Applied Mathematics and Computation

Computing the Tutte polynomial of Archimedean tilings

D. Garijo [a,*], M.E. Gegúndez [b], A. Márquez [a], M.P. Revuelta [a], F. Sagols [c]

[a] Departamento de Matemática Aplicada I, Universidad de Sevilla, Spain
[b] Departamento de Matemáticas, Universidad de Huelva, Spain
[c] Departamento de Matemáticas, CINVESTAV-IPN, Mexico

O

Keywords:
Tutte polynomial
Archimedean tilings
Tutte polynomial evaluations

A B S T R A C T

We describe an algorithm to compute the Tutte polynomial of large fragments of Archimedean tilings by squares, triangles, hexagons and combinations thereof. Our algorithm improves a well known method for computing the Tutte polynomial of square lattices. We also address the problem of obtaining Tutte polynomial evaluations from the symbolic expressions generated by our algorithm, improving the best known lower bound for the asymptotics of the number of spanning forests, and the lower and upper bounds for the asymptotics of the number of acyclic orientations of the square lattice.

1. Introduction

The Tutte polynomial is a two-variable polynomial $T(G;x,y)$ associated with any graph G, which contains a significant amount of information about the graph. It encodes many numerical invariants such as the number of spanning trees, the number of spanning forests, the number of acyclic orientations and the number of spanning connected subgraphs. Additionally, many one-variable polynomials associated to graphs can be obtained from partial evaluations of the Tutte polynomial, among them, the chromatic polynomial, the flow polynomial and the reliability polynomial [2].

It is well-known that the computation of the Tutte polynomial is NP-hard [16]. Also, evaluating $T(G;x,y)$ for specific points (x,y) is NP-hard except for the points on the hyperbola $(x-1)(y-1)=1$ or when (x,y) equals $(1,1),(-1,-1),(0,-1),(-1,0),(i,-i),(-i,i),(j,j^2),(j^2,j)$ where $j=e^{(2\pi i/3)}$ for which it can be done in polynomial time [16,26]. Vertigan [29] has extended this by showing that a similar result holds for planar graphs except that, in this case, for the additional points lying on the hyperbola $(x-1)(y-1)=2$, the problem can be solved again in polynomial time.

Thus, there are no efficient algorithms to compute the Tutte polynomial of large graphs, i.e., dealing with a significant number of vertices. Royle [20] developed an algorithm which determines in seconds the Tutte polynomial of scarce graphs with at most 14 vertices. Sekine et al. [21] described algorithms requiring about an hour in a workstation (from 1995) to compute Tutte polynomials of general random graphs with 14 vertices and 91 edges; also, it took a similar time to compute the Tutte polynomial of a plane graph with 144 vertices and 246 edges using their method. Haggard et al. [14] presented an algorithm that can go much further, processing random graphs of 14 vertices and 70 edges in seconds. They repeated their experiments with random graphs containing 16 vertices and 100 edges requiring 3000 s; they also performed additional experiments using random planar graphs, requiring 3500 s for 3-regular planar graphs with 40 vertices, and the same time

* Corresponding author.
E-mail addresses: dgarijo@us.es (D. Garijo), manuel.gegundez@gmail.com (M.E. Gegúndez), almar@us.es (A. Márquez), pastora@us.es (M.P. Revuelta), fsagols@math.cinvestav.edu.mx (F. Sagols).

was needed on 4-regular planar graphs with 26 vertices. The authors also used their implementation to find counterexamples to a conjecture of Welsh on the location of the real roots of the flow polynomial of a graph.

A natural approach to the problem of computing Tutte polynomials is to restrict the study to specific families of graphs. Calkin et al. [6] used transfer matrices to compute Tutte polynomials of square lattices in order to obtain asymptotic results for the number of forests and acyclic orientations. Nevertheless, they mentioned that, computationally, their method is not feasible since the required space to store the transfer matrix grows exponentially.

From the point of view of statistical physics, the Tutte polynomial appears as the partition function of the q-state Potts model $Z(G;q,v)$ [27]. Indeed,

$$T(G;x,y) = (x-1)^{-c}(y-1)^{-n}Z(G;(x-1)(y-1),(y-1)),$$

where G is a graph with n vertices and c connected components. In particular, the Tutte polynomial is remarkably close to two classical models: the Ising and the Potts models. They have an interpretation as direct evaluations of the Tutte polynomial [25].

Part of our motivation for computing the Tutte polynomial of fragments of Archimedean tilings is because of the importance of some of these structures as vertex models in statistical physics. There is an extensive physics literature about computing the partition function of families of graphs, most of which are square lattices, triangular lattices or different types of graphs with repetitive structures. See [7,8,23] for information about this topic.

In this paper, we present an algorithm for computing the Tutte polynomial of large fragments of Archimedean tilings by squares, triangles, hexagons and the possible combinations thereof. These tilings are introduced together with the Tutte polynomial in Section 2. In Section 3, we introduce the notion of *graphic tile* which is the key tool to codify Archimedean tilings as repetitive structures of graphs. Using this codification, in Section 4, we first describe the algorithm pointing out some factors which might affect its performance. Then, we discuss its running time complexity, and explain how our algorithm improves the method presented by Calkin et al. [6] for the square lattice. Finally, Section 5 is devoted to obtain numerical results since Tutte polynomial evaluations have important physical applications. We also compare our numerical results against the evaluations performed by Calkin et al. [6], improving the best known lower bound for the asymptotics of the number of spanning forests, and the lower and upper bounds for the asymptotics of the number of acyclic orientations of the square lattice. For evident reasons we do not include the Tutte polynomials we have calculated, but the files containing them, the programs used, and the documentation are available on request from fsagols@math.cinvestav.edu.mx.

2. Preliminaries

2.1. Archimedean tilings

A *plane tiling* is a countable family $\{T_1, T_2, \ldots\}$ of closed sets, called *tiles*, which cover the Euclidean plane without gaps or overlaps, i.e., the union of the tiles $T = \cup_i\{T_i\}$ is the whole plane and the interiors of the tiles are pairwise disjoint. In this paper, we follow the notation and terminology introduced by Grünbaum and Shephard in [13].

An *edge-to-edge* tiling is a type of plane tiling where each tile is a regular convex polygon and the intersection of every pair of non-disjoint tiles is either a *vertex of the tiling* or an *edge of the tiling* which correspond to vertices and edges of the polygons, respectively. Thus, no tile shares a partial side with any other tile.

A vertex of the tiling is said to be *of type* $p_1.p_2.\cdots.p_r$ if it is surrounded in cyclic order by regular polygons of order $p_1, p_2, \ldots, p_r$.

All the vertices of the tiling are *of the same type* if the same types of polygons meet in the same order (ignoring orientation) at each vertex. In this case, the tiling is denoted by $p_1.p_2.\cdots.p_r$, referring the vertex type.

An edge-to-edge plane tiling by copies of a single regular polygon is called a *regular tiling*. It is well known that there are exactly three regular tilings of the plane: by equilateral triangles, by squares or by regular hexagons (which are written, for short, $3^6, 4^4$ and 6^3, respectively). More generally,

Theorem 2.1 [13]. *There are precisely 11 distinct types of edge-to-edge tilings by regular polygons such that all the vertices of the tiling are of the same type.*

These 11 tilings are called *Archimedean tilings* (also *homogeneous tilings* or *semi-regular tilings*). Note that they include the three regular tilings.

We have considered those Archimedean tilings by three types of polygons: squares, triangles and hexagons. This gives a total of eight tilings which are: $3^6, 4^4, 6^3$, 3.4.6.4, $3^3.4^2$, $3^2.4.3.4$, $3^4.6$, and 3.6.3.6. Nevertheless, for the sake of brevity, we only show as examples in this paper the tilings 4^4 and 3.4.6.4. In both cases, where no confusion can arise, we shall write 4 and 3.4.6, indicating only the polygons that are used to obtain the tiling. See Figs. 1(d) and 2(b) for fragments of these tilings (the notation used in the caption of these figures will be introduced in the next section).

