Acta Polytechnica CTU Proceedings **13**:135–141, 2017
doi:10.14311/APP.2017.13.0135

available online at http://ojs.cvut.cz/ojs/index.php/app

OPTIMIZATION-BASED APPROACH TO TILING OF FINITE AREAS WITH ARBITRARY SETS OF WANG TILES

Marek Tyburec*, Jan Zeman

Czech Technical University in Prague, Faculty of Civil Engineering, Department of Mechanics, Thákurova 7, 166 29 Prague 6, Czech Republic

* corresponding author: marek.tyburec@fsv.cvut.cz

Abstract. Wang tiles proved to be a convenient tool for the design of aperiodic tilings in computer graphics and in materials engineering. While there are several algorithms for generation of finite-sized tilings, they exploit the specific structure of individual tile sets, which prevents their general usage. In this contribution, we reformulate the NP-complete tiling generation problem as a binary linear program, together with its linear and semidefinite relaxations suitable for the branch and bound method. Finally, we assess the performance of the established formulations on generations of several aperiodic tilings reported in the literature, and conclude that the linear relaxation is better suited for the problem.

Keywords: Wang tiles, binary and continuous linear programming, semidefinite programming, aperiodic tilings, relaxation.

1. Introduction

Wang tiles, squares with colored edges, were invented by and named in honor of Hao Wang, originally serving as a tool for studying the ∀∃∀ decidability problem of the predicate calculus [1]. Wang showed that the decidability problem is equivalent to the domino problem: assume a set of non-rotatable unit-sized (Wang) tiles with edges colored according to the ∀∃∀ problem. If it is feasible to cover the infinite plane by translating copies of the tiles, such that all their vertices lie at the integer lattice points of the plane and the adjoining edges share the same color, the problem is said to be solvable; unsolvable otherwise. For an illustration of a sample 2×3 Wang tiling, see Fig. 1.

In [2], Wang made a *fundamental conjecture* stating that the domino problem is solvable if and only if the tiling was *periodic*, i.e., if there existed a rectangular region of the tiling with identically colored horizontal, and vertical edges, respectively.

A year later, Kahr reduced the Turing machine halting problem [3, 4] into the origin-constrained domino problem [5], which implies the domino problem is also undecidable. This can be illustrated by introducing a Turing machine for each tile set, halting only if the domino problem is unsolvable. However, there is an infinite number of such tile sets, making the domino problem undecidable as well, and forbidding existence of a general finite algorithm for the generation of *infinite* tilings.

1.1. Aperiodic Tile Sets

Undecidability of the domino problem was also proved by Wang's student Berger, who used the principle of expanding squares for developing the sets of 20, 426 [6] and 104 tiles [7] that admit only *aperiodic* tilings of the infinite plane, contrary to the Wang fundamental conjecture.

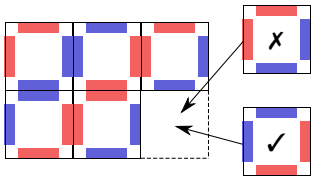


Figure 1. Periodic tiling composed of 2 Wang tiles over 2 colors.

Aiming to simplify the Berger's proof, smaller aperiodic sets of Wang tiles have been constructed. In 1966, Läuchli sent to professor Wang an aperiodic set of 40 tiles over 16 colors, but it remained unpublished until 1975 [8]. Meanwhile, unaware of Läuchli's result, Knuth simplified Berger's set to 92 tiles over 26 colors [9]; and Robinson developed sets of 104 and 52 tiles in 1967 [10], of 56 tiles over 12 colors in 1971 [11], and marked existence of a set of 35 tiles [11].

In 1973, Penrose developed a new approach based on kites and darts tiling, leading to a set of 34 tiles. Robinson, being in contact with Penrose, modified the Penrose's approach to reach a reduced set of 32 tiles over 16 colors [12]. Using the same technique together with Penrose rhombs tiling, Grünbaum obtained a set of 24 tiles over 9 colors in 1987 [12].

Another two tile sets were discovered due to Ammann. In 1978, he used Ammann bars to reach 16 tiles over 6 colors [13]. Building on the Ammann's A2 tiling, see, e.g., [12], Robinson obtained a set of 24 tiles over 24 colors in 1977.

Subsequent size reduction of the smallest aperiodic set occurred in 1996, as Kari developed a new method based on Mealy machines multiplying Beatty sequences, and presented a set of 14 tiles over 6 colors [14]. Čulík, using the same approach, reduced the set even further to 13 tiles over 5 colors [15].

The search for the minimal aperiodic set is concluded by Jeandel, who used a brute-force enumeration approach to find aperiodic sets of 11 tiles over 4 and 5 colors; and proved both that there does not exist an aperiodic set with 10 or fewer tiles and with less than 4 colors [16].

In addition to the classical Wang tiles, in 2006, Lagae introduced a subset of the Wang tiles, the *corner* tiles, with the connectivity information stored in the colored corners instead of the edges [17]. In the same year, he constructed an aperiodic set of 44 corner tiles over 6 colors [18]. The set was further simplified by Nurmi to 30 corner tiles over 6 colors [19]. Note that because the corner tiles can be straightforwardly converted to the edge-based Wang tiles [17], our approach naturally extends to the corner tiles, see Section 4 for a specific example.

