J. Algebra Comb. Discrete Appl.
13(1) • 15–27

Received: 19 June 2024
Accepted: 23 June 2025

Journal of Algebra Combinatorics Discrete Structures and Applications

An algorithm for counting domino tilings of a rectangular chessboard

Research Article

Abdulkarim M. Magomedov*, Serge A. Lawrence

Abstract: A recursive method is developed for counting domino tilings of a rectangular chessboard (the dimer problem). Based on this method, a new and enhanced recursive algorithm is proposed for solving this problem. Close connections with Fibonacci numbers are traced out.

2020 MSC: 52C20, 05B45, 11B39

Keywords: Domino tiling, Tiling counting, Dimer problem, Recurrence relation, Algorithm, Fibonacci numbers

1. Introduction

By domino tiles we mean 1×2 rectangles or shapes formed by the union of two unit-squares meeting edge-to-edge. Let $M = M(n \times k)$ be a rectangular $n \times k$ chessboard with integer side lengths n (height) and k (width). A domino tiling of M is a tessellation of M with domino tiles, in which the tiles can be laid either horizontally or vertically without gaps or overlaps. If n and k are both odd, then no domino tiling of $M(n \times k)$ exists, so we hereafter assume for certainty that k is even. This paper is mainly devoted to the following problem.

Domino Tiling Counting Problem (Dimer Problem, Gaps Not Allowed). Count the total number, $\tau(n, k)$, of domino tilings of $M = M(n \times k)$ *without gaps*.

* *The author is supported by the Department of Mathematics and Informatics, Dagestan Federal Research Center of the Russian Academy of Sciences.*

Abdulkarim M. Magomedov; Department of Discrete Mathematics and Informatics, Dagestan State University, and Department of Mathematics and Informatics, Dagestan Federal Research Center of the Russian Academy of Sciences, 45 M. Gadzhiev, Makhachkala, 367032, Dagestan, Russia, (email: magomedtagir1@yandex.ru).

Serge A. Lawrence (Corresponding Author); Department of Cryptology, Faculty of Mechanics and Mathematics, L.N. Gumilyov Eurasian National University, 2 Satbaev, Astana, 10000, Kazakhstan, and Institute of Service Technologies, Russian State University of Tourism and Service, Podolsk, Russia, (email: serge.a.lawrence@gmail.com).

The domino tiling counting problem is equivalent to the problem of counting the number of perfect matchings in a plane $n \times k$ grid graph, since each domino tiling of $M(n \times k)$ corresponds to a perfect matching in the grid graph. The latter problem arises in applied problems of physics and chemistry: Some classes of chemical compounds are easily synthesized only when the graphs of the compounds (modeling the molecules of the compounds) have a perfect matching; moreover, the more perfect matchings in the graph, the more stable the compound [7]. We refer the interested readers to the references [1–6, 8–14] dealing with more aspects of the domino tiling counting problem.

The domino tiling counting problem is also known as the dimer problem [11] and is known to be difficult (see [10] for details). This problem was solved by Kasteleyn [5] and by Temperley and Fisher [13] in the form of the following cosine formula for the number $\tau(n, k)$:

$$\tau(n,k) = 2^{\frac{nk}{2}} \prod_{i=1}^{n} \prod_{j=1}^{k} \left(\cos^2 \frac{\pi i}{n+1} + \cos^2 \frac{\pi j}{k+1} \right)^{\frac{1}{4}}.$$

However, floating-point operations make it difficult to calculate $\tau(n, k)$ accurately, using this cosine formula. For example, for the chessboard $M(16 \times 8)$, the result of calculations using the cosine formula, performed by a $C\#$ program using double-precision floating-point real variables, is 540061286536919, while the true value of $M(16 \times 8)$ is 540061286536921 (see [8]). This simple example shows unsuitability of the cosine formula for exact calculations, and this motivated us to design an algorithm which uses only integers.

The main purpose of the current paper is to propose an algorithmic solution which uses an enhanced all-integer recursive algorithm (Algorithm 2, Section 6) for counting the number $\tau(n, k)$, using only integers instead of a calculation involving real numbers. The key for the enhancement is our subroutine called *Generating a complete list of weakly consistent profiles* (Algorithm 1, Section 5). Our enhanced algorithm is more computationally efficient than known algorithmic solutions to the problem; see Section 7.

Our recursive method is based on a folklore dating back to Ahrens [1]. Our method has some resemblance with Ahrens' method, yet differs from it by our further developed profile technique (Section 3) and by better computational efficiency (Section 7).

2. Motivating problem: Ahrens' result [1]

The following is a problem which has motivated the current study. Unlike the domino tiling counting problem stated in Introduction, the following problem allows gaps between the tiles. Gaps between the tiles are allowed in this section only!

Strip Dimer Problem (Gaps Allowed). Assume that the chessboard is a $1 \times k$ strip of squares (k even). In how many ways can the strip be tiled, but with the modification that not all squares of the strip are necessarily covered with dominoes?

Given a bit string, a maximal substring entirely consisting of 0s, resp. 1s, is called a 0-*series*, resp. 1-*series*. An even, resp. odd, (0- or 1-) series is a series of even, resp. odd, length. A bit string is 0-*valid* if it contains no odd 0-series and is 1-*valid* if it contains no odd 1-series.

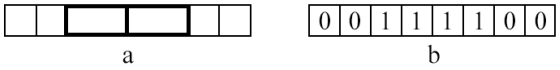

Figure 1. Domino tiling (a) and its binary code (b).

