Documentation for `RandomTilings`

Max van Horssen & Lennart Hübner

Department of Mathematics, KU Leuven,
Celestijnenlaan 200 B bus 2400, 3001 Leuven, Belgium
max.vanhorssen@kuleuven.be, lennart.huebner@kuleuven.be

September 23, 2025

Abstract

This document provides the documentation for the Python program `RandomTilings`, which is a translation into Python of the Matlab program `MatlabTilings` by Christophe Charlier. The underlying algorithm is the shuffling algorithm as described in arXiv:0111034. This document is a reduced adaptation of the *Help file* for `MatlabTilings`. We are grateful to Christophe Charlier for allowing us to make this program publicly available.

Contents

0 Specifics of the Python program

Let us begin with an important note. The Python program uses the `Numba` package to compile certain routines, resulting in a drastic performance improvement. This package needs to be installed to be able to run the program. The disadvantage of this package is that the type-handling is more delicate than is usually the case in Python; please be aware of this fact. In addition, the packages `NumPy` and `Matplotlib` are also required.

In the current version, not all options from `MatlabTilings` have been implemented in the Python program. Most notably, the program only produces images of tilings, and not of the corresponding non-intersecting path systems. This document mainly discusses the necessary changes important for the Python program. Some parts of the *Help file* for `MatlabTilings` are copied for context. For a more complete description, the reader should consult the *Help file*; please keep in mind that only some features are available in the Python program.

1 Lozenge tilings

1.1 `draw_hexagon`

The function `draw_hexagon` generates a random tiling of a hexagon for a given *periodic weighting*. The syntax is given by: `draw_hexagon(n,w,a,b,c,edge,dpi,show_figure)`. By default, `a=1,b=1,c=1,edge=0,dpi=200,show_figure=True`.

1. `draw_hexagon(n,w)` is the same as `draw_hexagon(n,w,1,1,1,0,200,True)`.
2. The sides of the hexagon are of length `a*n`, `b*n`, and `c*n`.
3. `w` is the weighting on the edges. For a $p \times q$ periodic weightings, `w` must be a matrix of size $2p \times q$. We adopt the convention that the weightings is assigned on the bottom left corner of the hexagon as shown in Figure 1, and is then extended by periodicity over the full hexagon.

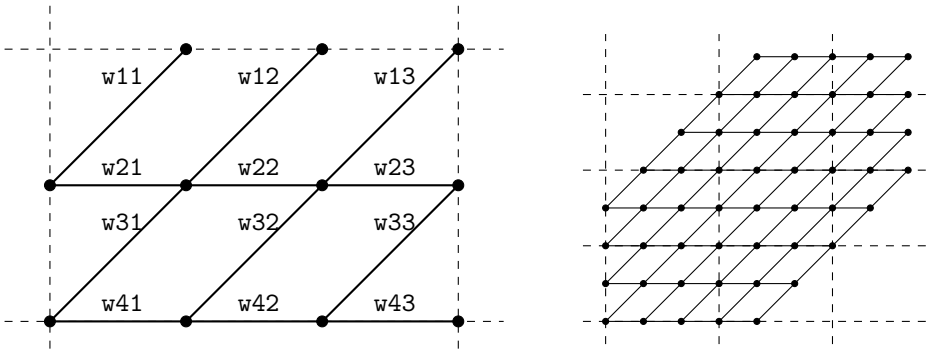
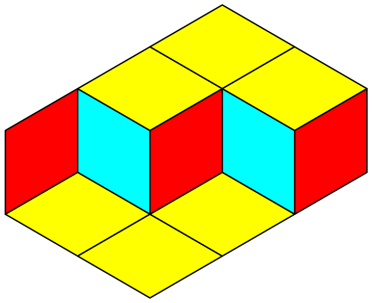


Figure 1: A 2×3 periodic weighting on a $4 \times 4 \times 4$ hexagon. The dashed lines emphasize the periodicity. Here `w=np.array([[w11,w12,w13],[w21,w22,w23],[w31,w32,w33],[w41,w42,w43]])`.

4. `edge` is the thickness of the lines around the lozenges. Use `edge=0` for no edges.
5. `dpi` is the resolution of the image.
6. `show_figure` tells the program to show the figure when `show_figure = True`.
7. The function returns the figure `fig`, which allows the user to save it using, for example, `fig.savefig('figure_name.png')`.