1.2. Selected Applications

Due to the ability of some tile sets to generate aperiodic patterns, Wang tiles received a broad interest among disciplines. Also, their original significance for automated theorem proving [2] was supplemented by proofs in cellular automata theory [20].

In 2003, Cohen *et al.* introduced Wang tiles to the computer graphics community [21]. Since then, Wang tiles have been successfully used, e.g., for efficient synthesis of stochastic textures or for generation of Poisson disk distributions.

Molecular realization of Wang tiles is due to Winfree, who introduced a self-assembly of biological nanostructures into aperiodic crystals [22].

Novák was probably the first one who used Wang tiles in materials engineering for efficient compression of microstructures [23]; Doškář for their reconstruction [24]. Doškář further generated large stochastic samples of the Alporas® foam and its finite-element representations in [25]. In [26], Tyburec used Wang tiles to describe modular assembly of structures, whereas both the topology and arrangement of modules were subjects of optimization.

1.3. Tiling Generation Algorithms

Existing tiling generation algorithms are designed specifically to the tile sets they handle. In computer graphics, for example, it is essential to generate visually appealing patterns quickly, which is best achieved with stochastic tile sets. Tiling a finite area has then a $\mathcal{O}(n)$ complexity, adopting, e.g., the stochastic tiling [21], or the hash-based direct stochastic tiling [27] algorithms. The latter algorithm allows straightforward definition of boundary conditions.

In the case of mathematical logic, it is crucial to prove that an investigated tile set can tile the whole *infinite* plane, and does so only aperiodically. As the problem is undecidable, no general algorithm exists, and thus all the algorithms need to exploit the specific structure of each individual (family of) tile sets. Although the algorithms are fast, they remain tile set dependent. All tilings of finite-sized areas are restricted to be subsets of the infinite plane, which is not required in the finite domain, and it is almost impossible to introduce boundary conditions. On top of that, there provably exist aperiodic tile sets that admit only non-recursive tiling [28, 29], forbidding design of any problem-specific algorithm to tile the infinite plane.

To the authors' knowledge, the only method that has been used for tiling of finite-sized areas by arbitrary tile sets is the backtracking algorithm, see [30]. Although the algorithm is general, it is generally inefficient, as it commonly creates impossible assemblies too early. Moreover, distant boundary conditions make the problem more difficult to solve.

1.4. Contributions

This paper aims to overcome the shortcomings of the methods outlined in the previous section, and to develop an approach that handles tiling of finite-sized areas using arbitrary tile sets, together with a straightforward approach to define edge- or tile-based boundary conditions.

Our exposition is structured as follows. In Section 2, we develop a binary linear programming representation of the tiling generation problem. The formulation is relaxed, and its linear and semidefinite approximations are presented. In order to show generality of the formulations, we introduce their extensions to solve the tile-packing problem, to prove that a tile set can not tile the plane, and to prove that a tile set admits periodic tiling. The relaxations are finally employed in a branch and bound method in Section 3 and their performance is assessed in Section 4.

2. Optimization Problem Formulation

2.1. Valid Tiling

Consider a finite rectangular area $\mathcal{A}$ of the size $n_{\mathrm{t},h} \times n_{\mathrm{t},w}$, with $n_{\mathrm{t},h}$ and $n_{\mathrm{t},w}$ denoting its height and width, respectively. Placing Wang tiles (of a unit size) from the tile set $\mathcal{T}$, such that their vertices lie at the integer lattice points of $\mathcal{A}$, we obtain a tiling with $n_{\mathrm{t},h}$ rows and $n_{\mathrm{t},w}$ columns of tiles.

Each tile $k \in \{1, \ldots, n_{\mathrm{t}}\}$ from the tile set $\mathcal{T}$, with n_{t} denoting the number of tiles, is described by the 4-tuple (n_k, w_k, s_k, e_k), assigning color codes $\mathcal{C}$ to the north, west, south, and east edge of the tile, respectively.

In any valid tiling, for any two adjacent tiles k and ℓ, with the tile ℓ being placed in the east of k, it holds that

$$e_k = w_\ell. \tag{1}$$

Similarly, for any two vertically adjacent tiles k and m, with the tile m being placed in the south of k, it stands that

$$s_k = n_m. \tag{2}$$

Both the cases are illustrated in Fig. 2.

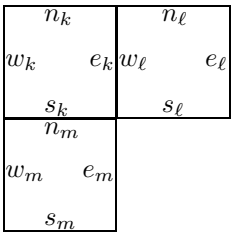


FIGURE 2. Horizontal, and vertical connectivity among tiles k-ℓ, and k-m, respectively.

2.2. Binary Linear Programming Formulation

Let $\mathbf{x} \in \{0,1\}^{n_{\mathrm{t},h} \cdot n_{\mathrm{t},w} \cdot n_{\mathrm{t}}}$ denote a binary vector describing the placement of individual tiles within $\mathcal{A}$. The vector consists of all the $x_{i,j,k}$ variables, where $i \in \{1, \ldots, n_{\mathrm{t},h}\}$ and $j \in \{1, \ldots, n_{\mathrm{t},w}\}$ are the row and column iterators, respectively. Let us also define

$$x_{i,j,k} = \begin{cases} 0 & \text{if the tile } k \text{ is } \textit{not} \text{ at } (i,j), \\ 1 & \text{if the tile } k \text{ is at } (i,j). \end{cases} \tag{3}$$

Because each position is occupied by a single tile, we have

$$\sum_{k \in \mathcal{T}} x_{i,j,k} = 1, \quad \forall i \in \mathcal{H}, \forall j \in \mathcal{W}. \tag{4}$$

The compatibility constraint (1) can be written in terms of $\mathbf{x}$ after realizing that the number of tiles at (i,j) that have east edge colored by $c \in \mathcal{C}$ has to be equal to the number of tiles at $(i, j+1)$ with west edge colored also by c. Consequently, we have

$$\sum_{k \in \mathcal{T}} x_{i,j,k}[e_k = c] - \sum_{k \in \mathcal{T}} x_{i,j+1,k}[w_k = c] = 0. \tag{5}$$

As the sums in Eq. (5) are either equal to 0 or 1, resulting from Eq. (4), only two options are possible: If the edge is colored by c, the relation simplifies to

$$1 - 1 = 0, \tag{6}$$

otherwise, it equals to

$$0 - 0 = 0, \tag{7}$$

showing that (5) remains valid in both cases.

After applying the same approach to Eq. (2) and writing constraints through the entire area $\mathcal{A}$, we obtain the following binary linear program:

$$\min_{\mathbf{x}} \mathbf{0} \tag{8a}$$

$$\text{s.t.} \sum_{k \in \mathcal{T}} x_{i,j,k}[e_k = c] - \sum_{k \in \mathcal{T}} x_{i,j+1,k}[w_k = c] = 0, \tag{8b}$$
$$\forall c \in \mathcal{C}, \forall i \in \mathcal{H}, \forall j \in \mathcal{W} \setminus \{n_{\mathrm{t},w}\},$$

$$\sum_{k \in \mathcal{T}} x_{i,j,k}[s_k = c] - \sum_{k \in \mathcal{T}} x_{i+1,j,k}[n_k = c] = 0, \tag{8c}$$
$$\forall c \in \mathcal{C}, \forall j \in \mathcal{W}, \forall i \in \mathcal{H} \setminus \{n_{\mathrm{t},h}\},$$

$$\sum_{k \in \mathcal{T}} x_{i,j,k} = 1, \quad \forall i \in \mathcal{H}, \forall j \in \mathcal{W}, \tag{8d}$$

$$x_{i,j,k} \in \{0,1\}, \quad \forall i \in \mathcal{H}, \forall j \in \mathcal{W}, \forall k \in \mathcal{T}. \tag{8e}$$

Despite the program (8) being formulated straightforwardly, it is hard to solve in this original form due to the (non-convex) integrality constraint (8e). Note that the problem is NP-complete because of its combinatorial nature [31].

2.3. Linear Programming Relaxation

In order to make the problem (8) easier to solve, let us *relax* the integrality constraint (8e) into a linear form

$$0 \leq x_{i,j,k} \leq 1, \quad \forall i \in \mathcal{H}, \forall j \in \mathcal{W}, \forall k \in \mathcal{T}. \tag{9}$$

Clearly, Eq. (9) allows for all configurations of (8e), but additionally also intermediate non-binary values. In addition, because the constraints in (8) are linear, the resulting approximation is *convex* and reads as

$$\min_{\mathbf{x}} \mathbf{0} \tag{10a}$$

$$\text{s.t.} \sum_{k \in \mathcal{T}} x_{i,j,k}[e_k = c] - \sum_{k \in \mathcal{T}} x_{i,j+1,k}[w_k = c] = 0, \tag{10b}$$
$$\forall c \in \mathcal{C}, \forall i \in \mathcal{H}, \forall j \in \mathcal{W} \setminus \{n_{\mathrm{t},w}\},$$

$$\sum_{k \in \mathcal{T}} x_{i,j,k}[s_k = c] - \sum_{k \in \mathcal{T}} x_{i+1,j,k}[n_k = c] = 0, \tag{10c}$$
$$\forall c \in \mathcal{C}, \forall j \in \mathcal{W}, \forall i \in \mathcal{H} \setminus \{n_{\mathrm{t},h}\},$$

$$\sum_{k \in \mathcal{T}} x_{i,j,k} = 1, \quad \forall i \in \mathcal{H}, \forall j \in \mathcal{W}, \tag{10d}$$

$$0 \leq x_{i,j,k} \leq 1, \quad \forall i \in \mathcal{H}, \forall j \in \mathcal{W}, \forall k \in \mathcal{T}. \tag{10e}$$

2.4. Semidefinite Programming Relaxation

The integrality constraint (8e) can be rewritten using the non-convex quadratic constraint

$$x_{i,j,k}^2 - x_{i,j,k} = 0, \quad \forall i \in \mathcal{H}, \forall j \in \mathcal{W}, \forall k \in \mathcal{T}, \tag{11}$$

which is satisfied only if $x_{i,j,k} = \{0,1\}$, being thus equivalent to (8e). This quadratic form does not simplify the solution, because the problem is still NP-complete, but we will use it for the derivation of a semidefinite programming relaxation.

Let us now substitute the binary variables $\mathbf{x} \in \{0,1\}$ with $\mathbf{y} \in \{-1,1\}$, which requires

$$\mathbf{x} = \frac{1}{2}(\mathbf{y} + 1). \tag{12}$$

Inserting this into Eq. (3) yields

$$y_{i,j,k} = \begin{cases} -1 & \text{if the tile } k \text{ is } \textit{not} \text{ at } (i,j), \\ 1 & \text{if the tile } k \text{ is at } (i,j). \end{cases} \tag{13}$$

Consequently, we can write a non-convex quadratically-constrained optimization program in terms of $\mathbf{y}$

$$\min_{\mathbf{y}} \mathbf{0} \tag{14a}$$

$$\text{s.t.} \sum_{k\in\mathcal{T}} (y_{i,j,k}+1)[e_k = c] - \sum_{k\in\mathcal{T}} (y_{i,j+1,k}+1)[w_k = c] = 0, \tag{14b}$$
$$\forall c \in \mathcal{C}, \forall i \in \mathcal{H}, \forall j \in \mathcal{W} \setminus \{n_{\mathrm{t},w}\},$$

$$\sum_{k\in\mathcal{T}} (y_{i,j,k}+1)[s_k = c] - \sum_{k\in\mathcal{T}} (y_{i+1,j,k}+1)[n_k = c] = 0, \tag{14c}$$
$$\forall c \in \mathcal{C}, \forall j \in \mathcal{W}, \forall i \in \mathcal{H} \setminus \{n_{\mathrm{t},h}\},$$

$$\sum_{k\in\mathcal{T}} y_{i,j,k} = 2 - n_{\mathrm{t}}, \quad \forall i \in \mathcal{H}, \forall j \in \mathcal{W}, \tag{14d}$$

$$y_{i,j,k}^2 = 1, \quad \forall i \in \mathcal{H}, \forall j \in \mathcal{W}, \forall k \in \mathcal{T}. \tag{14e}$$

Let us further introduce a symmetric matrix $\mathbf{Y} \in \mathcal{S}^{n_{\mathrm{t},y}\cdot n_{\mathrm{t},x}\cdot n_{\mathrm{t}}}$ defined as

$$\mathbf{Y} = \mathbf{y}\mathbf{y}^{\mathrm{T}}. \tag{15}$$

The definition directly implies that any solution to the quadratically-constrained optimization problem (14) renders the matrix $\mathbf{Y}$ positive semidefinite, in our notation $\mathbf{Y} \succeq \mathbf{0}$, and the rank of the matrix $\mathbf{Y}$ equal to 1. Moreover, the definition of $\mathbf{y}$ constrains all the elements in the main diagonal of $\mathbf{Y}$ equal to 1. Consequently, another equivalent formulation to the problem (8) reads as

$$\min_{\mathbf{y},\mathbf{Y}} \mathbf{0} \tag{16a}$$

$$\text{s.t.} \sum_{k\in\mathcal{T}} (y_{i,j,k}+1)[e_k = c] - \sum_{k\in\mathcal{T}} (y_{i,j+1,k}+1)[w_k = c] = 0, \tag{16b}$$
$$\forall c \in \mathcal{C}, \forall i \in \mathcal{H}, \forall j \in \mathcal{W} \setminus \{n_{\mathrm{t},w}\},$$

$$\sum_{k\in\mathcal{T}} (y_{i,j,k}+1)[s_k = c] - \sum_{k\in\mathcal{T}} (y_{i+1,j,k}+1)[n_k = c] = 0, \tag{16c}$$
$$\forall c \in \mathcal{C}, \forall j \in \mathcal{W}, \forall i \in \mathcal{H} \setminus \{n_{\mathrm{t},h}\},$$

$$\sum_{k\in\mathcal{T}} y_{i,j,k} = 2 - n_{\mathrm{t}}, \quad \forall i \in \mathcal{H}, \forall j \in \mathcal{W}, \tag{16d}$$

$$\mathrm{diag}(\mathbf{Y}) = \mathbf{1}, \tag{16e}$$

$$\mathbf{Y} - \mathbf{y}\mathbf{y}^{\mathrm{T}} = \mathbf{0}. \tag{16f}$$

The only non-convex constraint (16f) can be relaxed, see, e.g., [32], into a convex form

$$\mathbf{Y} - \mathbf{y}\mathbf{y}^{\mathrm{T}} \succeq \mathbf{0}, \tag{17}$$

and based on the Schur complement lemma equivalently rewritten to

$$\begin{pmatrix} 1 & \mathbf{y}^{\mathrm{T}} \\ \mathbf{y} & \mathbf{Y} \end{pmatrix} \succeq \mathbf{0}. \tag{18}$$