To solve the strip dimer problem, we encode domino tilings of a $1 \times k$ strip of squares (k even) with k-bit strings (that is, bit strings of length k): If a square of the strip is covered with a domino tile, we

label it with a 1, otherwise with a 0. Then domino tilings of the $1 \times k$ strip correspond to 1-valid k-bit strings. Now the strip dimer problem can be rephrased as follows: *How many* 1*-valid* k*-bit strings are there?*

An example of a domino tiling is shown in Figure 1(a), where $k = 8$. The squares of the strip are numbered from left to right, starting from 0. While squares 2, 3, 4, 5 are covered with dominoes, squares 0, 1, 6, 7 are left uncovered.

The following is the recurrence relation for Fibonacci numbers:

$$F(0) = 0, \quad F(1) = 1, \qquad F(m) = F(m-1) + F(m-2) \qquad (m \geqslant 2). \tag{1}$$

The Fibonacci numbers have a closed-form expression known as Binet's formula:

$$F(m) = \frac{\varphi^m - \psi^m}{\sqrt{5}}, \tag{2}$$

where $\varphi = \frac{1+\sqrt{5}}{2}$ and $\psi = \frac{1-\sqrt{5}}{2}$. Various connections of Fibonacci numbers with domino tilings have been traced out by a number of authors; see [1–3, 11, 12]; for example, recall the elementary school problem of counting domino tilings of an $n \times 2$ board without gaps [12, p. 75]. The following is another folklore result, which is proved in Section 4 and is used in Section 5.

Lemma 2.1. *The total number of* 1*-valid* k*-bit strings, which is clearly equal to the total number of* 0*-valid* k*-bit strings, is equal to the Fibonacci number* $F(k+1)$*.*

Corollary 2.2 (Ahrens [1])**.** *The number of ways to cover a* $1 \times k$ *strip of squares with domino tiles, possibly with gaps between the tiles, is equal to the Fibonacci number* $F(k+1)$*.*

Proof. The corollary is indeed a consequence of Lemma 2.1 due to the fact that any covering of a $1 \times k$ strip of squares with domino tiles (possibly with gaps) corresponds to a 1-valid k-bit string, and converse. □

3. Recursive method: Profile technique

From now on, we assume that no gaps are allowed between the tiles! In this section we describe a recursive method for solving the domino tiling counting problem stated in the Introduction.

We number the columns of the chessboard $M = M(n \times k)$ from left to right—0, 1, …, $k-1$—and the rows from bottom to top—1, …, n. Denote by (i, j) the square that is in row i and column j. Denote by $(i, j)-(i, j+1)$, resp. $(i, j)-(i+1, j)$, the horizontal, resp. vertical, domino tile (if any) that covers the square (i, j) and directs rightward, resp. upward.

Given a domino tiling of M, we label the tiles with numerals 1, 2, …, $\frac{kn}{2}$ (k even) so that the tiles closer to the base of M are labeled with smaller numerals. We follow the *chronological tiling rule*, that is, each tile is laid at (discrete) time instant equal to the numeral label of the tile. At each time instant, the current state of row i ($i = 1, \ldots, n$) is described by a k-bit string in which the bit in position j is 1 if square (i, j) is covered with a tile and is 0 otherwise ($j = 0, 1, \ldots, k-1$).

Let t_i be the smallest time instant at which all the squares of row $i-1$ get covered with tiles. The whole process of tiling row i splits into two successive processes as follows:

1) *pre-active tiling* of the row i is underway during the period $(t_{i-1}, t_i]$,

2) *active tiling* of the row i is underway during the period $(t_i, t_{i+1}]$ and consists of successively laying either a horizontal tile $(i, j)-(i, j+1)$ or a vertical tile $(i, j)-(i+1, j)$.

The chronological tiling rule ensures that the process of active tiling a row begins as soon as all the squares of the preceding (lower) row are already covered with tiles.

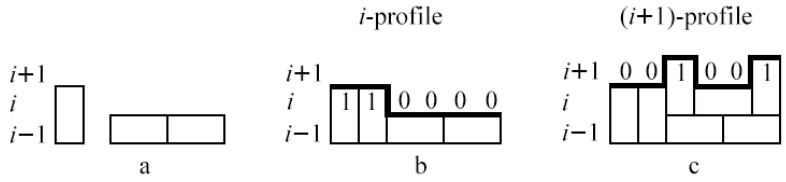

Figure 2. No i-profile has been produced yet at the time instant shown in panel (a) because an uncovered square remains in row $i-1$; panel (b) shows $\mathrm{pr}(i) = 110000$; panel (c) shows $\mathrm{pr}(i+1) = 001001$.