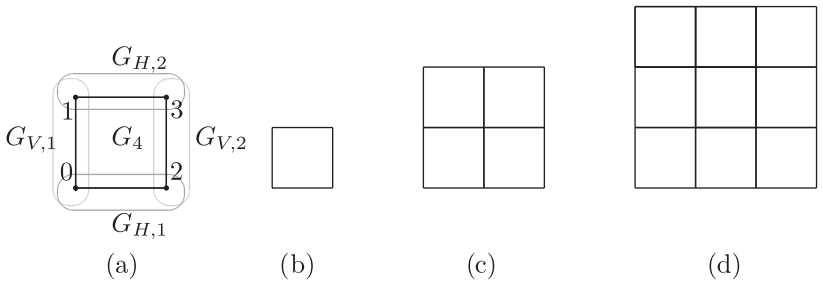

Fig. 1. Graphic tile $\mathcal{T}_4$: (a) $G_4, G_{H,1}, G_{H,2}, G_{V,1}$ and $G_{V,2}$; (b) $\mathcal{T}_4(1,1)$; (c) $\mathcal{T}_4(2,2)$ and (d) $\mathcal{T}_4(3,3)$.

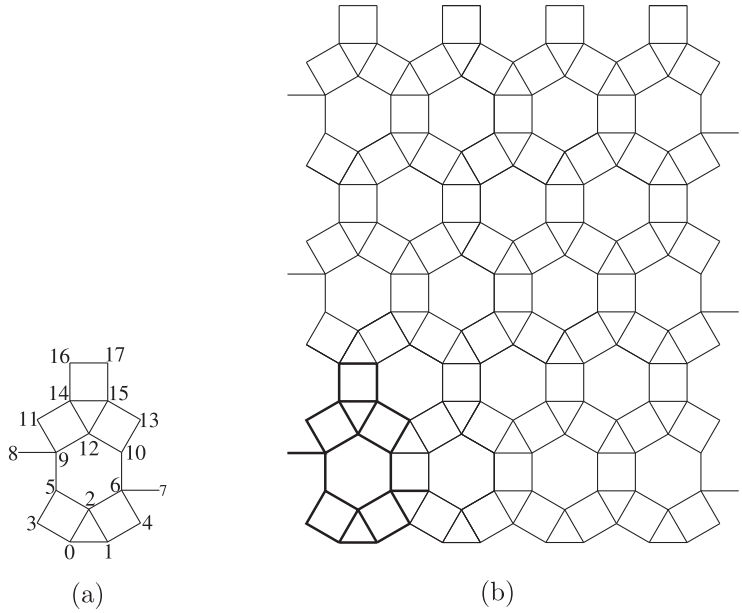

Fig. 2. Graphic tile $\mathcal{T}_{346}$: (a) G_{346} and (b) $\mathcal{T}_{346}(3,4)$.

2.2. The Tutte polynomial

Let $G = (V, E)$ be a graph with $k(G)$ connected components, *rank* $r(G) = |V| - k(G)$ and *nullity* $n(G) = |E| - r(G)$. For every subset $A \subseteq E$, its *rank* is $r(A) = |V| - k(G|A)$ where $k(G|A)$ is the number of connected components of the spanning subgraph (V, A). The Tutte polynomial of G is given by

$$T(G; x, y) = \sum_{A \subseteq E} (x-1)^{r(E)-r(A)} (y-1)^{|A|-r(A)}. \tag{1}$$

The graphs resulting from *deleting* and *contracting* an edge $e \in E$ are denoted by $G - e$ and G/e, respectively. An edge e is a *bridge* in G if $r(G-e) = r(G) - 1$ and a *loop* in G if $n(G/e) = n(G) - 1$. The edges that are neither a bridge nor a loop are called *ordinary*.

A function F from (isomorphism classes of) graphs to the polynomial ring $\mathbb{C}[\alpha, \beta, \gamma, x, y]$ is a generalized Tutte–Gröthendieck invariant [4,11] if it satisfies, for each graph $G = (V, E)$ and any edge $e \in E$, the following recurrence relations

$$F(G) = \begin{cases} \gamma^{|V|} & \text{if } E = \emptyset, \\ xF(G/e) & \text{if } e \text{ is a bridge}, \\ yF(G-e) & \text{if } e \text{ is a loop}, \\ \alpha F(G/e) + \beta F(G-e) & \text{if } e \text{ is an ordinary edge}. \end{cases} \tag{2}$$

The Tutte polynomial is an example of Tutte–Gröthendieck invariant for $\alpha = \beta = \gamma = 1$ [26] and so,

$$T(G;x,y) = \begin{cases} 1 & \text{if } E = \emptyset, \\ xT(G/e;x,y) & \text{if is a bridge}, \\ yT(G-e;x,y) & \text{if is a loop}, \\ T(G/e;x,y) + T(G-e;x,y) & \text{if } e \text{ is ordinary}. \end{cases} \tag{3}$$

This alternative formulation in terms of deleting and contracting edges is, perhaps, one of its most important properties. In fact, the following theorem states that every Tutte–Gröthendieck invariant is essentially an evaluation of the Tutte polynomial.

Theorem 2.2 [4]. *If F is a generalized Tutte–Gröthendieck invariant satisfying Eq. (3) then*

$$F(G) = \gamma^{k(G)}\alpha^{r(G)}\beta^{n(G)}T\left(G;\frac{x}{\alpha},\frac{y}{\beta}\right).$$

The interpretation of this expression for $\alpha = 0$ or $\beta = 0$ is given in [3].

Note that the definition of the Tutte polynomial as a Tutte–Gröthendieck invariant makes clear that the coefficients of the Tutte polynomial are non-negative integers [3] which is not evident from Eq. (1).

3. Codifying large fragments of Archimedean tilings

We start by introducing the notion of *graphic tile* which is the key tool to codify Archimedean tilings as repetitive structures of graphs.

A *graphic tile* is a 7-tuple $\mathcal{T} = (G, G_{H,1}, G_{H,2}, \phi_H, G_{V,1}, G_{V,2}, \phi_V)$ where: (1) G is a graph, called the *base graph*, (2) $G_{H,1}, G_{H,2}, G_{V,1}$ and $G_{V,2}$ are subgraphs of G such that $V(G_{H,1}) \cap V(G_{H,2}) = \emptyset$ and $V(G_{V,1}) \cap V(G_{V,2}) = \emptyset$, (3) ϕ_H is an isomorphism from $G_{H,1}$ to $G_{H,2}$, (4) ϕ_V is an isomorphism from $G_{V,1}$ to $G_{V,2}$.

A first example of graphic tile is illustrated in Fig. 1(a) where we have

$$\mathcal{T}_4 = (G_4 = (\{0,1,2,3\},\{(0,1),(1,3),(3,2),(2,0)\}), G_{H,1} = (\{0,2\},\{(0,2)\}), G_{H,2} = (\{1,3\},\{(1,3)\}),$$
$$\phi_H = \{\{(0,1),(2,3)\}\}, G_{V,1} = (\{0,1\},\{(0,1)\}), G_{V,2} = (\{2,3\},\{(2,3)\}), \phi_V = \{\{(0,2),(1,3)\}\})$$

and an isomorphism is written as $\phi = \{(v,\phi(v)) \mid v \in V\}$.

From now on, let m,n be positive integers and let $\mathcal{T}$ be a graphic tile. Denote by $\mathcal{T}(m,n)$ the graph composed by an arrangement of mn graphs $G_{i,j}, 0 \leqslant i \leqslant m-1, 0 \leqslant j \leqslant n-1$, where each $G_{i,j}$ is a copy of G. For $i = 0,\ldots,m-2$ the vertices in $G_{H,2}$ of $G_{i,j}$ are identified with their corresponding isomorphic vertices in $G_{H,1}$ of $G_{i+1,j}$, and for $j = 0,\ldots,n-2$ the vertices in $G_{V,2}$ of $G_{i,j}$ are identified with their corresponding isomorphic vertices in $G_{V,1}$ of $G_{i,j+1}$. Any parallel edge generated in this construction is deleted.