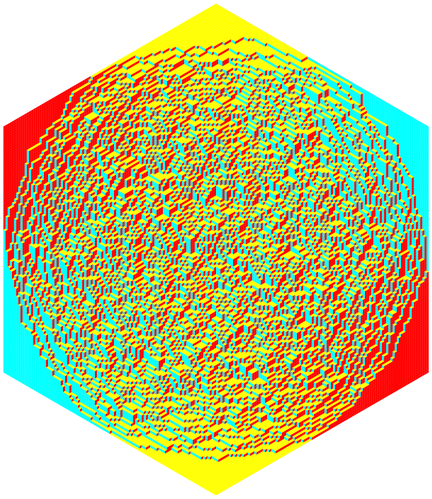
1.1.1 About a, b, and c

With `w=np.array([[1,1]]).T`, `draw_hexagon(1,w,1,2,3,edge=1)` produces:



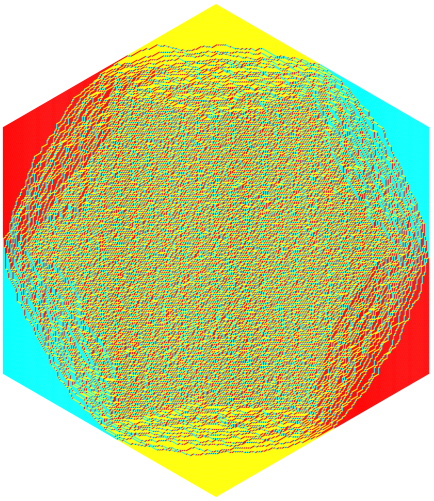
1.1.2 The uniform weighting

With `w=np.array([[1,1]]).T`, `draw_hexagon(100,w,dpi=500)` takes around 10-12 seconds on our computer to produce:



1.1.3 A 3×3 periodic weighting

With `alpha1=0.3, alpha2=0.3, w=np.array([[1,1,1],[1,1,1],[1,1,1],[1/alpha1,alpha2, alpha1/alpha2],[1,1,1],[alpha1,1/alpha2,alpha2/alpha1]])`, `draw_hexagon(200,w,dpi=500)` takes around 2 minutes on our computer to produce the example from arXiv:2412.03115:



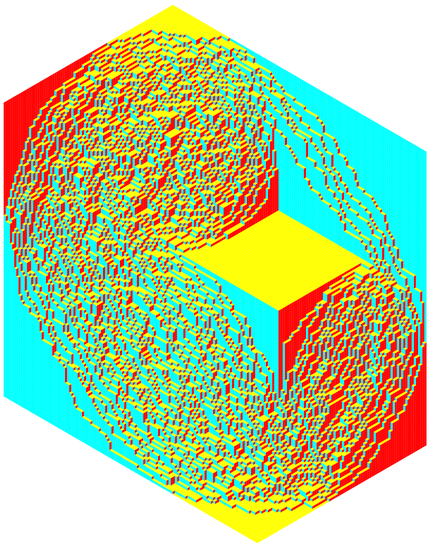
1.2 draw_hexagon_gap

The function `draw_hexagon_gap` generates a random tiling of a hexagon for a given *periodic weighting* with some specified *gaps*. The syntax is given by: `draw_hexagon_gap(n,w,gap,a,b,c,edge,dpi,show_figure)`. By default, `a=1,b=1,c=1,edge=0,dpi=200,show_figure=True`.

1. `gap` must be of the form `np.array([[t1,x1,y1],[t2,x2,y2],...,[tm,xm,ym]])`, which means that at each time `tj`, there is a vertical gap from `xj` to `yj`. The points must satisfy `tj` $\in \{0, 1, \ldots, 2n\}$ and `xj`,`yj` $\in \mathbb{Z}$. The program also allows for `xj=yj`, which represents a single point `xj`.
2. The function return a vector `vec=[logPn,logZnNum,logZnDen]` and a figure `fig`, i.e., `vec, fig = draw_hexagon_gap(n,w,gap)`.
3. `logZnNum` is the log of the partition function of the hexagon with the specified gaps.
4. `logZnDen` is the log of the partition function of the hexagon without any gaps.
5. `logPn=logZnNum-logZnDen` is the log of the probability to observe a tiling with the specified gaps.