During a chronological tiling process, the bit string formed in row i at time t_i is called an *i-profile* and is denoted by $\mathrm{pr}(i)$ where $i = 0, 1, \ldots, n+1$. Observe that the whole chessboard has to be tiled up at time t_{n+1} and therefore the following are the initial and final profiles: $\mathrm{pr}(0) = \mathrm{pr}(n+1) = 0\ldots0$ (k zeros). Since the i-profile is tied to specific time t_i, any changes in row i do not alter $\mathrm{pr}(i)$ after time t_i. Figure 2 presents an example of changing from $\mathrm{pr}(i)$ in Figure 2(b) to $\mathrm{pr}(i+1)$ in Figure 2(c) by laying two vertical and one horizontal tiles. Notice that the bitwise boolean product of any two consecutive profiles is $0\ldots0$ (k zeros); for example, $\mathrm{pr}(i) \cdot \mathrm{pr}(i+1) = 110000 \cdot 001001 = 000000$ (where $1 \cdot 1 = 1$, $1 \cdot 0 = 0$, $0 \cdot 1 = 0$, and $0 \cdot 0 = 0$). In general, we say that a *profile q is strongly consistent with a profile p*, and write $q \sim_s p$, provided there exists a specific chronological process of tiling M so that $p = \mathrm{pr}(i)$ and $q = \mathrm{pr}(i+1)$ for some $i \in \{0, \ldots, n\}$.

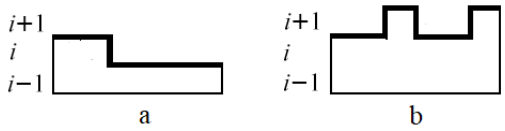

Figure 3. (a) $\widetilde{M}(i, \mathrm{pr}(i))$, (b) $\widetilde{M}(i+1, \mathrm{pr}(i+1))$.

Consider the process of chronological tiling the chessboard M in general. At time t_i, the profile $\mathrm{pr}(i)$ has just been formed. Let $\widetilde{M}(i, \mathrm{pr}(i))$ be a board that is the part of M that only contains the squares covered by the time t_i; these notion and notation are illustrated by Figures 3(a) and 3(b) which correspond to Figures 2(b) and 2(c), respectively. Note that $\widetilde{M}(i, \mathrm{pr}(i))$ is regarded as just a part of the bord M but not as any specific tiling of it. For example, $\widetilde{M}(i, \mathrm{pr}(i)) = \widetilde{M}(i, 110000)$, or $\widetilde{M}(i, 48)$ in decimal notation, is the part of the board M under the bold rectilinear path, shown in Figure 3(a). Furthermore, $\widetilde{M}(i+1,\, \mathrm{pr}(i+1)) = \widetilde{M}(i+1,\, 001001) = \widetilde{M}(i+1,\, 9)$ is the part of M under the bold rectilinear path, shown in Figure 3(b). A domino tiling of $\widetilde{M}(i,\, \mathrm{pr}(i))$ is a tessellation of this board with domino tiles, in which the tiles can be laid either horizontally or vertically without gaps or overlaps.

The essence of our approach is to traverse through the chessboard M from bottom to top, chronologically expanding the tiled area by laying tiles one by one without missing any tiling option. Note that each domino tiling of $\widetilde{M}(i,\, \mathrm{pr}(i))$ is uniquely expanded (by additionally laying out necessary tiles) to a tiling of $\widetilde{M}(i+1, q)$ for any $(i+1)$-profile q strongly consistent with $\mathrm{pr}(i)$. For a given profile q, we need to identify profiles $\{p\}$ such that $q \sim_s p$. Then the number of all domino tilings of $\widetilde{M}(i+1,\, q)$ is calculated by summing the numbers of tilings of $\widetilde{M}(i, p)$ over all profiles p such that $q \sim_s p$. Letting $\widetilde{\tau}(i, p)$ be the total number of domino tilings of $\widetilde{M}(i, p)$, we summarize the discussion in the following recurrence formula which is to be applied repeatedly for $i = 0, \ldots, n$:

$$\widetilde{\tau}(i+1, q) = \sum_{p \mid\, q \sim_s p} \widetilde{\tau}(i, p), \tag{3}$$

where $\widetilde{\tau}(0, p) = 1$ if $p = 0$, and $\widetilde{\tau}(0, p) = 0$ otherwise. The desired value of $\tau(n, k)$ is found as $\widetilde{\tau}(n+1, 0)$. To complete the description of the recursive method, we need a criterion for checking profile consistency.

The boolean operator $\oplus$ is defined by $1 \oplus 1 = 0$, $1 \oplus 0 = 1$, $0 \oplus 1 = 1$, $0 \oplus 0 = 0$. Denote by p_j and q_j the bits in position j of k-bit strings p and q, respectively, $j \in \{0, \ldots, k-1\}$.

Theorem 3.1 (Criterion for Strong Consistency of Profiles)**.** *Let p be a k-bit string that appears as a profile $p = \mathrm{pr}(i)$ in some chronological tiling of M, where $0 \leqslant i \leqslant n$. A k-bit string q has the property that $q \sim_s p$ if and only if the following two conditions are satisfied:*

(a) $p \cdot q = 0 \ldots 0$ (k zeros), (b) $p \oplus q$ is a 0-valid bit string.

Proof. *Necessity.* Notice that $q_j = 1$ is possible only as a result of laying a vertical tile $(i, j){-}(i+1, j)$, which is possible only in case $p_j = 0$. Also notice that if $p_j = 1$, then $q_j = 0$. Thus, the following two implications are valid: $(q_j = 1 \Rightarrow p_j = 0)$ and $(p_j = 1 \Rightarrow q_j = 0)$, whence condition (a) is satisfied. Now, since (a) holds, $(p \oplus q)_j = 0$ is equivalent to the simultaneous equations $p_j = 0$ and $q_j = 0$. The equation $p_j = 0$ means that square (i, j) was not tiled during the pre-active period of tiling of row i so that (i, j) was covered with a tile during the process of active tiling row i and moreover, that tile is necessarily horizontal because $q_j = 0$. It follows that $p \oplus q$ is 0-valid.

Sufficiency. Assume conditions (a) and (b) are satisfied. We need to prove that there exists a chronological tiling of the chessboard M with $p = \mathrm{pr}(i)$ and $q = \mathrm{pr}(i+1)$. To do this, partition the set of positions $j \in \{0, \ldots, k-1\}$ into three disjoint subsets as follows:

$$\begin{aligned} A &= \{j \mid p_j = 0 \wedge q_j = 1\}, \\ B &= \{j \mid p_j = 1 \wedge q_j = 0\}, \\ C &= \{j \mid p_j = 0 \wedge q_j = 0\}. \end{aligned} \tag{4}$$

The event $p_j = 1 \wedge q_j = 1$ cannot occur by condition (a) of the theorem. Pick a chronological tiling of M with $\mathrm{pr}(i) = p$ at time t_i ($i \in \{0, \ldots, n\}$). To extend the tiling upward so as to have q as $(i+1)$-profile, cover each square (i, j) of row i by a vertical tile $(i, j){-}(i+1, j)$ if $j \in A$ and keep $(i+1, j)$ uncovered if $j \in B$. Now, by condition (b), the set C of not-yet-covered squares of row i constitutes the union of even 0-series of $p \oplus q$. Thus, successively laying horizontal tiles covering C (in the process of active tiling row i) is the only way to keep zero values in the corresponding squares of row $(i+1)$, so that the extension is unique. Clearly, the $(i+1)$-profile of the resulting tiling coincides with the given bit string q, and so that $q \sim_s p$. □

Thanks to Theorem 3.1, we have to perform $p \oplus q$ only in case $p \cdot q = 0 \ldots 0$ (k zeros), in which case $p \oplus q$ can be effectively replaced by the decimal summation $p + q$ followed by converting the sum back to binary. For example, for the two consecutive profiles in Figure 2, we have $\mathrm{pr}(i) \oplus \mathrm{pr}(i+1) = 110000 \oplus 001001 = 48 + 9 = 57 = 111001$.

Note that it is not true that all k-bit strings can occur as real profiles; for instance, for k even, no k-bit strings with an odd number of 1s can ever occur as real profiles. Nonetheless, it will be convenient to regard such "unreal profiles" as *fake profiles*. The *weak consistency* of (possibly fake) profiles p and q is defined by

$$c_{pq} = \begin{cases} 1 & \text{if both conditions (a) and (b) of Theorem 3.1 hold,} \\ 0 & \text{otherwise.} \end{cases}$$

If $c_{pq} = 1$, we write $q \sim_w p$ and say that q and p are weakly consistent; clearly, the relation $\sim_w$ is symmetric.

Corollary 3.2. *If $q \sim_w p$, then $q \sim_s p$ unless the profile p is fake.*

Proof. Assume that $q \sim_w p$ and that p is not fake. Then, by Theorem 3.1, $q \sim_w p$ is equivalent to $q \sim_s p$, which completes the proof. □

Now we can rewrite equation (3) as follows:

$$\widetilde{\tau}(i+1,q) = \sum_{p=0}^{2^k-1} c_{pq}\,\widetilde{\tau}(i,p), \tag{5}$$

where for each $i = 0,\ldots,n$, the bit string q runs through $0,\ldots,2^k-1$ (in decimal notation) and where $\widetilde{\tau}(1,0) = 1,\ \widetilde{\tau}(1,1) = \cdots = \widetilde{\tau}(1,2^k-1) = 0$. The desired value of $\tau(n,k)$ is found as $\widetilde{\tau}(n+1,0)$.

The following are five important points to be aware of concerning equation (5):

• Equation $\widetilde{\tau}(1,0) = 1$ corresponds to the vacuous tiling and is technical;

• although profiles are defined to be k-bit strings, they may be identified with their decimal representations,

• a disadvantage of changing to equation (5), instead of equation (3), is that we acquire fake profiles, but a big advantage is that it is much easier to check the weak consistency than strong consistency of (possibly fake) profiles;

• if a current bit string q is a fake profile, all profiles weakly consistent with q are also fake, so that the corresponding terms of the sum in equation (5) vanish;

• our recursive method is based on equation (5) and has a computational advantage by using only integers.

Another disadvantage of the recursive method (as described in this section) is that we have to check out all k-bit strings on weak consistency with a current (possibly fake) profile. In Section 5 we compensate for this disadvantage by beforehand preparing lists of profiles weakly consistent with a given one.

4. Building a list of all 0-valid bit strings of a given length

Let s be a bit string in $\{1, 0\,0\}$ and let r be an arbitrary 0-valid bit string. The concatenation of s to r is denoted $s^\frown r$; for example, $1^\frown 1\,1\,0\,0 = 1\,1\,1\,0\,0$. Let $\{D_i\}$ $(i = 1,2,\ldots)$ be an *array of arrays*, that is, ith element D_i is a list of all 0-valid bit strings of length i. For each i, the list D_i splits into two parts so that the bit strings in one part have the prefix 1 while those in the other have the prefix $0\,0$. If the prefix 1 is deleted in all bit strings of the former part and the prefix $0\,0$ is deleted in those of the latter part, then the former part changes to the list D_{i-1} while the latter part becomes the list D_{i-2}.

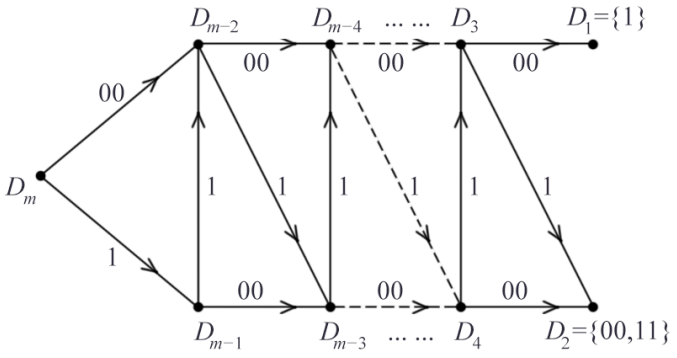

Figure 4. For an arbitrarily fixed m, we repeatedly split D_i $(i = m, m-1,\ldots,3)$ into D_{i-1} and D_{i-2} by deleting the prefixes, which recursive process ultimately enables us to evaluate $|D_m|$.

Figure 4 shows a digraph with m nodes, in which the nodes represent the lists D_i $(i = m,\ldots,1)$ and in which the arcs of the forms $(D_i,\ D_{i-1})$ and $(D_i,\ D_{i-2})$ are labeled by 1 and $0\,0$, respectively. Consider

the following sequence of the sizes $|D_i|$:

$$|D_1| = 1, \quad |D_2| = 2, \quad |D_i| = |D_{i-1}| + |D_{i-2}| \text{ for } i = 3, 4, \dots .$$

Thus, $|D_m| = F(m+1)$ (cf. equation (1)), which provides a proof of Lemma 2.1.

Given below are the first five successive lists D_m, where $s^\frown D_m$ stands for the list formed by concatenating the string s to each bit string in D_m:

$$D_1 = \{1\}, \quad D_2 = \{00, 11\}, \quad D_3 = (1^\frown D_2) \cup (00^\frown D_1) = \{100, 111, 001\},$$
$$D_4 = (1^\frown D_3) \cup (00^\frown D_2) = \{1100, 1111, 1001, 0000, 0011\}, \; D_5 = (1^\frown D_4) \cup (00^\frown D_3) =$$
$$= \{11100, 11111, 11001, 10000, 10011, 00100, 00111, 00001\}, \dots .$$

The above-described method is formalized below in the form of pseudocode and will be referred to as *GVS procedure*, where GVS stands for "generating valid strings".

procedure GVS$(k, \{D_i\})$

Input: k: even positive integer

$D_1 := \{1\}$

$D_2 := \{00, 11\}$

Output: $\{D_i\}$: lists of all 0-valid bit strings of lengths i $(i = 1, 2, \dots, k)$

for $i := 3$ **to** k **do**

$D(i) := (1^\frown D_{i-1}) \cup (00^\frown D_{i-2})$

endfor

return $\{D_i\}$

end

Since the procedure GVS only uses the operations of string concatenation and merging two lists of strings, it follows from the folklore formula,

$$F(1) + F(2) + \cdots + F(k) = F(k+2) - 1,$$

that the procedure GVS creates $F(k+2) - 1$ bit strings, and thus, by equation (2), we come to the following:

Lemma 4.1. *The procedure GVS generates all* 0*-valid bit strings of lengths at most* k*. Its computational complexity is* $O(\varphi^k)$.

5. Generating a complete list of weakly consistent profiles

To be able to find all (possibly fake) profiles weakly consistent with a given one (also possibly fake), we developed a computer code, named *Generating a complete list of weakly consistent profiles*, using *C#* programming language. Below is a description of the algorithm and a justification (Theorem 5.1) that the algorithm gives the correct answer.

To construct a (possibly fake) profile q weakly consistent with a given (possibly fake) profile p, we firstly put 0s in all positions j of q whenever $p_j = 1$. Then we identify all 0-series of p and insert all possible 0-valid bit strings in the positions of q corresponding to those 0-series in p and successively output the so-generated bit strings $\{q\}$ as profiles weakly consistent with p.

Algorithm 1 *Generating a complete list of weakly consistent profiles*

Input: k: even positive integer {the width of the chessboard M}

p: k-bit profile (possibly fake) {a bit string of length k}

Output: $W_k(p)$ {a complete list $\{q\}$ of all profiles weakly consistent with p}

Step 1: If the profile p is a single 1-series, then the only consistent profile is $q = 0\ldots0$ (k zeros); in such event, add q to the list $W_k(p)$ and terminate the algorithm. Otherwise, denote by $r = r(p)$ the total number of 0-series of p and, for each 0-series, calculate the range P_i ($i = 0, \ldots, r-1$) of its bits' positions in the string p, and determine the length, $\ell = \ell(p)$, of the longest of those 0-series.

Step 2: Using the procedure GVS$(\ell, \{D_i\})$, generate a complete list of 0-valid bit strings with length at most ℓ. Then generate complete lists R_i of all 0-valid bit strings that fit the ranges P_i, respectively.

Step 3: For each tuple $(t_0, \ldots, t_{r-1})$ of bit strings such that $t_0 \in R_0, \ldots, t_{r-1} \in R_{r-1}$, prepare an all-zero bit string $0\ldots0$ (k zeros) and insert each t_i into the positions of the all-zero bit string, corresponding to the positions of P_i in p, and then add the whole string obtained to the list $W_k(p)$ as a newly generated profile q weakly consistent with p.

End of Algorithm 1

Here is an example to illustrate the performance of Algorithm 1 for $k = 10$. Given profile $p = 1100010000$ (in fact, this profile is fake, which is not essential for the algorithm) which has two 0-series (so $r = 2$) of lengths at most $\ell = 4$, we show how Algorithm 1 generates a complete list of weakly consistent profiles $\{q\}$. By calling the subroutine GVS$(\ell, \{D_i\})$, the following lists of 0-valid bit strings are generated:

$$D_3 = R_0 = \{100, 111, 001\}, \qquad D_4 = R_1 = \{1100, 1111, 1001, 0000, 0011\}.$$

To generate all (possibly fake) profiles $\{q\}$ consistent with p, the algorithm repeatedly inserts all pairs of bit strings $s_0 \in R_0$, $s_1 \in R_1$ in the positions of P_0 and P_1 (respectively) of the string $0\ldots0$ (10 zeros), adding all so generated bit strings to the cumulative list $W_{10}(1100010000)$. The complete list is summarized in Table 1.

Table 1. List $W_{10}(1100010000)$ of all profiles $\{q\}$ consistent with the profile $p = 1100010000$.

0010001100	0011101100	0000101100
0010001111	0011101111	0000101111
0010001001	0011101001	0000101001
0010000000	0011100000	0000100000
0010000011	0011100011	0000100011

Theorem 5.1 (Justification of Algorithm 1)**.** *i) Given a k-bit (k even) profile p (possibly fake), Algorithm 1 generates a complete list $W_k(p)$ of weakly consistent with p profiles $\{q\}$,*

ii) $|W_k(p)| = F(|s_0|+1)\cdot\ldots\cdot F(|s_{r-1}|+1)$, where $F(\cdot)$ are Fibonacci numbers defined by equation (1), or (2), and $s_0, \ldots, s_{r-1}$ are all 0-series of p, and $|s_m|$ is the length of s_m.

Proof. To prove part (i), we need to show that *each* consistent with p profile q is generated by Algorithm 1. Let q be an arbitrary profile consistent with the given profile p. Recall (Section 3) that two profiles (possibly fake) are called weakly consistent provided that both conditions (a) and (b) of Theorem 3.1 are met. By condition (a), $q_j = 0$ whenever $p_j = 1$, and the substrings of q in between those 0s are all necessarily 0-valid by condition (b), so that the statement follows.

Part (ii) is a direct consequence of Lemma 2.1. □

Theorem 5.2. *The computational complexity of Algorithm 1 is $O(\varphi^k)$.*

Proof. The statement follows from Lemma 4.1. □

6. Main result: Enhanced algorithm

The crucial motivation for enhancing our original recursive method (Section 3) is that by equation (5) only two rows, i and $i+1$, of the 2-dimensional $(n+1)\times 2^k$ array $\widetilde{\tau}$ are in use at each time t_i. This enables us to exploit two 1-dimensional arrays, α and β, given by $\alpha(p) = \widetilde{\tau}(i,p)$, $\beta(q) = \widetilde{\tau}(i+1,q)$, instead of one large 2-dimensional array $\widetilde{\tau}$. Now equation (5) takes the following simple form:

$$\beta(q) = \sum_{p\in W_k(q)} \alpha(p). \tag{6}$$

Our enhanced algorithm (Algorithm 2 below) processes profiles in decimal notation and calls the subroutine *Generating a complete list of weakly consistent profiles* (Algorithm 1, Section 5).

Algorithm 2 *Calculating the total number of domino tilings*

Input: k: even positive integer {the width of the chessboard M}

n: positive integer {the height of M}

Output: $\tau = \tau(n,k)$: the total number of ways to tile M with dominoes.

Step 1: For each k-bit string p running through $0, 1, \ldots, 2^k-1$ (in decimal notation), call the subroutine *Generating a complete list of weakly consistent profiles* (k, p), which forms the list $W_k(p)$.

Step 2: Assign initial values to the entries of β as follows:

$$\beta(0) = 1,\ \beta(1) = \cdots = \beta(2^k-1) = 0.$$

Step 3: Iterate n times over the following two steps:

Step 3.1: Copy the array β to array α.

Step 3.2: Apply equation (6) for all (possibly fake) profiles q in $\{0, 1, \ldots, 2^k-1\}$.

Return the current value $\beta(0)$ which is in fact equal to $\tau(n,k)$.

End of Algorithm 2

$$C_4 = \begin{bmatrix}
1&0&0&1&0&0&0&0&0&1&0&0&1&0&0&1\\
0&0&1&0&0&0&0&0&1&0&0&0&0&0&1&0\\
0&1&0&0&0&0&0&0&0&0&0&0&0&1&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&1&0&0&1&0&0&1&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0
\end{bmatrix} \tag{7}$$

Here is an example to illustrate the performance of Algorithm 2 in the specific case $k = n = 4$. To perform Step 1 of Algorithm 2, we use Algorithm 1 (Section 5). The result can be seen from the

symmetric matrix $C_k = C_4$ given by equation (7) and defined to be a $2^k \times 2^k = 2^4 \times 2^4$ zero–one matrix $C_k = [c_{pq}]$ (see Section 3 for the notation). To generate the list $W_k(q) = W_4(q)$ for each profile $q = 0, 1, \ldots, 15$, traverse row q of the matrix C_4 from left to right: The profiles weakly consistent with q are easily identified as those corresponding to the 1 entries in that row, where profiles are presented in decimal notation. For example, $W_4(0)$ consists of the profiles 0, 3, 9, 12, 15; these correspond to the 1 entries in row 0 of C_4 (see equation (7)).