Among all possible fragments of tilings generated by graphic tiles as described above, we focus on Archimedian tilings. To identify them, we use the vertex type as a sub-index for the graphic tile $\mathcal{T}$ and also for the base graph G (see Figs. 1 and 2). For instance,

$$\mathcal{T}_{3.4.6} = (G_{3.4.6}, G_{H,1} = (\{0,1\},\{(0,1)\}),\ G_{H,2} = (\{16,17\},\{(16,17)\}),\ \phi_H = \{(0,16),(1,17)\},$$
$$G_{V,1} = (\{3,5,8,11\},\emptyset),\ G_{V,2} = (\{4,7,10,13\},\emptyset),\ \phi_V = \{(3,4),(5,7),(8,10),(11,13)\})$$

and $\mathcal{T}_{3.4.6}(m,n)$ denotes the fragment of the Archimedean tiling by triangles, squares and hexagons formed by the arrangement of mn copies of the base graph (Fig. 2). To simplify this notation the dots separating the numbers of the type are dropped.

We shall use the following non-standard definition of sum of two graphs. For a graph H and $e \in E(H)$, denote by $p_H(e)$ the multiplicity of e (i.e., the number of edges sharing the same end vertices). Given two vertex-labeled graphs H_1 and H_2, the *sum* $H_1 + H_2$ has vertex set $V(H_1) \cup V(H_2)$ and edge set consisting of all edges $e \in E(H_1) \cup E(H_2)$ taken with multiplicity $p_{H_1+H_2}(e) = \max(p_{H_1}(e), p_{H_2}(e))$. Fig. 3 illustrates this operation.

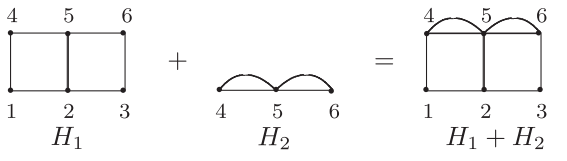

Fig. 3. Sum of H_1 and H_2.

Let k be an integer in $[1,m]$. The *k-level graph* of $\mathcal{T}(m,n)$, denoted by L_k, is the graph induced by the edges in $\mathcal{T}(m,n)$ coming from the copies $G_{k-1,j}$ for $0 \leqslant j \leqslant n-1$; therefore $\mathcal{T}(m,n) = \Sigma_{k=1}^{m} L_k$ (see Fig. 4). Clearly, two graphs L_k and $L_{k'}$ are isomorphic for any pair of integers $k, k' \in [1,m]$.

The edges in L_k are split into two sub-sets: B_k containing those edges coming from a copy of one edge in $G_{H,1}$, and $E_k = E(L_k) - B_k$. Note that the edges in B_k do not necessarily form a connected graph. Observe also that these edges belong to both L_k and $L_{k-1}, 1 < k \leqslant m$.

For technical reasons, we shall use L_{m+1}, B_{m+1} and E_{m+1} which are natural extensions to the graphs introduced above.

4. Computing the Tutte polynomial of $\mathcal{T}(m,n)$

In this section, we first present an algorithm to compute the Tutte polynomial of large fragments of Archimedean tilings. Then, we analyze its running time complexity, and explain how our algorithm improves the method presented by Calkin et al. in [6] for the square lattice.

The computation of the Tutte polynomial of $\mathcal{T}(m,n)$ proceeds by levels starting at $k=m$, then it continues with level $k=m-1$ and so on. To start, we process level m applying the recursive relations given by Eq. (3) to $T(L_m;x,y)$. Then an expression of the following type is obtained,

$$T(L_m;x,y) = \sum_{i=1}^{t} P_i(x,y)\,T(R_{m,i};x,y), \tag{4}$$

where $R_{m,i}$ is a *residual graph*, i.e., a graph obtained from L_m by successively deleting or contracting edges that satisfies two conditions: (1) it contains B_m as a subgraph; (2) the resulting graph of either deleting or contracting any of its edges does not contain B_m as a subgraph. It is easy to check that B_m is a residual graph of L_m, say $B_m = R_{m,1}$. Further, variable t in Eq. (4) indicates the number of different residual graphs obtained from L_m, and $P_i(x,y)$ is a polynomial in x,y with coefficients in $\mathbb{Z}$.

Note that if m is replaced by an arbitrary k in Eq. (4), the equality is preserved since L_m and L_k are isomorphic graphs and so the residual graphs $R_{m,i}$ and $R_{k,i}$ are exactly the same. Hence, an explicit reference to the level is irrelevant and the residual graphs corresponding to any L_k will be written as $R_i, 1 \leqslant i \leqslant t$, assuming that $R_1 = B_k$.

Now, for each residual graph $R_i, 1 \leqslant i \leqslant t$, we determine $T(L_k + R_i;x,y)$ by applying the same contraction–deletion procedure as above (except for the case $L_k + R_1$ which is isomorphic to L_k). Different residual graphs from those obtained for L_k might appear. Denote by $\mathcal{R}_1$ the set of all residual graphs coming from $L_k + R_i$ for $i = 1,\ldots,t$. Then, apply again the contraction–deletion process to $L_k + R$ for every $R \in \mathcal{R}_1$ giving rise to a new set of residual graphs $\mathcal{R}_2$. The procedure is repeated until no new residual graph appears. Denote by $R_1, R_2, \ldots, R_s, s \geqslant t$, the complete sequence of residual graphs obtained after completing the process. Compute $T(L_k + R_\ell;x,y)$ for $\ell = 1,\ldots,s$ (many of which have already been computed in previous steps of the process) obtaining similar expressions to Eq. (4),

$$T(L_k + R_\ell;x,y) = \sum_{j=1}^{s} P_{\ell j}(x,y)\,T(R_j;x,y). \tag{5}$$

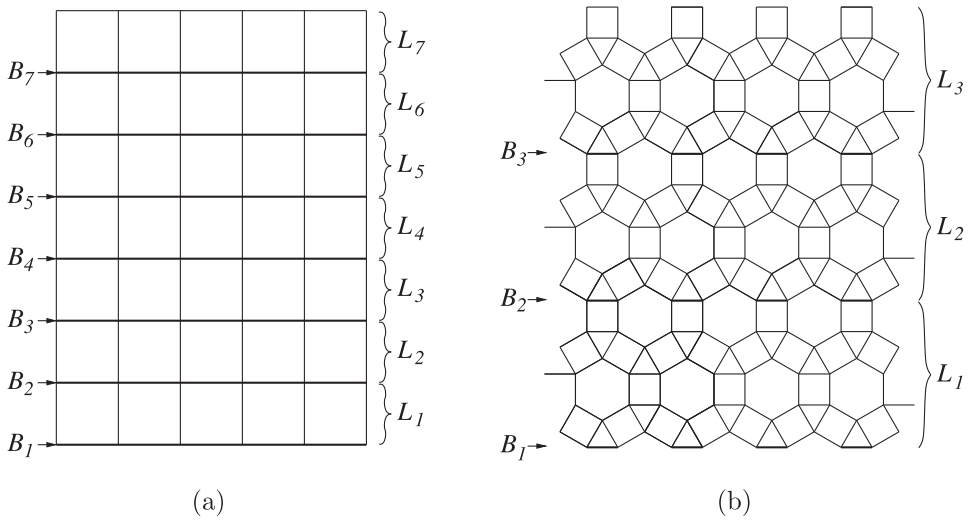

Fig. 4. Definitions of L_k and B_k: (a) for $\mathcal{T}_4(7,5)$ and (b) for $\mathcal{T}_{346}(3,4)$.

Notice that only deletions and contractions which do not modify B_k are done. Observe also that the number s of residual graphs could be infinite unless the edges in R_ℓ are processed first. The deletion and the contraction of these edges do not modify B_k since, by definition, $V(G_{H,1})$ is disjoint from $V(G_{H,2})$; in other words B_k and R_ℓ do not have common vertices. After all edges in R_ℓ are deleted or contracted, the remaining graph has at most $|E_m|$ edges which could be deleted and contracted, so the number of residual graphs is bounded and s is finite. Thus, we have shown the following result.

Proposition 4.1. *For every integer $k, 1 \leqslant k \leqslant m$, there exists a finite number s of residual graphs $R_1, \ldots, R_s$ with $R_1 = B_k$ and such that, for every $\ell \in [1, s]$ the Tutte polynomial $T(L_k + R_\ell; x, y)$ can be expressed in terms of $T(R_j; x, y)$ for $j = 1, \ldots, s$.*

Observe that, in Eq. (5), $P_{\ell,j}(x, y)$ are polynomials in x and y (which might be zero) and independent of k since for any pair $k, k' \in [1, m]$ the graph $L_k + R_\ell$ is isomorphic to $L_{k'} + R_\ell$. Hence, for each residual graph R_ℓ, the Tutte polynomial $T(\mathcal{T}(m, n) + R_\ell; x, y)$ can be computed as follows

$$T(\mathcal{T}(m,n) + R_\ell; x, y) = T\left(\sum_{k=1}^{m} L_k + R_\ell; x, y\right) = \sum_{j=1}^{s} P_{\ell,j}(x, y) \cdot T\left(\sum_{k=1}^{m-1} L_k + R_j; x, y\right). \tag{6}$$

Then

$$\begin{bmatrix} T\left(\sum_{k=1}^{m} L_k + R_1; x, y\right) \\ T\left(\sum_{k=1}^{m} L_k + R_2; x, y\right) \\ \cdots \\ T\left(\sum_{k=1}^{m} L_k + R_s; x, y\right) \end{bmatrix} = M \cdot \begin{bmatrix} T\left(\sum_{k=1}^{m-1} L_k + R_1; x, y\right) \\ T\left(\sum_{k=1}^{m-1} L_k + R_2; x, y\right) \\ \cdots \\ T\left(\sum_{k=1}^{m-1} L_k + R_s; x, y\right) \end{bmatrix} = M^2 \cdot \begin{bmatrix} T\left(\sum_{k=1}^{m-2} L_k + R_1; x, y\right) \\ T\left(\sum_{k=1}^{m-2} L_k + R_2; x, y\right) \\ \cdots \\ T\left(\sum_{k=1}^{m-2} L_k + R_s; x, y\right) \end{bmatrix} = \ldots = M^m \cdot \mathbf{b}, \tag{7}$$

where

$$M = \begin{bmatrix} P_{1,1}(x,y) & P_{1,2}(x,y) & \ldots & P_{1,s}(x,y) \\ P_{2,1}(x,y) & P_{2,2}(x,y) & \ldots & P_{2,s}(x,y) \\ \ldots & \ldots & \ldots & \ldots \\ P_{s,1}(x,y) & P_{s,2}(x,y) & \ldots & P_{s,s}(x,y) \end{bmatrix} \quad \text{and} \tag{8}$$

$$\mathbf{b} = \begin{bmatrix} T(R_1; x, y) \\ T(R_2; x, y) \\ \cdots \\ T(R_s; x, y) \end{bmatrix}. \tag{9}$$

Thus, we reach our main result in this section.

Theorem 4.2. *The Tutte polynomial of $\mathcal{T}(m, n)$ is obtained in the first entry of the vector $M^m \cdot \mathbf{b}$ at right hand of Eq. (7).*

Note that Theorem 4.2 would be useless without Proposition 4.1 which guarantees that M has finite order and so the Tutte polynomial of $\mathcal{T}(m, n)$ can be evaluated in an effective way.

4.1. The implemented algorithm

The codification described in Section 3 and Theorem 4.2 allow developing the following algorithm to compute the Tutte polynomial of $\mathcal{T}(m, n)$. It takes as input the value n and a graphic tile $\mathcal{T}$ and yields the matrix M, and the vector $\mathbf{b}$ appearing at right hand of Eq. (7).

ALGORITHM TUTTE-MATRIX-COMPUTATION [To find the matrix elements to compute the Tutte polynomial of $\mathcal{T}(m,n)$].

Input: $n, \mathcal{T}$.
Output: M, **b**.
Method:
"M and B represent an $s \times s$ two-dimensional matrix and an s one-dimen-sional vector, respectively. Here s stands for the number of residual graphs and it will remain unknown until the end of the algorithm"

1. *LayerModel* $\leftarrow G$
2. **for** $i \leftarrow n$ **do**
3. Identify the last copy of $G_{V,2}$ incorporated to *LayerModel* with the subgraph $G_{V,1}$ of a new copy of G.
4. B_k*Model* $\leftarrow$Graph induced by the edges in *LayerModel* coming from a copy of $G_{H,1}$.
5. B_{k+1}*Model* $\leftarrow$Graph induced by the edges in *LayerModel* coming from a copy of $G_{H,2}$.
6. *ResidualQueue* $\leftarrow \{B_k\text{Model}\}$
7. *ordinal*$(B_k\text{Model}) \leftarrow 1$
8. *NextOrdinal* $\leftarrow 2$
9. **while** *ResidualQueue* $\neq \emptyset$ **do**
10. $R \leftarrow$ *extractElement*(*ResidualQueue*)
11. $H \leftarrow$ *LayerModel* $+ R$
"In this operation the isomorphic copy of B_{k+1}*Model* of *LayerModel* is identified to the isomorphic copy of B_k*Model* in R"
12. Starting with the edges in R apply the recursive formula to compute $T(H;x,y)$ until the expression is solely in terms of residual graphs.
13. **for** each residual graph R' in the expression for $T(H;x,y)$ **do**
14. **if** R' has neither been processed yet nor is in *ResidualQueue* **then**
15. *ResidualQueue* $\leftarrow$ *ResidualQueue* $\cup \{R'\}$
16. *ordinal*$(R') \leftarrow$ *NextOrdinal*
17. *NextOrdinal* $\leftarrow$ *NextOrdinal* $+ 1$
18. $M[\text{ordinal}(R), \text{ordinal}(R')] \leftarrow$coefficient of $T(R';x,y)$ in $T(H;x,y)$.
19. $B[\text{ordinal}(r)] \leftarrow T(R;x,y)$
20. **for** $i \leftarrow 1$ **to** *ordinal* $- 1$ **do**
21. **for** $j \leftarrow 1$ **to** *ordinal* $- 1$ **do**
22. **if** $M[i,j]$ was not assigned a value **then**
23. $M[i,j] \leftarrow 0$
24. Return M and **b**.

At line 12, bridges should be identified carefully. Since only an isolated layer is considered, one bridge in it could not be a bridge considering the layers below. To avoid this, virtual edges are added to H between the vertices in B_k which are connected by paths belonging to layers L_ℓ with $\ell < k$. Thus, an edge in H is a bridge in the whole structure if and only if it is a bridge in $H \cup \{\text{virtual edges}\}$.

The algorithm does not need to construct the whole $\mathcal{T}(m,n)$, only one layer is built and stored in *layerModel*. At each iteration of the **while** in step 9, one residual graph is added to the top of *layerModel* to find the Tutte polynomial of the resulting graph. Variable *NextOrdinal* assigns to each residual graph an ordinal number to identify it and to locate the right entry in M of every coefficient of $T(H;x,y)$. This code is improved changing line 14 by

14'. **if** (R' has neither been processed yet nor appears in *ResidualQueue*) and not (*Isomorphic* $\leftarrow$ (Is isomorphic *LayerModel* $+ R'$ to *LayerModel* $+ R''$ for a residual graph R'' that has been processed or is in *ResidualQueue*?)) **then**

and adding the lines

17.1 **else if** *Isomorphic* = *true* **then**

17.2 Change any occurrence of $T(R';x,y)$ in $T(H;x,y)$ by $T(R'';x,y)$ and re-associate the coefficients.

17.3 **continue**

As it will be shown in the next subsection, this modification in the code can be done efficiently since the overall time complexity of the algorithm does not change. On the other hand, these lines improve the code in the sense that the dimension of the matrix M generated by the algorithm is reduced by about half.

4.2. Complexity of the algorithm

The running time complexity of Algorithm Tutte-Matrix-Computation is dominated by the number of times that line 9 is executed, and this is precisely the number of residual graphs which depends on n, and from now on, will be denoted by $s(n)$. The exact value of $s(n)$ depends upon the graphic tile and the order in which the edges are contracted-deleted. The **for** at line 13 is executed at most $s(n)$ times, and a dictionary is used in line 14 to verify in $O(\log s(n))$ time if R' has been processed or not. Thus, we have the following result.

Theorem 4.3. *The running time complexity of Algorithm* Tutte-Matrix-Computation is $O(s(n)^2 \log s(n))$.

As it was said before, the code is improved changing line 14 by 14' and adding the lines 17.1 to 17.3. Now, we show that this implementation can be done efficiently since the graph isomorphism problem can be solved in linear time for residual graphs, and so the overall time complexity of the algorithm does not change.