1.2.1 A hexagon tiling with a gap

With `w=np.array([[1,1]]).T,gap=np.array([[130,100.5,140.5]])`, `draw_hexagon_gap(1,w,gap, 120,120,80,dpi=500)` produces:



2 Aztec diamond

2.1 draw_aztec

The function `draw_aztec` generates a random tiling of an Aztec diamond for a given *periodic weighting*. The syntax is given by: `draw_aztec(n,w,edge,dpi,rotated,show_figure)`. By default, `edge=0,dpi=200, rotated=True,show_figure=True`.

1. `draw_aztec(n,w)` is the same as `draw_aztec(n,w,0,200,True,True)`.
2. `w` is the weighting on the edges, which can be a matrix of any size. We adopt the convention that the weightings is assigned on the bottom left corner of the Aztec diamond as shown in Figure 1, and is then extended by periodicity over the full Aztec diamond.

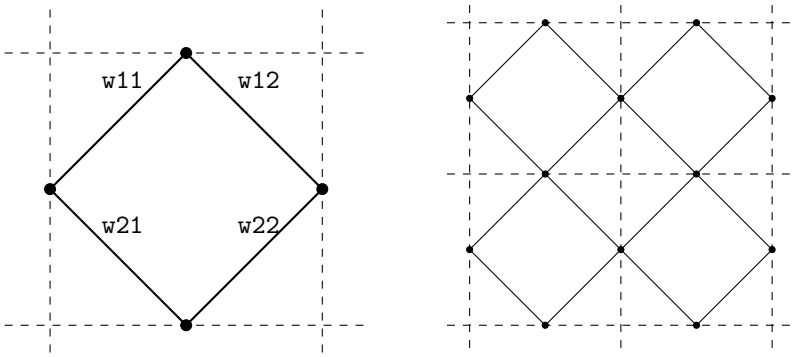
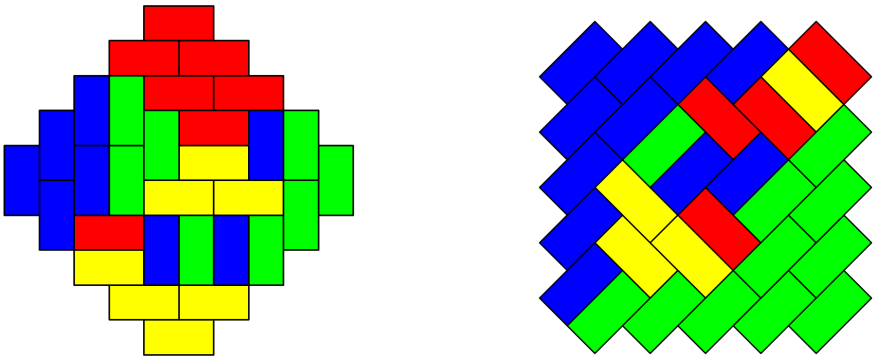


Figure 2: Right: an Aztec diamond of size 2. The dashed lines emphasize the periodicity. Here `w=np.array([[w11,w12],[w21,w22]])`.

3. `edge` is the thickness of the lines around the lozenges. Use `edge=0` for no edges.
4. `dpi` is the resolution of the image.
5. `rotated` tells the program to rotate the figure by 45 degrees when `rotated = True`.
6. `show_figure` tells the program to show the figure when `show_figure = True`.
7. The function returns the figure `fig`, which allows the user to save it using, for example, `fig.savefig('figure_name.png')`.