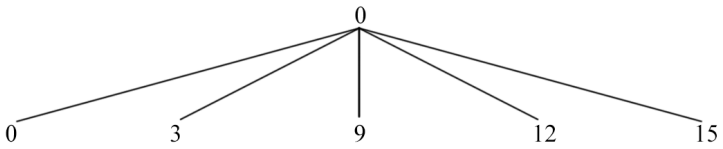

Figure 5. The output of the first iteration of Algorithm 2.

Since $W_4(0) = \{0, 3, 9, 12, 15\}$, at Step 3, after the first iteration, we have

$$\beta(0) = \beta(3) = \beta(9) = \beta(12) = \beta(15) = 1,$$

and $\beta(i) = 0$ for all other i; this is recorded graphically by the diagram in Figure 5.

We represent the successive iterations of Step 3 as levels of a node-labeled tree, in which node labels represent profiles and two nodes are adjacent provided their labels correspond to (weakly) consistent profiles. The root of the tree is at level 0, and the levels of the tree correspond to the iterations of the algorithm. All children of profile q are profiles consistent with q.

Table 2. Short form of recording the results of the second iteration. The upper row shows profiles. The lower row shows the multiplicities of occurrences of the corresponding profiles at level 2; $\beta(0) = 5$.

i	0	3	9	12	15	6
$\beta(i)$	5	2	1	2	1	1

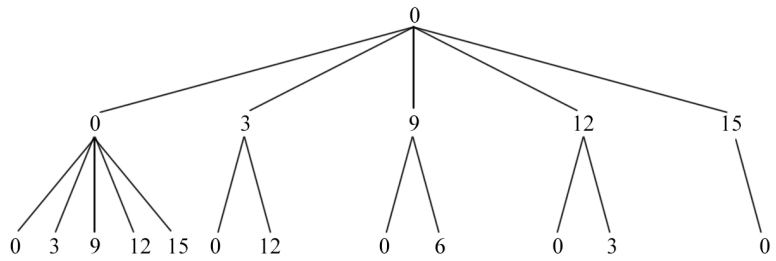

Figure 6. The output of the second iteration of Algorithm 2.

The results of the second iteration are shown by the diagram in Figure 6. It can be seen that the number of nodes with same label, i, at one level is equal to $\beta(i)$. The results of the second iteration are presented in Table 2, in which, for brevity, all non-zero values of $\beta(i)$ are placed under the corresponding profiles i. In total, there are five nodes with label 0 in level 2, and so that $\tau(2, 4) = \beta(0) = 5$.

The results of the third iteration are shown in Figure 7, in which, for brevity of drawing, only pairwise distinct (node-labeled) subtrees between levels 2 and 3 are retained, with multiplicities shown

Table 3. Short form of recording the results of the third iteration; $\beta(0) = 11$**.**

i	0	3	9	12	15	6
$\beta(i)$	11	7	6	7	5	1

under respective subtrees; for example, the leftmost subtree occurs 5 times. There are a total number of $5+2+1+2+1 = 11$ nodes labeled 0 on level 3. Also, the results of the third iteration are summarized briefly in Table 3.

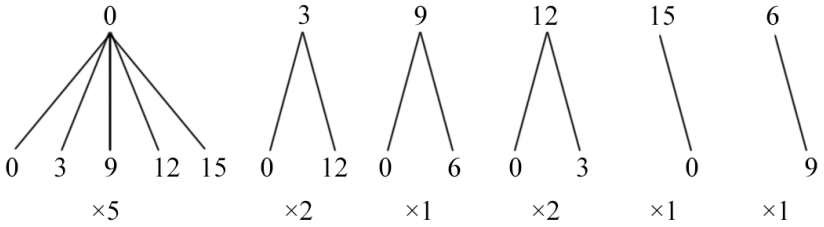

Figure 7. The output of the third iteration of Algorithm 2.

The fourth, and final, iteration is performed similarly; see Figure 8. With help from Figure 8, we obtain $\beta(0) = 11+7+6+7+5 = 36$, and so that $\tau(4,4) = 36$. Since after the second and third iterations $\beta(0)$ is equal to 5 and 11, respectively, then $\tau(2,4) = 5$ and $\tau(3,4) = 11$.

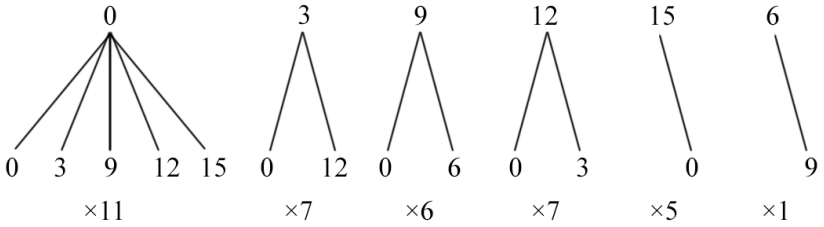

Figure 8. The output of the fourth iteration of Algorithm 2.