Finally, the semidefinite programming relaxation of the binary linear program (8) reads as

$$\min_{\mathbf{y},\mathbf{Y}} \mathbf{0} \tag{19a}$$

$$\text{s.t.} \sum_{k\in\mathcal{T}} (y_{i,j,k}+1)[e_k = c] - \sum_{k\in\mathcal{T}} (y_{i,j+1,k}+1)[w_k = c] = 0, \tag{19b}$$
$$\forall c \in \mathcal{C}, \forall i \in \mathcal{H}, \forall j \in \mathcal{W} \setminus \{n_{\mathrm{t},w}\},$$

$$\sum_{k\in\mathcal{T}} (y_{i,j,k}+1)[s_k = c] - \sum_{k\in\mathcal{T}} (y_{i+1,j,k}+1)[n_k = c] = 0, \tag{19c}$$
$$\forall c \in \mathcal{C}, \forall j \in \mathcal{W}, \forall i \in \mathcal{H} \setminus \{n_{\mathrm{t},h}\},$$

$$\sum_{k\in\mathcal{T}} y_{i,j,k} = 2 - n_{\mathrm{t}}, \quad \forall i \in \mathcal{H}, \forall j \in \mathcal{W}, \tag{19d}$$

$$\mathrm{diag}(\mathbf{Y}) = \mathbf{1}, \tag{19e}$$

$$\begin{pmatrix} 1 & \mathbf{y}^{\mathrm{T}} \\ \mathbf{y} & \mathbf{Y} \end{pmatrix} \succeq \mathbf{0}, \tag{19f}$$

$$-\mathbf{1} \leq \mathbf{y} \leq \mathbf{1}. \tag{19g}$$

2.5. Extensions

Introduced formulations can include additional requirements for generated tilings. We list some of them below, but only in terms of $\mathbf{x}$, as substitution of Eq. (12) into developed equations is obvious.

2.5.1. Tile-Based Boundary Conditions

There are four types of tile-based boundary conditions. First, we can enforce placement of the tile $k \in \mathcal{T}$ at position (i, j):

$$x_{i,j,k} = 1, \quad i \in \mathcal{H}, j \in \mathcal{W}, k \in \mathcal{T}. \tag{20}$$

Conversely, avoiding the tile k at (i, j) requires

$$x_{i,j,k} = 0, \quad i \in \mathcal{H}, j \in \mathcal{W}, k \in \mathcal{T}. \tag{21}$$

The requirement that the same tile is placed at both (i, j) and (p, q) can be written as

$$x_{i,j,k} - x_{p,q,k} = 0, \quad \{i, p\} \in \mathcal{H}, \{j, q\} \in \mathcal{W}, \forall k \in \mathcal{T}. \tag{22}$$

Enforcing different tiles at (i, j) and (p, q) requires

$$x_{i,j,k} + x_{p,q,k} \leq 1, \quad \{i, p\} \in \mathcal{H}, \{j, q\} \in \mathcal{W}, \forall k \in \mathcal{T}. \tag{23}$$

2.5.2. Edge-Based Boundary Conditions

Starting from Eq. (5), we can also easily formulate edge-based boundary conditions. To constrain the north edge at (i, j) to the color $c \in \mathcal{C}$, we write

$$\sum_{k\in T} x_{i,j,k}[n_k = c] = 1, \quad i \in \mathcal{H}, j \in \mathcal{W}, c \in \mathcal{C}. \tag{24}$$

If the north edge at (i, j) has to differ from c, it holds that

$$\sum_{k\in T} x_{i,j,k}[n_k = c] = 0, \quad i \in \mathcal{H}, j \in \mathcal{W}, c \in \mathcal{C}. \tag{25}$$

Equal color c of two edges, e.g., of the north edge at (i,j) and of the west edge at (p,q), is provided by

$$\sum_{k\in T} x_{i,j,k}[n_k = c] - \sum_{k\in T} x_{p,q,k}[w_k = c] = 0, \quad \{i,p\} \in \mathcal{H}, \{j,q\} \in \mathcal{W}, \forall c \in \mathcal{C}. \tag{26}$$

Finally, different coloring of the north edge at (i,j) and the west edge at (p,q) is guaranteed through

$$\sum_{k\in T} x_{i,j,k}[n_k = c] + \sum_{k\in T} x_{p,q,k}[w_k = c] \leq 1, \quad \{i,p\} \in \mathcal{H}, \{j,q\} \in \mathcal{W}, \forall c \in \mathcal{C}. \tag{27}$$

2.5.3. Periodic Tiling

Periodic tiling is a tiling with equally colored both horizontal and also both vertical edges. Thus, periodic tiling secures tilability of the infinite plane. Building on the edge-based boundary conditions, a periodic tiling can be enforced by

$$\sum_{k\in T} x_{1,j,k}[n_k = c] - \sum_{k\in T} x_{n_{\mathrm{t},h},j,k}[s_k = c] = 0, \quad \forall j \in \mathcal{W}, \forall c \in \mathcal{C}, \tag{28a}$$

$$\sum_{k\in T} x_{i,1,k}[w_k = c] - \sum_{k\in T} x_{i,n_{\mathrm{t},w},k}[e_k = c] = 0, \quad \forall i \in \mathcal{H}, \forall c \in \mathcal{C}. \tag{28b}$$

2.5.4. Tile Packing Problem

The tile packing problem, see, e.g., [30], consists in generation of a tiling with periodic boundary conditions, and each tile has to be placed exactly once.