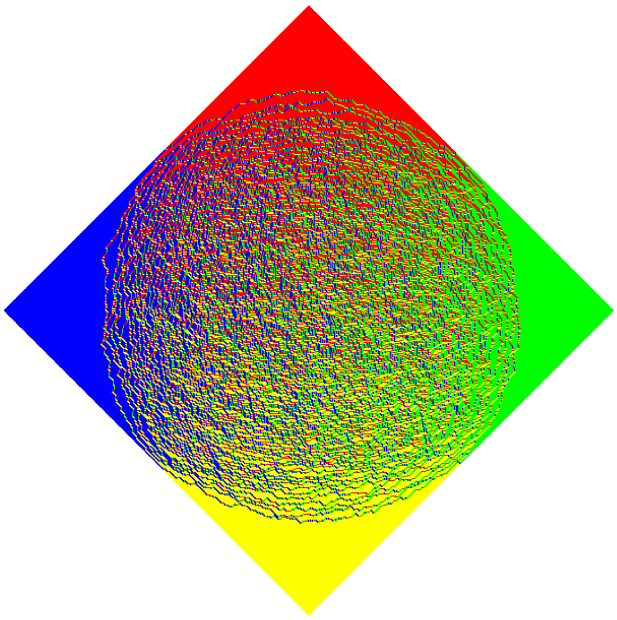
2.1.1 About `rotated`

With `w=np.array([[1]]).T`, `draw_aztec(5,w,edge=1,rotated=True)` (right) and `draw_aztec(5,w,edge=1,rotated=False)` (left) produce:



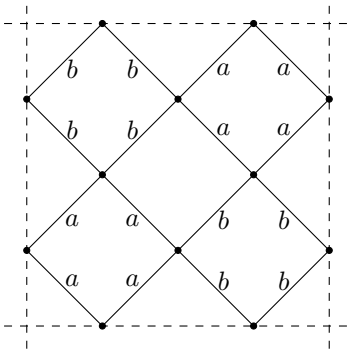
2.1.2 The uniform weighting

With `w=np.array([[1]]).T`, `draw_aztec(300,w,dpi=500)` takes around 15–20 seconds on our computer to produce:

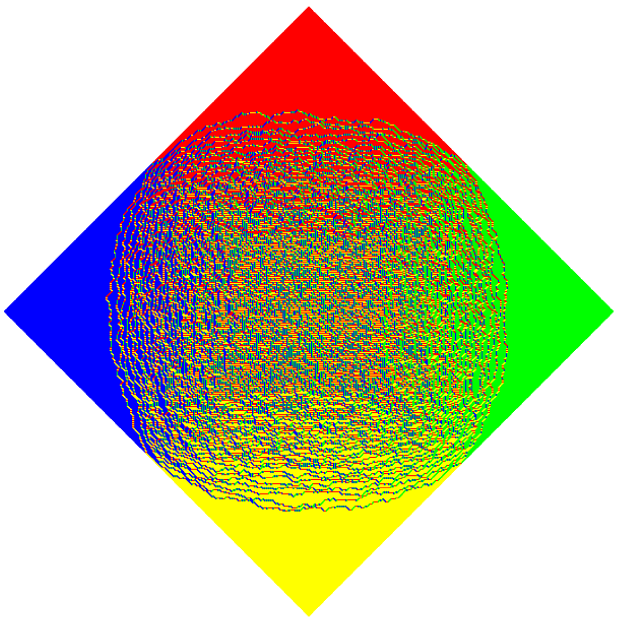


2.1.3 A 2×2 periodic weighting

The 2×2 periodic weightings from arXiv:1410.2385 is given by



With `a=0.5, b=1, w=np.array([[b,b,a,a],[b,b,a,a],[a,a,b,b],[a,a,b,b]])`, `draw_aztec(300,w,dpi=500)`:



2.2 draw_aztec_gap

The function `draw_aztec_gap` generates a random tiling of an Aztec diamond for a given *periodic weighting* with some specified *gaps*. The syntax is given by: `draw_aztec_gap(n,w,gap,edge,dpi,rotated,show_figure)`. By default, `edge=0,dpi=200,rotated=True,show_figure=True`.

1. `gap` must be of the form `np.array([[t1,x1,y1],[t2,x2,y2],...,[tm,xm,ym]])`, which means that at each time `tj`, there is a vertical gap from `xj` to `yj`. The points must satisfy `tj` $\in \{0, 1, \ldots, 2n\}$ and `xj`,`yj` $\in \mathbb{Z}$. The program also allows for `xj=yj`, which represents a single point `xj`.
2. The function return a vector `vec=[logPn,logZnNum,logZnDen]` and a figure `fig`, i.e., `vec, fig = draw_hexagon_gap(n,w,gap)`.
3. `logZnNum` is the log of the partition function of the hexagon with the specified gaps.
4. `logZnDen` is the log of the partition function of the hexagon without any gaps.
5. `logPn=logZnNum-logZnDen` is the log of the probability to observe a tiling with the specified gaps.

2.2.1 An Aztec diamond tiling with a gap

With `w=np.array([[1]]).T,gap=np.array([[2*150-1,80,150]])`, `draw_aztec_gap(300,w,gap,dpi=500)` produces: