-1)\alpha\begin{smallmatrix}&a&\\b&&\end{smallmatrix}0\beta + \displaystyle\sum_{i=0}^{\nu-2} i\alpha\begin{smallmatrix}&a&\\b&&\end{smallmatrix}(i+1)\beta, & \text{root tile;}\\ \displaystyle\sum_{i=0}^{\nu-1} i\alpha\begin{smallmatrix}&a&\\b&&\end{smallmatrix}i\beta, & \text{not root tile.}\end{cases}$$

Example 20. Enumerate the ways to place $\square+\begin{smallmatrix}\square\\\square\end{smallmatrix}$ in a $l\times h$ chessboard that the placements are left-right symmetric.

We still regard it as a tiling of $\square+\square+\begin{smallmatrix}\square\\\square\end{smallmatrix}$. To preserve symmetry, we fold the chessboard along its vertical axis, as shown below.



When the width of the chessboard is even, each position after folding consists of two stacked tiles. We restrict these two tiles to originate from corresponding positions in two left-right symmetric polyominoes. That is, We obtain the ($\mathbb{K}$-)tile set by

$$\alpha\begin{smallmatrix}&a&\\b&&\end{smallmatrix}\beta, \alpha'\begin{smallmatrix}&a'&\\b'&&\end{smallmatrix}\beta' \mapsto \alpha\beta'\begin{smallmatrix}&aa'&\\bb'&&\end{smallmatrix}\beta\alpha',$$

where $\alpha\begin{smallmatrix}a\\ \square\\ b\end{smallmatrix}\beta$ and $\alpha'\begin{smallmatrix}a'\\ \square\\ b'\end{smallmatrix}\beta'$ range over all Wang tiles that may appear in the left-right symmetric positions within a single polyomino. Note that such a correspondence is not necessarily one-to-one. For the Wang decomposition of $\square + \boxminus + \boxplus$ (see Example 5), it is

$$\square 2 \leftrightarrow 1\square, \quad \square 1 \leftrightarrow 1\square,$$

$$2\square 1\hookleftarrow, \quad \begin{smallmatrix}\\ \square\\ 2\end{smallmatrix} \hookleftarrow, \quad \begin{smallmatrix}2\\ \square\\ 1\end{smallmatrix} \hookleftarrow, \quad \begin{smallmatrix}1\\ \square\end{smallmatrix} \hookleftarrow, \quad \begin{smallmatrix}\\ \square\\ 1\end{smallmatrix} \hookleftarrow, \quad \square \hookleftarrow .$$

The tiles in the second row are self-corresponding, meaning they may appear in positions within the polyomino that are left-right symmetric to themselves. Thus, the resulting tile set is obtained as follows:

$$\square 21 + 12\square + \square 11 + 11\square$$

$$+21\square 12 + \begin{smallmatrix}\\ \square\\ 22\end{smallmatrix} + \begin{smallmatrix}22\\ \square\\ 11\end{smallmatrix} + \begin{smallmatrix}11\\ \square\end{smallmatrix} + \begin{smallmatrix}\\ \square\\ 11\end{smallmatrix} + \square.$$

We tile these tiles on a chessboard with only half the width, where the right boundary corresponds to the vertical symmetry axis of the original chessboard. Therefore, we no longer restrict the characters on the right boundary to be $\sharp$. Instead, we restrict that the two characters at the same position are identical. That is, perform the following substitution on the tile set for the rightmost column:

$$\alpha\beta'\begin{smallmatrix}aa'\\ \square\\ bb'\end{smallmatrix}\beta\alpha' \mapsto \begin{cases} \alpha\beta'\begin{smallmatrix}aa'\\ \square\\ bb'\end{smallmatrix}, & \beta = \alpha'; \\ \emptyset, & \beta \neq \alpha'. \end{cases}$$

For this example, the resulting tile set is

$$12\square + 11\square + \begin{smallmatrix}\\ \square\\ 22\end{smallmatrix} + \begin{smallmatrix}22\\ \square\\ 11\end{smallmatrix} + \begin{smallmatrix}11\\ \square\end{smallmatrix} + \begin{smallmatrix}\\ \square\\ 11\end{smallmatrix} + 2 \cdot \square.$$

Here the all-blank tile $\square$ carries weight 2: it arises from both the all-blank tile $\square$

and the tile $\square 11$ (with only a right label 11) in the folded set above, which both lose their right labels under the boundary restriction $\beta = \alpha'$ and correspond, respectively, to the empty placement and to a single horizontal domino.

And if the width is odd, we no longer perform the substitution described above. Instead, we add a column on the right, consisting of tiles obtained by horizontally folding the tiles from the original tile set, that is:

$$\alpha \begin{smallmatrix} & a & \\ & \square & \\ & b & \end{smallmatrix} \beta \mapsto \alpha\beta \begin{smallmatrix} & a & \\ & \square & \\ & b & \end{smallmatrix},$$

where $\alpha \begin{smallmatrix} & a & \\ & \square & \\ & b & \end{smallmatrix} \beta$ range over all of those Wang tiles that may appear in positions with left-right symmetry within the polyominoes. For this example, the resulting tile set is

$$21\square + \begin{smallmatrix} & \square & \\ & 2 & \end{smallmatrix} + \begin{smallmatrix} & 2 & \\ & \square & \\ & 1 & \end{smallmatrix} + \begin{smallmatrix} & 1 & \\ & \square & \end{smallmatrix} + \begin{smallmatrix} & \square & \\ & 1 & \end{smallmatrix} + \square.$$

Such restrictions on symmetry do not significantly increase the computational complexity. On the contrary, the computational complexity can likely to be decreased, as the folded tiles may contain fewer pairs of horizontal and vertical characters than the total possible number, while the width is reduced to half or approximately half. In this example, the folded tiles contain the following horizontal and vertical character pairs (i.e., the new alphabets for positions excluding the rightmost column)

$$\hat{\Theta} = \{\sharp, 11, 12, 21\}, \hat{\Sigma} = \{\sharp, 11, 22\},$$

while their sizes are $\hat{\theta} = 4$, $\hat{\sigma} = 3$.

Consequently, for the width l, the size of the state tensor is reduced from $\theta\sigma^l$ to

$$\begin{cases} \hat{\theta}\hat{\sigma}^{\frac{l}{2}}, & 2 \mid l; \\ \hat{\theta}\sigma\hat{\sigma}^{\frac{l-1}{2}}, & 2 \nmid l. \end{cases}$$

It is not difficult to see that this reduction is exponential when $\hat{\sigma} < \sigma^2$.

In addition, one may transpose the chessboard and apply the same method to fold it vertically and perform the enumeration. In this case, the size of the tensor (or, more precisely, the size during the majority of the tiling progress) becomes

$$\hat{\sigma}\hat{\theta}^h.$$

When performing the enumeration, one should choose the approach that results in a smaller tensor size to enhance computational efficiency.

Based on the enumeration with symmetry restriction, by applying Polya's enumeration theorem, we can also compute the tiling enumeration that flip symmetries are not distinguished.

5.3 Open problems

As demonstrated in Section 4, the key to transforming the tiling of polyominoes (or their variants) into Wang tiling lies in the Wang decomposition. This, however, leaves numerous questions pertaining to Wang decomposition unresolved.

As mentioned earlier, the Wang decompositions presented in this paper all rely on manual trial and verification. Verifying that a found decomposition is a correct Wang decomposition is not difficult: one simply enumerates all possible combinations of tiles and compares the resulting shapes and their weights with the polyomino set. However, our methods for producing Wang decompositions for different polyomino sets are not mutually applicable. Finding a general method to produce a Wang decomposition for an arbitrary polyomino set appears to be a rather difficult computational problem.

Furthermore, how to prove the optimality of a given Wang decomposition also remains an open problem. The Wang decompositions presented in this paper seem to admit no further optimization, but how can this be proved theoretically?

Problem 21. How can we find the (optimal) Wang decomposition (if it exists) for a given set of polyominoes (or their variants)?

For a given optimal Wang decomposition, how do we prove that it is optimal?

We have proved that a Wang decomposition always exists at least for polyomino sets of finite area, but the situation becomes much more complicated for the infinite-area case. This naturally leads to the following question:

Problem 22. Under what conditions can an infinite set of polyominoes (or their variants) admit a Wang decomposition?

Moreover, tiling enumeration problems that are too large still remain beyond the computational capacity of our algorithm. Consequently, we are compelled to inquire:

Problem 23. Is there potential to further simplify computations for certain special tiling enumeration problems by leveraging the properties of tensors?

Problem 24. Is it possible to derive specific form of the enumeration formulas (or prove their impossibility) based on the tensor operations we defined?

For tilings of excessively large scales, we can also provide estimations. From the computational results, it can be observed that when $l, h \to \infty$, the growth trend of most of the enumeration of tilings within $l \times h$ chessboards approximates to

$$N(l,h) \sim \lambda^{lh},$$

where λ is called the *entropy constant* [25, 26]; or, satisfy such approximation periodically. For instance, in the case of $\square + \boxplus$ tiling enumeration, we can provide the approximation:

$$N(l,h) \sim \begin{cases} \lambda^{lh}, & 2 \mid lh; \\ 0, & 2 \nmid lh, \end{cases}$$

and $\lambda \approx 1.46$, a rough estimate of the constant based on existing results.

Hence, we are prompted to inquire:

Problem 25. Can a universal method be found to provide approximations for large-scale cases of polyomino tiling enumeration?

When an approximate formula is found, is there a way to obtain the exact value of the constants appearing in it (such as the entropy constant)? If not, how can we further improve its estimation?

References

[1] T. Lai. Problems in the enumeration of tilings. arXiv:2109.01466, 2021.