In order to describe the problem in our framework, we just need to use Eq. (28) together with

$$\sum_{i\in\mathcal{H}}\sum_{j\in\mathcal{W}} x_{i,j,k} = 1, \quad \forall k \in \mathcal{T}. \tag{29}$$

2.5.5. Objective Function

Obviously, the developed formulations can contain arbitrary convex objective function. For instance, we can minimize the maximum occurrences of tiles, compose the tiling such that it fits some pattern, etc.

3. The Branch and Bound Method

The whole integer design space, i.e., $\{0,1\}$ for the linear (10) and $\{-1,1\}$ for the semidefinite (19) approximation, can be described using a *tree* data structure, with each *node* corresponding to variables $\mathbf{x}$, or $\mathbf{y}$. However, as there is exactly one tile per position, it is advantageous to use the n_t-ary tree representation instead of the binary one, so that the nodes correspond to the positions and branch into n_t children, avoiding solution to infeasible programs placing multiple tiles at the same position.

Solution to the developed approximations, (10) or (19), corresponds to exploring the *root node* of the tree. Unfortunately, because the approximations widen the feasible design space, their solution does not assure to solve the original problem (8) due to the design variables being allowed to take non-integer values. In order to overcome that, we *branch* some nodes of the tree, i.e., gradually fix all variables that correspond to the specific tile position to all possible combinations of integer values, n_t in our case.

The relaxation is solved in each node, *bounding* the problem[1], hence the name of the method *branch and bound* [33]. The branching and bounding continues until the optimal (feasible) solution to (8) is found. If no such solution exists, the solution space is proven to be empty and the tiling generation problem infeasible. Without boundary conditions, infeasibility proves the tile set does not tile infinite plane.

3.1. Branching Rule

In our implementation, we firstly branch the most promising nodes, in which the integer infeasibility of the convex relaxation is minimal. For the linear relaxation, we have

$$I_\mathrm{inf} = \sum_{i\in\mathcal{H}}\sum_{j\in\mathcal{W}}\sum_{k\in\mathcal{T}} (x_{i,j,k} - x_{i,j,k}^2), \tag{30}$$

or

$$I_\mathrm{inf} = n_{\mathrm{t},h} n_{\mathrm{t},w} n_\mathrm{t} - \sum_{i\in\mathcal{H}}\sum_{j\in\mathcal{W}}\sum_{k\in\mathcal{T}} y_{i,j,k}^2 \tag{31}$$

for the semidefinite relaxation. If $I_\mathrm{inf} = 0$, a feasible solution to the original problem was found.

3.2. Variables Ordering

In each node to be branched, we have to define variables that will be fixed. It seems to be advantageous to fix the variables corresponding to the most difficult position, i.e., to the position with the highest integer infeasibility. In the case of the linear relaxation, the to-be-fixed position (i,j) requires

$$\max_{\forall i\in\mathcal{H},\forall j\in\mathcal{W}} \sum_{k\in\mathcal{T}} (x_{i,j,k} - x_{i,j,k}^2). \tag{32}$$

Similarly, for the semidefinite relaxation we write

$$\max_{\forall i\in\mathcal{H},\forall j\in\mathcal{W}} (n_\mathrm{t} - \sum_{k\in\mathcal{T}} y_{i,j,k}^2). \tag{33}$$

4. Examples

Building upon the previous sections, we implemented a custom branch and bound algorithm in C++. The linear relaxations are solved using Gurobi [34], the semidefinite programming relaxations are optimized by Mosek [35]. As Gurobi contains a build-in state-of-the-art branch and cut algorithm, usable only for the linear relaxations, we also measured its performance.

Five tile sets were tested altogether: Jeandel's 11 tiles over 4 colors [16], Čulík's 13 tiles [15], Ammann's 16 tiles [12], Lauchli's 40 tiles [12], and Nurmi's

[1] Bounding occurs only if an objective fuction is employed.