The graph isomorphism problem (GI) consists of determining whether two given graphs are isomorphic. Its complexity is one of the major open problems: It is easy to see that GI $\in$ NP, but it is unknown whether GI $\in$ P $\cup$ NP-complete [10]. For residual graphs, GI can be solved in linear time: for two different residual graphs R_i and R_j, we are only interested in isomorphisms between R_i and R_j that can be extended to isomorphisms between $\sum_{k=1}^{\ell} L_k + R_i$ and $\sum_{k=1}^{\ell} L_k + R_j$ for arbitrarily large values of ℓ. That is because in Eq. (6), the recursion is established in terms of the Tutte polynomials $T\left(\sum_{k=1}^{\ell} L_k + R_i; x, y\right)$ where ℓ is arbitrarily large.

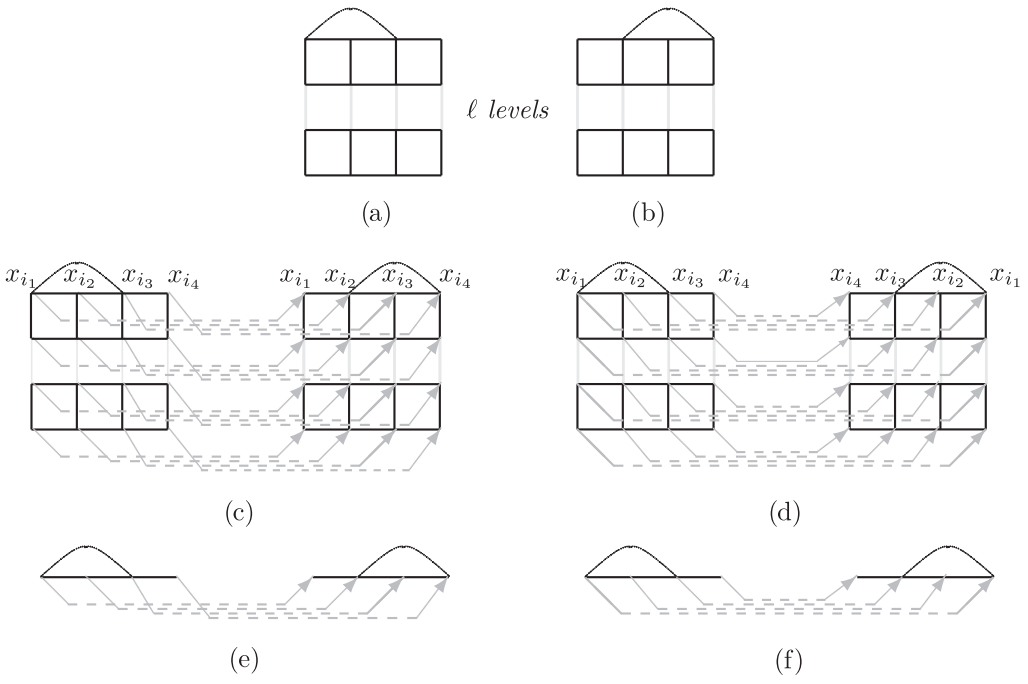

Fig. 5. Possible isomorphisms between residual graphs: in (a) and (b) appear, respectively, graphs $\sum_{k=1}^{\ell} L_k + R_i$ and $\sum_{k=1}^{\ell} L_k + R_j$ for $\mathcal{T}_4$; in (c) and (d) are depicted the two possible isomorphisms $f_{1,\ell}$ and $f_{2,\ell}$, respectively; in (e) and (f) are shown, respectively, the restrictions of $f_{1,\ell}$ and $f_{2,\ell}$ on R_i.

In Fig. 5(a) and (b), we give an example for $\mathcal{T}_4$. Since ℓ is arbitrarily large, there are only two possible isomorphisms between $\sum_{k=1}^{\ell} L_k + R_i$ and $\sum_{k=1}^{\ell} L_k + R_j$, say $f_{1,\ell}$ and $f_{2,\ell}$, depicted in Fig. 5(c) and (d), respectively. The function $f_{1,\ell}$ is the identity and $f_{2,\ell}$ transforms the path $x_{i_1}x_{i_2}\ldots x_{i_n}$ into $x_{i_n}x_{i_{n-1}}\ldots x_{i_1}$. Hence, we only need these two restricted functions (see Fig. 5(e) and (f)) and so the test can be done in $O(n)$ time. A similar argument is valid for any other graphic tile.

Proposition 4.4. *The graph isomorphism problem can be solved in linear time for residual graphs.*

4.3. The square lattice as input

In this subsection, we analyze the behavior of $s(n)$ when $\mathcal{T}_4$ is used as input and compare our algorithm against the method presented by Calkin et al. in [6].

A residual graph for $\mathcal{T}_4$ is an n-length path $P = x_0, \ldots, x_n$ embedded on the plane and oriented from x_0 to x_n, plus some additional non-crossing edges lying on the left side of the path. The complete sets of residual graphs for $\mathcal{T}_4$ and $n = 1, 2, 3$ are depicted in Fig. 6 where $s(1) = 2, s(2) = 8$ and $s(3) = 40$.

The number $s(n)$ satisfies the following recurrence inequality,

$$s(n) \leqslant 2\sum_{k=1}^{n-1} s(n-k)s(k),$$

where the factor 2 is necessary since none of the $\sum_{k=1}^{n-1} s(n-k)s(k)$ residual graphs counted in the recursion contains the edge x_0x_n, and so, we should count two residual graphs. This recurrence could be solved as the Catalan numbers recurrence [24]. The result is an exponential expression on n growing faster than the Catalan numbers.

Let us denote now by $s_a(n)$ the actual number of residual graphs generated by the algorithm. In general, $s_a(n) \leqslant s(n)$ and $s_a(n)$ depends on the graphic tile as well as on the order in which the edges in H are deleted–contracted at line 12 in the algorithm. Instead of developing the details to get an upper bound for $s(n)$, we rather present in Table 1 the experimental values of $s_a(n)$, obtained with the computer program implementing Algorithm Tutte-Matrix-Computation when calculating the Tutte polynomial of $\mathcal{T}_4(n, n)$ for $n = 1, \ldots, 10$.

In this table we have also included the Catalan numbers C_n. Observe that in all the computations, $s_a(n)$ is lower than C_{n+1}. In the column fourth of Table 1, we note that the algorithm first started improving this advantage but from $n = 5$, it decreased gradually. If this tendency continues, for $n \approx 14$ the situation will be reverted.

We compare now our algorithm against the method presented by Calkin et al. in [6]. There, a square transfer matrix of size C_{n+1} is built to evaluate $T(\mathcal{T}_4(n, n); x, y)$ based on the rank polynomial. The transfer matrix works like the matrix M in our algorithm, but its size is bigger for $n \leqslant 10$, and probably for $n \leqslant 14$ if the tendency observed in Table 1 is maintained. Calkin et al. [6] report evaluations up to $n = 6$ and our algorithm advanced 4 steps more. Further improvements over this method are discussed in Section 5, there columns fifth and sixth of Table 1 are explained.

Similar behaviors of $s(n)$ and $s_a(n)$ are observed for the other Archimedian tilings we have worked with.

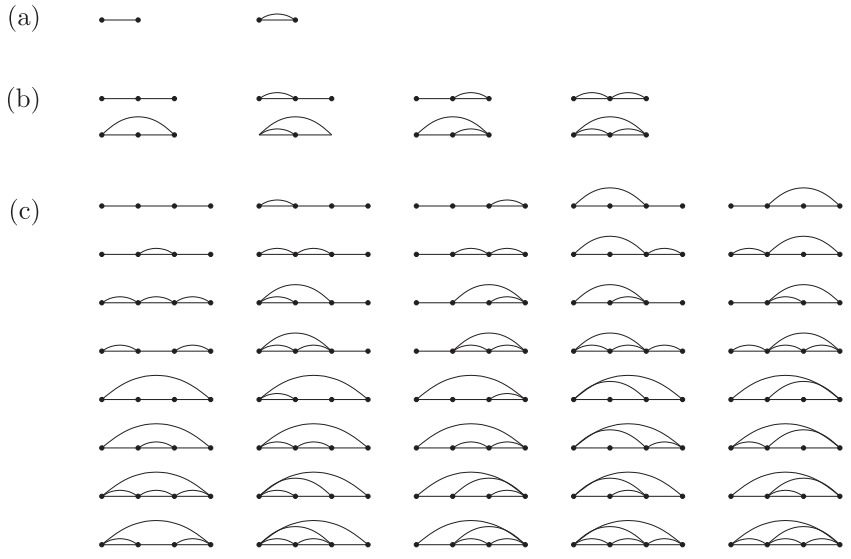

Fig. 6. Residual graphs: (a) $n = 1$, (b) $n = 2$, (c) $n = 3$.