[2] P. A. MacMahon. *Combinatory analysis*, Volumes I and II. Chelsea Publishing Co., New York, 1960.

[3] J. Propp. Enumeration of matchings: problems and progress. In *New perspectives in algebraic combinatorics*, volume 38 of *Math. Sci. Res. Inst. Publ.*, pages 255–291. Cambridge Univ. Press, Cambridge, 1999.

[4] P. W. Kasteleyn. The statistics of dimers on a lattice. I. The number of dimer arrangements on a quadratic lattice. *Physica*, 27(12): 1209–1225, 1961.

[5] H. N. V. Temperley and M. E. Fisher. Dimer problem in statistical mechanics—an exact result. *Philos. Mag. (8)*, 6(68): 1061–1063, 1961.

[6] P. W. Kasteleyn. Dimer statistics and phase transitions. *J. Math. Phys.*, 4(2): 287–293, 1963.

[7] D. Klarner and J. Pollack. Domino tilings of rectangles with fixed width. *Discrete Math.*, 32(1): 45–52, 1980.

[8] H. Cohn, R. Kenyon, and J. Propp. A variational principle for domino tilings. *J. Amer. Math. Soc.*, 14(2): 297–346, 2001.

[9] P. Di Francesco, P. Zinn-Justin, and J.-B. Zuber. Determinant formulae for some tiling problems and application to fully packed loops. *Ann. Inst. Fourier (Grenoble)*, 55(6): 2025–2050, 2005.

[10] I. Fischer. Computations versus bijections for tiling enumeration. *Adv. in Appl. Math.*, 142: 102427, 2023. doi:10.1016/j.aam.2022.102427

[11] B. S. Chlebus. Domino-tiling games. *J. Comput. System Sci.*, 32(3): 374–392, 1986.

[12] M. Ebbesen, P. Fischer, and C. Witt. Edge-matching problems with rotations. In *Fundamentals of computation theory*, volume 6914 of *Lecture Notes in Comput. Sci.*, pages 114–125. Springer, Berlin, 2011. doi:10.1007/978-3-642-22953-4_10

[13] H. Wang. Proving theorems by pattern recognition—II. *Bell System Tech. J.*, 40(1): 1–41, 1961.

[14] R. Berger. The undecidability of the domino problem. *Mem. Amer. Math. Soc.*, 66, 1966.

[15] C. Moore. Some polyomino tilings of the plane. arXiv:math/9905012, 1999.

[16] G. Barequet, M. Moffie, A. Ribó, and G. Rote. Counting polyominoes on twisted cylinders. In *Discrete models for complex systems (DMCS'03)*, volume AE of *Discrete Math. Theor. Comput. Sci. Proc.*, pages 369–374. 2005.

[17] P. Chinn, R. Grimaldi, and S. Heubach. Tiling with L's and squares. *J. Integer Seq.*, 10(2): Article 07.2.8, 2007.

[18] E. D. Demaine and M. L. Demaine. Jigsaw puzzles, edge matching, and polyomino packing: connections and complexity. *Graphs Combin.*, 23: 195–208, 2007. doi:10.1007/s00373-007-0713-4

[19] É. Janvresse, T. de la Rue, and Y. Velenik. A note on domino shuffling. *Electron. J. Combin.*, 13(1): #R30, 2006.

[20] H. Fukuda, N. Mutoh, G. Nakamura, and D. Schattschneider. Enumeration of polyominoes, polyiamonds and polyhexes for isohedral tilings with rotational symmetry. In *Computational geometry and graph theory*, volume 4535 of *Lecture Notes in Comput. Sci.*, pages 68–78. Springer, Berlin, 2008.

[21] E. Nordenstam. On the shuffling algorithm for domino tilings. *Electron. J. Probab.*, 15: 75–95, 2010.

[22] S. Desreux and É. Rémila. An optimal algorithm to generate tilings. *J. Discrete Algorithms*, 4(1): 168–180, 2006. doi:10.1016/j.jda.2005.01.003

[23] S.-J. Ran, E. Tirrito, C. Peng, X. Chen, L. Tagliacozzo, G. Su, and M. Lewenstein. *Tensor network contractions: methods and applications to quantum many-body systems*. Lecture Notes in Physics 964. Springer, Cham, 2020. doi:10.1007/978-3-030-34489-4

[24] J. D. Biamonte, J. Morton, and J. Turner. Tensor network contractions for #SAT. *J. Stat. Phys.*, 160(5): 1389–1404, 2015.

[25] S. Forchhammer and J. Justesen. Entropy bounds for constrained two-dimensional fields. *IEEE Trans. Inform. Theory*, 45(1): 118–127, 1999.

[26] S. Friedland and U. N. Peled. Theory of computation of multidimensional entropy with an application to the monomer-dimer problem. *Adv. in Appl. Math.*, 34(3): 486–522, 2005. doi:10.1016/j.aam.2004.08.005