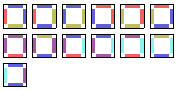
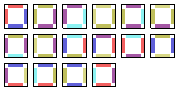
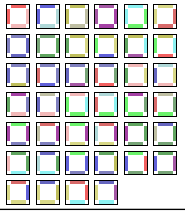
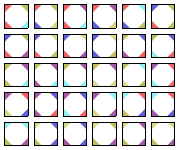
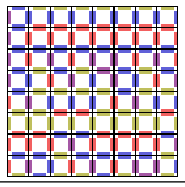
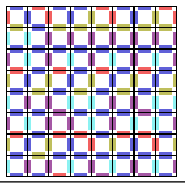
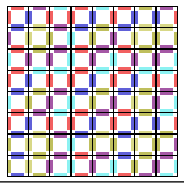
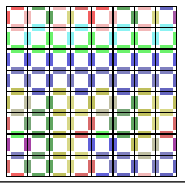
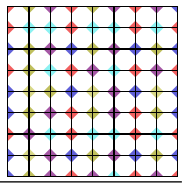
	11(4) [16]	**13(5) [15]**	**16(6) [12]**	**40(16) [8]**	**30(6) [19]**
Tile set					
Tiling sample					
LP 5×5	0.13 s/12 rel.	0.10 s/14 rel.	0.10 s/17 rel.	0.01 s/1 rel.	0.01 s/1 rel.
SDP 5×5	12.65 s/78 rel.	11.07 s/27 rel.	21.66 s/33 rel.	360.40 s/121 rel.	126.17 s/31 rel.
LP 6×6	0.10 s/12 rel.	0.12 s/14 rel.	0.13 s/17 rel.	0.02 s/1 rel.	0.03 s/1 rel.
SDP 6×6	69.60 s/34 rel.	227.24 s/79 rel.	184.23 s/33 rel.	1517.59 s/121 rel.	1522.72 s/91 rel.
LP 15×15	151.82 s/2190 rel.	over 6000 s	2.15 s/17 rel.	8.52 s/41 rel.	0.13 s/1 rel.
LP$_\mathbf{G}$ 15×15	0.06 s/0 rel.	0.09 s/0 rel.	0.39 s/0 rel.	0.79 s/0 rel.	1.49 s/0 rel.
LP 20×20	1545.94 s/26137 rel.	over 6000 s	8.52 s/17 rel.	32.09 s/41 rel.	4.23 s/31 rel.
LP$_\mathbf{G}$ 20×20	0.08 s/0 rel.	0.12 s/0 rel.	0.79 s/0 rel.	1.57 s/0 rel.	8.01 s/0 rel.
LP$_\mathbf{G}$ 25×25	396.5 s/7368 rel.	0.12 s/0 rel.	1.34 s/0 rel.	2.96 s/0 rel.	23.95 s/0 rel.
LP$_\mathbf{G}$ 30×30	390.18 s/3821 rel.	1361.51 s/5950 rel.	2.31 s/0 rel.	5.23 s/0 rel.	48.97 s/0 rel.

Table 1. Time demands and solved relaxations count in generation of aperiodic tilings. LP and SDP denote custom developed branch and bound method supplied with the linear and semidefinite relaxation, respectively; $\mathrm{LP_G}$ stands for Gurobi cut-and-branch method and the linear relaxation.

set of 30 corner tiles [19]. The tile sets were selected to sample different construction methods, tile set sizes, and numbers of colors used.

Results of the benchmarks, ran on the Intel® Core™ i5-4210H processor, are summarized in Table 1. They indicate that the linear relaxation surpassed the semidefinite one in terms of speed and in the number of explored nodes. The latter results from the property of the simplex algorithm, used for the solution of linear relaxations, to terminate in vertices of the feasible space polytope. These are more likely integer-feasible than an interior point found by the interior-point algorithm, used for the solution of semidefinite programs.

Comparison of our branch and bound algorithm and Gurobi's build-in branch and cut, both using the linear relaxation, is even clearer. Our implementation lacks heuristics, parallelism, and generation of cuts, consequently being inefficient for tile sets with large number of degrees of freedom, such as the Čulík one.

5. Conclusions

In this contribution, we demonstrated that the tiling generation problem of finite-sized areas can be formulated as an integer optimization problem; and we also introduced its linear and semidefinite programming relaxations. Both the relaxations were employed in the branch and bound method and benchmarked on five tile sets. The results indicate that the linear relaxation is more suitable for practical applications.

6. Nomenclature

List of symbols

$\mathcal{A}$ Rectangular area
$\mathcal{C}$ Color set
$\mathcal{H}$ Set of rows
$\mathcal{T}$ Tile set
$\mathcal{W}$ Set of columns
c Color iterator
e_k East edge color of the tile k
i Row iterator
j Column iterator
k, ℓ, m Tile iterators
n_c Number of colors in $\mathcal{C}$
n_k North edge color of the tile k
s_k South edge color of the tile k
$n_{\mathrm{t},h}$ Number of rows in $\mathcal{H}$
$n_{\mathrm{t},w}$ Number of columns in $\mathcal{W}$
$x_{i,j,k}$ Design variable, $x_{i,j,k} \in \{0, 1\}$
$y_{i,j,k}$ Design variable, $y_{i,j,k} \in \{-1, 1\}$
w_k West edge color of the tile k
I_inf Integer infeasibility

Acknowledgements

This work was supported partly by the Grant Agency of the CTU in Prague, SGS17/042/OHK1/1T/11, by the Grant Agency of the Czech Republic, GAČR 15-07299S, by the Ministry of Industry and Trade, MPO FV10202, and by the Technology Agency of the Czech Republic, TAČR TH02020420.

References

[1] H. Wang. Dominoes and the AEA case of the decision problem. *Symposium on the Mathematical Theory of Automata* pp. 23–55, 1962.

[2] H. Wang. Proving theorems by pattern recognition - II. *Bell System Technical Journal* **40**(1):1–41, 1961. DOI:10.1002/j.1538-7305.1961.tb03975.x.

[3] A. M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society* **s2-42**(1):230–265, 1937. DOI:10.1112/plms/s2-42.1.230.

[4] M. Davis. *Computability & Unsolvability*. Dover Books on Computer Science Series. Dover, 1958.

[5] A. S. Kahr, E. F. Moore, H. Wang. Entscheidungsproblem reduced to the ∀∃∀ case. *Proceedings of the National Academy of Sciences* **48**(3):365–377, 1962.