Table 1
Residual graphs yield by Algorithm TUTTE-MATRIX-COMPUTATION on $\mathcal{T}_4$.

n	C_{n+1}	$s_a(n)$	$s_a(n)/C_{n+1}$	$Rank(M_{\mathcal{T}_4}(n;2,1))$	$Rank(M_{\mathcal{T}_4}(n;2,0))$
1	2	2	1	2	1
2	5	4	0.8	4	2
3	14	10	0.71	10	3
4	42	28	0.66	28	8
5	132	93	0.70	92	18
6	429	313	0.73	310	48
7	1430	1110	0.78	1099	120
8	4862	3948	0.81	3915	315
9	16,796	14,252	0.84	14,250	–
10	58,786	51,570	0.88	51,570	–

5. Numerical results

So far we have generated symbolic expressions of Tutte polynomials, but not evaluation at some specific point. Probably, from the statistical physics perspective, evaluations are much more relevant. In this section, we address this problem and compare our methods and results against those in [6] where, as it was mentioned before, a similar work is presented for the square lattice.

Tutte polynomial evaluations require arithmetic operations involving huge numbers, even for small tiling fragments. Exact results demand the use of arbitrary precision arithmetic. To generate either symbolic expressions or numerical evaluations, our computer programs receive an input parameter indicating the output type. In the former option, the polynomial is generated directly. In the last one, the specific point (x_0, y_0) to make the evaluation must be provided, and a text file containing the matrix M and vector $\mathbf{b}$ in Eq. (7) is generated; this file is processed by a C++ program that uses the NTL library [18] to compute the final result. We have chosen NTL because it is well designed to manage arbitrary precision arithmetic, and it is widely free available in platforms which are UNIX compatible.

Thus, throughout this section, M and $\mathbf{b}$ represent the numeric matrix and the numeric vector obtained from evaluating Expressions (8) and (9) at a specific point (x_0, y_0). They are used in Eq. (7) to evaluate the Tutte polynomial at the point (x_0, y_0). Sometimes we use the notations $M_{\mathcal{T}}(n; x_0, y_0)$ and $\mathbf{b}_{\mathcal{T}}(n; x_0, y_0)$ to make clear the graphic tile $\mathcal{T}$, the value of the n parameter, and the point (x_0, y_0) where the Tutte polynomial is being evaluated. The rest of symbols appearing undefined in this section have the same meaning as in Section 4.

The evaluation of $T(\mathcal{T}(m,n); x, y)$ at the point (x_0, y_0) can be performed as

$$T(\mathcal{T}(m,n); x_0, y_0) = \pi_1(M^m \cdot \mathbf{b}). \tag{10}$$

Here π_i is the projection of the ith coordinate. The computation can be performed using either floating point arithmetic and direct discrete exponentiation or arbitrary precision arithmetic.

5.1. Evaluations under floating point arithmetic and direct discrete exponentiation

If the computation is performed using floating point arithmetic and direct discrete exponentiation [22, page 2, 1.1.4], the time complexity of evaluating Eq. (10) is

$$O(s_a(n)^{2.3727} \cdot \log m), \tag{11}$$

where the term $s(n)^{2.3727}$ is the best known upper bound for the running time to perform a single matrix product of square matrices (the algorithm achieving this bound is an improvement by Vassilevska [28] to the Coppersmith and Winograd method [9]), and $\log m$ is the number of matrix multiplications required to evaluate M^m by the discrete exponentiation method.

As an application of the Cayley–Hamilton theorem, it is also possible to evaluate M^m by computing the eigenvalues and the characteristic polynomial of M [12, pages 194-197]. In this method, a polynomial in λ, say λ^m, is written in the form

$$\lambda^m = Q(\lambda) \cdot P_M(\lambda) + R(\lambda),$$

where $P_M(\lambda)$ is the characteristic polynomial of M, $Q(\lambda)$ is found by long division, and the residual polynomial $R(\lambda)$ is of degree $s_a(n)$ or less. Now, since $P_M(M) = 0$ then $M^m = R(M)$. Thus, to compute M^m we only need to evaluate $R(M)$.

The residual $R(M)$ can be determined by a method which first finds the eigenvalues $\lambda_1, \ldots \lambda_{s_a(n)}$ of M in $O(s_a(n)^{2.3727})$ time, then $\lambda_i^m = R(\lambda_i)$ for $i = 1, \ldots s_a(n)$ is a system of $s_a(n)$ linear equations which can be solved in $O(s_a(n)^{2.3727})$ time to get the coefficients of $R(\lambda)$. The evaluation of λ_i^m for $i = 1, \ldots, s_a(n)$ is performed using the discrete exponentiation and takes $O(s_a(n) \cdot \log m)$ time. Finally, $R(M)$ is evaluated by the Horner's rule which is asymptotically optimal [5, Chapter 5], and needs $O(s_a(n))$ matrix multiplications which can be performed in $O(s_a(n) \cdot s_a(n)^{2.3727}) = O(s_a(n)^{3.3727})$ time. Therefore, the total running time required to compute M^m by this technique is

$$O(s_a(n)\cdot\max\{s_a(n)^{2.3727},\log m\}). \tag{12}$$

Hence, by Expressions (11) and (12), the final running time complexity to evaluate Eq. (10) is

$$O(s_a(n)\cdot\min\{s_a(n)^{1.3727}\cdot\log m,\max\{s_a(n)^{2.3727},\log m\}\}). \tag{13}$$

The hardest part in the above expression is $s_a(n)$. Nevertheless, this number can be reduced (which implies a reduction in the dimension of the matrix M generated by the algorithm) as follows: In general, M is singular and so assume that the row M_i is a linear combination of some other rows, say for simplicity, $M_1, M_2, \ldots, M_k$ with $i \neq 1,\ldots,k$. Thus,

$$M_i = \sum_{j=1}^{k}\beta_j M_j \tag{14}$$

for values $\beta_1 \neq 0,\ldots,\beta_k \neq 0$. Then, in every row M_ℓ with $\ell \neq i$, we can replace (recall Eq. (5)) the entry $M_{\ell,j}$ by $M_{\ell,j}+\beta_j M_{\ell,i}$ for $j=1,\ldots,k$, and drop the ith row and column of the matrix M as well as the ith element of the vector $\mathbf{b}$ without altering Eq. (10). This is a *minor reduction* of M at i. If we represent by M_{red} the non-singular matrix obtained from successive minor reductions on M until no additional minor reduction could be performed, then the following result is straightforward.

Proposition 5.1. *There exist a non-singular matrix M_{red} and a vector $\mathbf{b}_{red}$ both of size rank(M) such that $\pi_1(M^m\cdot\mathbf{b}) = \pi_1((M_{red})^m\cdot\mathbf{b}_{red})$ for every positive integer m.*

Thus, the term $s_a(n)$ in Expression (13) can be replaced by rank(M). To illustrate the effect of minor reductions on $s_a(n)$, in the columns fifth and sixth of Table 1 we show, respectively, the ranks of $M_{\mathcal{T}_4}(n;2,1)$ and $M_{\mathcal{T}_4}(n;2,0)$ for the evaluations performed on $\mathcal{T}_4$. As we can see $\text{rank}(M_{\mathcal{T}_4}(n;2,1))/s_a(n)\approx 1$ independently of the value of n and so minor reductions did not produce any improvement. On the other hand, the quotient $\text{rank}(M_{\mathcal{T}_4}(n;2,0))/s_a(n)$ goes from 1 (for $n=1$) to $315/3948 = 0.08$ (for $n=8$), and thus, minor reductions yield smaller matrices as n is increased. The final effect of minor reductions, in general, depends upon the graphic tile and the point where the Tutte polynomial evaluation is performed.