The computational complexity of Algorithm 2 is significantly reduced if compared to the direct recursive method (Section 3). In fact, the complexity is determined by Step 1, in which $W_k(p)$ is calculated for $p = 0, \ldots, 2^k - 1$, in combination with Step 3.2 (iterated n times), in which $|W_k(q)| - 1$ additions are performed for each $q = 0, \ldots, 2^k - 1$. Since by Theorem 5.2, the complexity of calculating $W_k(p)$ is equal to $O(\varphi^k)$, it follows that the complexity of Algorithm 2 is estimated as

$$2^k O(\varphi^k) + n\, 2^k O(\varphi^k) = O(n\,(2\varphi)^k).$$

Furthermore, Algorithm 2 can be further optimized in terms of execution time: At the last iteration of Step 3, we only need the value of $\beta(0)$, after which the process can be immediately terminated without further calculation of $\beta(1)$, $\beta(2)$, ...

7. Comparison with known algorithms

In this section we highlight advantages of our enhanced algorithm (Algorithm 2, Section 6) by comparing it against existing algorithms [1, 6, 11].

We start with Ahrens' algorithm [1]. While our algorithm readily allows to evaluate $\tau(n,k)$ for very large n, such evaluations are missing in [1]. Moreover, all attempts to calculate $\tau(18\,000, 8)$, using Ahrens' algorithm without our key subroutine GVS (Section 4), have failed because of memory shortage. In contrast, our algorithm has even succeeded in calculating the value of $\tau(150\,000, 8)$, working with the same software and hardware, and here is the result:

$$\tau(150\,000, 8) = 13873887020492488\ldots05663089457,$$

where the *whole* decimal string has been actually calculated, although only the beginning and end are shown above, with the whole string being of length 143 189.

We now proceed to the algorithms [6, 11] producing outputs which are either incorrect or incomplete, respectively. The following are the values for $\tau(10, 8)$, $\tau(11, 8)$ and $\tau(12, 8)$, inaccurately calculated in [6]: 1031151240, 8940739821 and 82741005789 (respectively), while their true values calculated using our algorithm are as follows: 1031151241, 8940739824 and 82741005829. The values obtained by using our algorithm coincide with the ones obtained in [11]. However, in [11], an unsuccessful attempt is made to fill in the table of values of $\tau(n,k)$ for $n = 2, 3, \ldots, 30$, $k = 0, 1, \ldots, 9$. More precisely, for no k ($k > 4$), is the table completely filled out in [11] for n running through 2 to 30. In contrast, our algorithm readily allows us to obtain the values of $\tau(n,k)$ for

$n = 1, 2, \ldots, 513$, $k = 16$ (see [8]), and

$n = 0, 1, \ldots, 11$, $k = 32$ (see [9]).

Finally, the bottom line is that the reason for the high accuracy of our enhanced algorithm lies in highly efficient and tech-enhanced usage of memory.

Acknowledgments: We wish to express our appreciation to Professor Curtis Cooper of the University of Central Missouri for reading a previous version of the manuscript and making a number of valuable comments, and also to our illustrator, Alex Lao, for preparing the artwork for this paper. Finally, we would like to thank the two anonymous peer reviewers for taking the time to read an earlier draft of this article and for their helpful and constructive comments and suggestions.

References

[1] J. H. Ahrens, Paving the chessboard, Journal of Combinatorial Theory, Series A. 31(3) (1981) 277–288.
[2] S. Butler, P. Horn, E. Tressler, Intersecting domino tilings, The Fibonacci Quarterly. 48(2) (2010) 114–120.
[3] D. DeFord, Enumerating distinct chessboard tilings, The Fibonacci Quarterly. 52(5) (2014) 102–116.
[4] R. L. Graham, D. E. Knuth, O. Patashnik, Concrete mathematics. A foundation for computer science, 2nd edition, Addison-Wesley Publishing Company, Reading, MA. (1994).
[5] P. W. Kasteleyn, The statistic of dimers on a lattice I: The number of dimer arrangements on quadratic lattice, Physica. 27 (1961) 1209–1225.
[6] D. Klarner, J. Pollack, Domino tilings of rectangles with fixed width, Discrete Mathematics. 32(1) (1980) 45–52.
[7] L. Lóvász, M. D. Plummer, Matching theory, North-Holland Math. Stud. 121, Ann. Discrete Math. 29, North-Holland, Amsterdam (1986).

[8] A. M. Magomedov, S. Lawrencenko, Number of domino tilings of a 16 X n rectangle, The On-Line Encyclopedia of Integer Sequences, A340532 – OEIS (10 Jan. 2021).
[9] A. M. Magomedov, S. Lawrencenko, Number of domino tilings of a 32 X n rectangle, The On-Line Encyclopedia of Integer Sequences, A347054 – OEIS (14 Aug. 2021).
[10] J. K. Percus, Combinatorial methods, Applied Mathematical Sciences. 4, Springer-Verlag, New York-Heidelberg (1971).
[11] R. C. Read, A note on tiling rectangles with dominoes, The Fibonacci Quarterly. 18(1) (1980) 24–27.
[12] A. Skopenkov, Mathematics via Problems: Part 1: Algebra, translated from the Russian original by P. Zeitz and S. G. Shubin, foreword by P. Zeitz, MSRI Math. Circ. Libr. 25, MSRI, Berkeley, CA; American Mathematical Society, Providence, RI. (2021).
[13] H. N. V. Temperley, M. E. Fisher, Dimer problem in statistical mechanics—an exact result, The Philosophical Magazine: A Journal of Theoretical Experimental and Applied Physics. 6(68) (1961) 1061–1063.
[14] L. G. Valiant, The complexity of computing the permanent, Theoretical Computer Science. 8(2) (1979) 189–201.