[6] R. Berger. *The undecidability of the domino problem*. 66. American Mathematical Society (AMS), 1966. DOI:10.1090/memo/0066.

[7] R. Berger. *The Undecidability of the Domino Problem*. Ph.D. thesis, Harvard University, 1964.

[8] H. Wang. Notes on a class of tiling problems. *Fundamenta Mathematicae* **82**(4):295–305, 1975.

[9] D. E. Knuth. *The art of computer programming. Vol. 1: Fundamental algorithms*. Addison-Wesley Publishing Co., Reading, Mass.-London-Don Mills, Ont, 1968.

[10] B. Poizat. Une théorie finiement axiomatisable et superstable. *Groupe d'étude des théories stables* **3**(1):1–9, 1980.

[11] R. M. Robinson. Undecidability and nonperiodicity for tilings of the plane. *Inventiones Mathematicae* **12**(3):177–209, 1971. DOI:10.1007/bf01418780.

[12] B. Grünbaum, G. C. Shephard. *Tilings and patterns*. Freeman, 1987.

[13] R. M. Robinson. Undecidable tiling problems in the hyperbolic plane. *Inventiones Mathematicae* **44**(3):259–264, 1978. DOI:10.1007/bf01403163.

[14] J. Kari. A small aperiodic set of Wang tiles. *Discrete Mathematics* **160**(1-3):259–264, 1996. DOI:10.1016/0012-365x(95)00120-l.

[15] K. Čulík. An aperiodic set of 13 Wang tiles. *Discrete Mathematics* **160**(1-3):245–251, 1996. DOI:10.1016/s0012-365x(96)00118-5.

[16] E. Jeandel, M. Rao. An aperiodic set of 11 Wang tiles, 2015. `arXiv:1506.06492v1`.

[17] A. Lagae, P. Dutré. An alternative for Wang tiles. *ACM Transactions on Graphics* **25**(4):1442–1459, 2006. DOI:10.1145/1183287.1183296.

[18] A. Lagae, J. Kari, P. Dutré. Aperiodic sets of square tiles with colored corners. *Report CW* **460**, 2006.

[19] T. Nurmi. From checkerboard to cloverfield: Using Wang tiles in seamless non-periodic patterns. *Bridges Finland Conference Proceedings* 2016.

[20] J. Kari. Reversibility of 2d cellular automata is undecidable. *Physica D: Nonlinear Phenomena* **45**(1-3):379–385, 1990. DOI:10.1016/0167-2789(90)90195-u.

[21] M. F. Cohen, J. Shade, S. Hiller, O. Deussen. Wang tiles for image and texture generation. *ACM Transactions on Graphics* **22**(3):287, 2003. DOI:10.1145/882262.882265.

[22] E. Winfree, F. Liu, L. A. Wenzler, N. C. Seeman. Design and self-assembly of two-dimensional DNA crystals. *Nature* **394**(6693):539–544, 1998. DOI:10.1038/28998.

[23] J. Novák, A. Kučerová, J. Zeman. Compressing random microstructures via stochastic Wang tilings. *Physical Review E* **86**(4), 2012. DOI:10.1103/physreve.86.040104.

[24] M. Doškář, J. Novák, J. Zeman. Aperiodic compression and reconstruction of real-world material systems based on Wang tiles. *Physical Review E* **90**(6), 2014. DOI:10.1103/physreve.90.062118.

[25] M. Doškář, J. Novák. A jigsaw puzzle framework for homogenization of high porosity foams. *Computers & Structures* **166**:33–41, 2016. DOI:10.1016/j.compstruc.2016.01.003.

[26] M. Tyburec. Modular-topology optimization of truss structures composed of Wang tiles, Master thesis, Czech Technical University in Prague, 2017. DOI:10.13140/rg.2.2.23612.85126.

[27] A. Lagae, P. Dutré. A procedural object distribution function. *ACM Transactions on Graphics* **24**(4):1442–1461, 2005. DOI:10.1145/1095878.1095888.

[28] D. Myers. Nonrecursive tilings of the plane. II. *Journal of Symbolic Logic* **39**, 1974. DOI:10.2307/2272641.

[29] E. Jeandel. On immortal configurations in Turing machines. In *Lecture Notes in Computer Science*, pp. 334–343. Springer Berlin Heidelberg, 2012. DOI:10.1007/978-3-642-30870-3_34.

[30] A. Lagae, P. Dutré. The tile packing problem. *Geombinatorics* **17**(1):8–18, 2007.

[31] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pp. 85–103. Springer US, 1972. DOI:10.1007/978-1-4684-2001-2_9.

[32] S. Boyd, L. Vandenberghe. Semidefinite programming relaxations of non-convex problems in control and combinatorial optimization. In *Communications, Computation, Control, and Signal Processing*, pp. 279–287. Springer US, 1997. DOI:10.1007/978-1-4615-6281-8_15.

[33] G. Nemhauser, L. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, Inc., 1988. DOI:10.1002/9781118627372.

[34] Gurobi Optimization, Inc. Gurobi optimizer reference manual, 2017. `http://www.gurobi.com`.

[35] MOSEK ApS. MOSEK Fusion API for C++ 8.1.0.23, 2017. `http://docs.mosek.com/8.1/cxxfusion`.