To obtain M_{red}, a base of the linear space spanned by $M\cdot\mathbf{x}=\mathbf{0}$ must be computed. The running time is $O(s_a(n)^{2.3727})$ [15] which is the time needed to solve the homogeneous system of equations. A faster heuristic, which produced acceptable results in all our experiments, consists in making minor reductions to eliminate repeated rows in M (i.e., when $k=1$ in Eq. (14)). The heuristic can be implemented in $O(s_a(n)\cdot\log s_a(n))$ time which is the time complexity needed to sort in lexicographical order the rows in M. In the experiments with $\mathcal{T}_4$, the sizes of the reduced matrices $(M_{\mathcal{T}_4}(n;2,0))_{red}$ and $(M_{\mathcal{T}_4}(n;2,1))_{red}$ were close to $s_a(n)$. For $(M_{\mathcal{T}_4}(n;2,0))_{red}$ with $n=1,\ldots,10$, the sizes were 1, 2, 4, 11, 30, 87, 254, 755, 2249 and 6757, respectively. Comparing with column sixth of Table 1, we observe that the heuristic did not found the best values but the time complexity was lower.

5.2. Evaluations under arbitrary precision arithmetic

Now, we analyze the time complexity to obtain Tutte polynomial evaluations under arbitrary precision arithmetic. Our results will be also useful to evaluate the limit of Expression (10) as m tends to infinity. We first recall some definitions.

Let $A=(A_{ij})$ be a square real matrix of size k. A is said to be *positive* (*non-negative*) if each entry A_{ij} in A is greater than (greater than or equal to) zero. A directed graph G is said to be *strongly connected* if for every edge $ij\in E(G)$ there exists a directed path from i to j. Let G_A be the directed graph with k vertices, and having an edge from the vertex i to the vertex j whenever $A_{ij}>0$. The matrix A is said to be *irreducible* if and only if G_A is strongly connected. The *period* of an index i is the greatest common divisor of all natural numbers ℓ such that $(A^\ell)_{ii}>0$. When A is irreducible, the period of every index is the same and it is called the *period* of A [17, page 16]. If the period is equal to 1, A is *aperiodic*. Finally, A is *primitive* if it is non-negative and its ℓth power is positive for some natural number ℓ (i.e., the same ℓ works for all pair of indexes). It can be shown that primitive matrices are the same as irreducible aperiodic non-negative matrices [15, Theorem 8.5.3].

Theorem 5.2. *The matrix $M_{\mathcal{T}}(n;x_0,y_0)$ is primitive for every pair (x_0,y_0) with $x_0\geqslant 0$ and $y_0\geqslant 0$.*

Proof. Let $M=M_{\mathcal{T}}(n;x_0,y_0)$ for $x_0\geqslant 0$ and $y_0\geqslant 0$. We prove that M is non-negative, aperiodic and irreducible.

Every coefficient of $P_{i,j}(x,y)$ in Expression (8) is non-negative since it counts the number of times that the reduced Tutte polynomial of some residual graph appears multiplying some specific monomial $x^\alpha\cdot y^\beta$. Hence, $P_{i,j}(x_0,y_0)\geqslant 0$ and so M is non-negative.

Consider now the graph G_M whose vertices correspond to residual graphs. It is easy to check that $P_{i,j}(x_0,y_0)>0$ if and only if the expression for $T(\sum_{k=1}^{m}L_k+R_i;x,y)$ contains $T(\sum_{k=1}^{m-1}L_k+R_j;x,y)$ (see Eq. (7)). Hence, $P_{1,j}(x_0,y_0)>0$ for every $j=2,\ldots,s$ and so there are paths from R_1 to the rest of the vertices in G_M. On the other hand, R_1 always appears as a residual graph when $\sum_{k=1}^{m}L_k+R_i$ is processed since $R_1=B_k$. Therefore, there are paths from every vertex in G_M to R_1. In other words, G_M is strongly connected and so M is irreducible.

Since M is irreducible then the period of every index is the same. For the index $i=1$, $M_{1,1}>0$ and so its period is equal to 1. Hence, M is aperiodic. Thus, the result follows. □

Table 2
Numerical evaluations of $T(\mathcal{T}_4(n,n);x,y)$ against the results in [6].

n	$T(\mathcal{T}_4(n,n);2,1)$ Calkin et al. [6]	$T(\mathcal{T}_4(n,n);2,0)$ Calkin et al. [6]	$T(\mathcal{T}_4(n,n);2,1)$	$T(\mathcal{T}_4(n,n);2,0)$
1	15	14	15	14
2	3102	2398	3102	2398
3	8,790,016	5,015,972	8,790,016	5,015,972
4	3.41008617... ...4080000e+11	1.28091434... ...2660000e+11	341008617408	128091434266
5	1.81075508... ...2420676e+17	3.99318561... ...3821266e+16	1810755082... ...42067552	3993185613... ...8212664
6	1.31592738... ...9374152e+24	1.51966368... ...2749935e+23	1315927389... ...3741520341... ...13856	1519663682... ...7499347493... ...7668
7	–	–	1308775232 ... ...7481758020... ...9987036404... ...864	7059965159... ...4554546403... ...07807067492
8	–	–	1781359755... ...8513208864... ...3635627145... ...305047963624	4003910412 ... ...3439212956... ...7992528033... ...2950062686
9	–	–		2771997268... ...7144020161... ...8769518884... ...2216504904... ...4889741764
10	–	–	–	see the text

The above result guarantees that the matrix M satisfies all the conditions of the Perron–Frobenius theorem as it is stated in [15, Theorem 8.5.1]. This has several consequences and the most relevant for our purpose are

1. The *spectral ratio* $\rho(M)$ (i.e., the supremum of the absolute value of the eigen-values of M) is a positive real number.
2. $\lim_{m\to\infty}(M^m)_{i,j}/\rho(M)^m \leqslant C$ for every pair of indexes i,j and some positive constant real number C. Thus, for large values of m the number of binary digits to represent the entries in M^m is $O(\log \rho(M)^m) = O(m)$. Therefore, in an arbitrary precision arithmetic system the time complexity to evaluate Eq. (10) is the Expression (13) multiplied by m:

$$O(m \cdot s_a(n) \cdot \min\{s_a(n)^{1.3727} \cdot \log m, \max\{s_a(n)^{2.3727}, \log m\}\}). \tag{15}$$

5.3. Numerical results for the square lattice

By using the results in the two previous subsections, we have evaluated the Tutte polynomial of fragments of the Archimedian tilings whose graphic tiles have been introduced in this paper. We present now the numerical results of the evaluations performed for the Tutte polynomial of $\mathcal{T}_4$, and compare them with the numerical results obtained by Calkin et al. [6].[1] In that paper, the evaluations were performed in a floating point arithmetic system and the values obtained at the points $(2,1)$ (spanning forests) and $(2,0)$ (acyclic orientations) appear in the columns second and third of Table 2. In the columns fourth and fifth appear our evaluations, performed in arbitrary precision arithmetic, and thus our results are exact. In the column fifth, for $n = 10$ we obtain

234276054730087485045064123957219978143922947025512224055818 8

acyclic orientations.

The technique applied in [6] is quite similar to ours, they found a transfer matrix that works like our matrix M in Eq. (10), but instead of using the deletion–contraction recurrence for the Tutte polynomial, they used the definition obtained from the rank polynomial.

Our numerical evaluations were performed in a desktop computer with two processors Intel Core i3.2129 (3 Mb Cache, 20 Gb Ram and 3.30 GHz), and advanced 4 steps more that Calkin et al. [6] even that we did the computations in an arbitrary precision arithmetic system. The dimension of the transfer matrix used by Calkin et al. [6] is the $(n+1)$th Catalan number (see Table 1), but they did not mention the computer technology used in 2003. By assuming that the computer we use today is ten times faster than theirs, and considering that for $n = 6$ their matrix has dimension 429, it is not reasonable to think that they could reach $n = 8$, running their programs in our computer, since it requires the construction of a matrix of dimension 4862 which has 128 times more entries than their matrix for $n = 6$. But we reached $n = 10$; this was possible by the opti-

[1] In this and other related papers, the variables m and n denote the maximum number of vertices in paths along the two orthogonal dimensions of the square lattice, respectively. As a consequence of the definition of graphic tile our variables m and n count edges instead of vertices.

mizations explained in this section. From the presentation given in [6], it is not evident that similar optimizations could be done in their approach.

On the other hand, the main purpose of Calkin et al. [6] was to find asymptotic bounds on the number of spanning of forests, $f(n)$, and the number of acyclic orientations, $\alpha(n)$, of $\mathcal{T}_4(n,n)$. They based their computations on a result similar to Theorem 5.2 and gave the following bounds:

$$3.64497 \leqslant \lim_{n\to\infty} f(n)^{1/n^2} \leqslant 3.74101.$$

$$3.41358 \leqslant \lim_{n\to\infty} \alpha(n)^{1/n^2} \leqslant 3.55449.$$

After a careful analysis, we concluded that we were able to improve these bounds by translating their work to our method. In fact, by working with $m = 7$ we reproduced the above bounds. To compute the lower bounds, we had to do several adjustments to our technique to compute the Tutte polynomial of Fan-graphs [6, Section 6] because it was necessary to combine two types of tiles in the same graphic tile. We do not consider necessary to give all the details since the translation between both approaches is immediate. Our final bounds are given in the following result.

Theorem 5.3

$$3.65166 \leqslant \lim_{n\to\infty} f(n)^{1/n^2} \leqslant 3.73635.$$

$$3.42351 \leqslant \lim_{n\to\infty} \alpha(n)^{1/n^2} \leqslant 3.5477.$$

Just recently, Mani [1] has found that $\lim_{n\to\infty} f(n)^{1/n^2} \leqslant 3.705603$, which improves our upper bound; it seems unfeasible that our method can enhance that value. The rest of the bounds given in Theorem 5.3 are the best known for the asymptotics of the number of spanning forests and the number of acyclic orientations of the square lattice.

6. Conclusions

We have used the deletion–contraction recurrence for the Tutte polynomial to design an algorithm to compute the Tutte polynomial of large fragments of Archimedean tilings by squares, triangles, hexagons and combinations thereof. These fragments have nm copies of a basic tile pattern placed in a rectangular arrangement where $n \leqslant 10$ and m is arbitrarily large. We have also analyzed the complexity of our algorithm and some factors that might affect its performance. Then, we have studied the problem of obtaining Tutte polynomial evaluations from the symbolic expressions generated by the algorithm. These numerical results play an important role from the statistical physics perspective where Archimedean tilings are important structures as vertex model and the Tutte polynomial appears as the partition function of the q-state Potts model.

In the particular case of the square lattice, we have enhanced the method given by Calkin el al. in [6]. Our technique is new, but the final way to compute Tutte polynomials is similar to their approach. However, our method generates smaller matrices at least for $n \leqslant 14$ and we may do further improvements. In this way, we have made evaluations for larger square lattices and we have improved the asymptotic bounds that they gave, obtaining the best known lower bound for the asymptotics of the number of spanning forests, and the lower and upper bounds for the asymptotics of the number of acyclic orientations of the square lattice.

The definition of graphic tile is general enough to consider all Archimedean tilings by squares, triangles and hexagons, including the three regular tilings, and variations such as tilings made by cubes. Certainly there are alternative definitions that could be used to extend the possibilities, but one thing should be considered: Proposition 4.1 (or something equivalent) should hold to guarantee that the number of residual graphs involved in Theorem 4.2 is finite.

Pearce et al. [19] have developed a general method to compute Tutte polynomials which is considered the best. It is based on heuristics to select the next edge to be contracted or deleted, in such a way that a minimum number of isomorphic graphs results. For fragments of tilings generated by graphic tiles, a similar best ordering of the edges must exist, in such a way that a minimum value for $s_a(n)$ could be guaranteed. Discovering the structure of such orderings would let us compute Tutte polynomials of larger fragments.

Acknowledgement

The authors thank ABACUS-CINVESTAV, CONACyT grant EDOMEX-2011-C01-165873.

References

[1] A.P. Mani, On some Tutte polynomial sequences in the square lattice, J. Combin. Theory Ser. B 102 (2012) 436–453.
[2] N.L. Biggs, Algebraic Graph Theory, second ed., Cambridge Univ. Press, 1993.
[3] B. Bollobás, Modern Graph Theory, Graduate Text in Mathematics, vol. 184, Springer-Verlag, 1998.
[4] T. Brylawski, J. Oxley, The Tutte polynomial and its applications, in: N. White (ed.), Matroid Applications, Encyclopedia of Mathematics and Its Applications, vol. 40, 1992, pp. 123–225.

[5] P. Bürgisser, M. Clausen, M.A. Shokrollahi, Algebraic Complexity Theory, Springer-Verlag, Berlin, 1997.
[6] N. Calkin, C. Merino, S. Noble, M. Noy, Improved bounds for the number of forests and acyclic orientations in the square lattice, Electr. J. Comb. 10 (R4) (2003) 1–18.
[7] S.C. Chang, J. Salas, R. Shrock, Exact Potts model partition functions for strips of the square lattice, J. Stat. Phys. 107 (5/6) (2002) 1207–1253.
[8] S.C. Chang, J.L. Jacobsen, J. Salas, R. Shrock, Exact Potts model partition functions for strips of the triangular lattice, J. Stat. Phys. 141 (3/4) (2004) 763–823.
[9] D. Coppersmith, S. Winograd, Matrix multiplication via arithmetic progressions, J. Symbolic Comp. 9 (3) (1990) 251–280.
[10] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman and Company, 1979.
[11] A.J. Goodall, Graph polynomials and Tutte–Gröthendieck invariants: an application of elementary finite Fourier analysis. arXiv:0806.4848v1.
[12] C.W. Groetsch, J.T. King, Matrix Methods and Applications, Prentice Hall, 1988.
[13] B. Grünbaum, G.C. Shepard, Tilings and Patterns, W.H. Freeman and Company, 1986.
[14] G. Haggard, D.J. Pearce, G. Royle, Computing Tutte polynomials, AMC Trans. Math. Softw. 37 (3) (2010), http://dx.doi.org/10.1145/1824801.1824802 (Article 24).
[15] R.A. Horn, C.R. Johnson, Matrix Analysis, Cambridge Univ. Press, 1985.
[16] F. Jaeger, D. Vertigan, D.J.A. Welsh, On the computational complexity of the Jones and Tutte polynomials, Math. Proc. Cambridge Philos. Soc. 108 (1) (1990) 35–53.
[17] B.P. Kitchens, Symbolic Dynamics: One-sided, Two-sided and Countable State Markov Shifts, Springer Verlag (Universitext), 1998.
[18] A Tour of NTL: summary of NTL's main modules. http://www.shoup.net/ntl/doc/tour-modules.html.
[19] D.J. Pearce, G. Haggard, G. Royle, Edge-selection heuristics for computing Tutte polynomials, Chicago J. Theor. Comput. Sci. 2010 (2010) (Article 6).
[20] G.F. Royle, Computing the Tutte polynomial of sparse graphs, Technical Report CORR 88–35, University of Waterloo, 1988.
[21] K. Sekine, H. Imai, S. Tani, Computing the Tutte polynomial of a graph of moderate size, Lecture Notes Comput. Sci. 1004 (1995) 224–233.
[22] A. Shen, Algorithms and Programming Problems and Solutions, Birkhäuser, 1997.
[23] A.D. Sokal, The multivariate Tutte polynomial (alias Potts model) for graphs and matroids, in: Surveys in Combinatorics, Lecture Note Series, London Mathematical Society, vol. 327, 2005, pp. 173–226.
[24] R.P. Stanley, Enumerative Combinatorics, Cambridge Studies in Advanced Mathematics 62, vol. 2, Cambridge Univ. Press, 1999.
[25] D.J.A. Welsh, Percolation and the random cluster model: combinatorial and algorithmic problems, in: M. Habib, C. McDiarmid, J. Ramírez-Alfonsín, B. Reed (Eds.), Probabilistic Methods for Algorithmic Discrete Mathematics, 1991.
[26] D.J.A. Welsh, Complexity: Knots, Colourings and Counting, Cambridge Univ. Press, 1993.
[27] D.J.A. Welsh, C. Merino, The Potts model and the Tutte polynomial, J. Math. Phys. 41 (3) (2000) 1127–1152.
[28] V. Vassilevska, Breaking the Coppersmith–Winograd barrier, Manuscript, 2011.
[29] D. Vertigan, The computational complexity of Tutte invariants for planar graphs, SIAM J. Comput. 35 (3) (2006) 690–712.