Tilings of the Aztec Diamonds

by

Zhiyuan Wei, Harsha Kenchareddy, Braeden Bertz, Ying Zheng

A paper submitted in fulfillment
of the requirements for the fall
2022 MXM project

at the
University of Wisconsin-Madison

Date of Submission: 12-19-2022

Large Rank Aztec Diamonds

Zhiyuan Wei, Harsha Kenchareddy, Braeden Bertz, Ying Zheng

Abstract

The Aztec Diamond is constructed by the union of unit squares in the plane whose edges lie on the lines of a square grid and whose centers (x, y) satisfy

$$|x-\frac{1}{2}|+|y-\frac{1}{2}|https://www.overleaf.com/project/63a221780b01796ff618e2b2 \leq n \tag{1}$$

https://www.overleaf.com/project/63a221780b01796ff618e2b2

We can associate a tiling of 2×1 non-overlapping rectangles and an associated height function to each tiling. At a large rank n, we see two distinct regions: A frozen region where dominoes of the same orientation group together, and a disordered region where the tilings follow no such order. These regions are visually separated by a circular boundary. Regarding the height function of large rank AD, several theoretical results about the fluctuations of the average height at a point and the covariance of the height exists in the literature. We implemented the domino shuffling algorithm to generate uniformly at random a rank n AD and then numerically test the theoretical results. We verified that the fluctuations of the height at a point (x, y) in the disordered region of the AD was normally distributed as the rank of the AD approaches infinity.

Contents

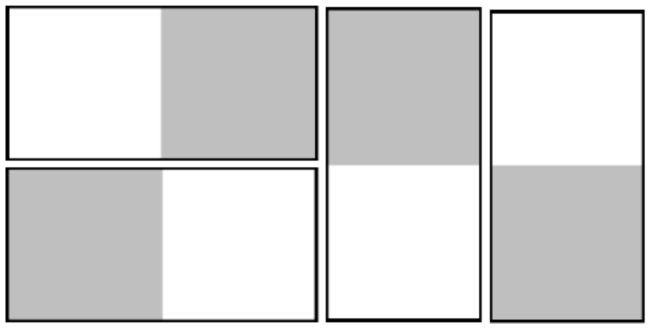

Figure 1: The four possible dominoes of an Aztec Diamond

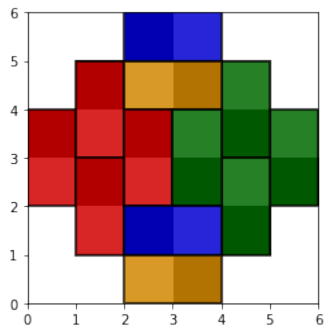

Figure 2: A domino tiling of rank 3 Aztec Diamond where each unique domino orientation has a different color associated with it

0.1 Construction of a Rank N Diamond

An Aztec diamond of order n is the region obtained from four staircase shapes of height n by gluing them together along the straight edges. It can therefore be defined as the union of unit squares in the plane whose edges lie on the lines of a square grid and whose centers (x, y) satisfy

$$|x - \frac{1}{2}| + |y - \frac{1}{2}| \leq n \tag{2}$$

If we consider embedding the squares in a checkerboard region, then we denote a domino as the union of any two unit squares that share an edge.

The recognition of the existence of four rather than two distinct sorts of tiles plays a key role in the formulation of the "shuffling algorithm". Shuffling was introduced in [1] to prove that the Aztec diamond of order n has exactly $2^{n(n+1)/2}$ domino tilings. here we use shuffling to generate tilings uniformly at random (Figure (2) was generated in precisely this way). This shuffling algorithm generates uniformly at random an Aztec Diamond of rank n, i.e., it ouputs any of the $2^{n(n+1)/2}$ rank n Aztec Diamonds with probability $2^{-(n(n+1)/2)}$. To generate our own rank n Aztec Diamonds, we could choose from a number of algorithms, including the Metropolis Hasting's algorithm and the Domino Shuffling algorithm.

At a very high level, the Metropolis Hasting's algorithm uses rejection sampling to sample from a distribution to generate the AD. However, the algorithm has several limitations when compared to Domino Shuffling. It is hard to know when we should stop to ensure a good sampling, and its slow in general. The Domino Shuffling algorithm is much faster and simpler. Domino shuffling is a stochastic procedure that turns a domino tiling of the Aztec diamond of rank $n-1$ into one of several domino tilings of the Aztec diamond of rank n. If one starts from the (empty) tiling of the Aztec diamond of rank 0 and applies shuffling n times, the result is a uniformly random domino tiling of the Aztec diamond of rank n. This is the algorithm that we implemented.

1. Start with rank 0 Aztec Diamond and uniformly choose a rank 1 tiling. There are only 2 possible options.

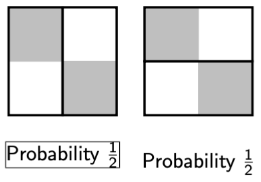

Figure 3: Rank 1 Aztec Diamond

2. With a tiling of rank-n, embed the tiling into an AD of rank $(n+1)$. Make sure to alternate the coloring of the squares.

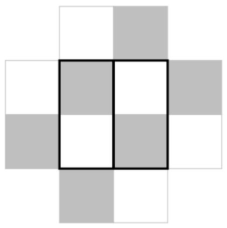

Figure 4: Rank 1 tiling embedded into rank 2 AD

3. Once we have the new AD of rank $(n+1)$ we slide the dominoes based on their orientations. There are 2 possible outcomes. (a) The dominoes slide away from each other leaving an empty 2×2 space between them. (b) The dominoes slide into each other or collide are both deleted leaving a empty 2×2 space. We call this destruction. From [1] we know that no dominoes overlap after destruction and sliding have taken place.

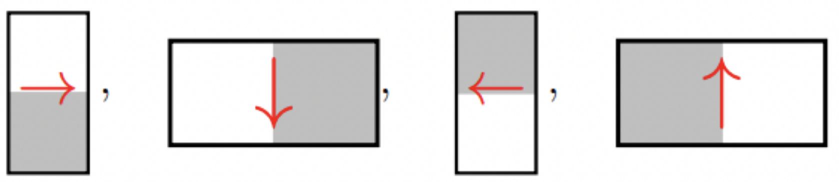

Figure 5: How to slide a domino given its position on the Aztec Diamond

4. Once we slide and destroy every domino we are left with multiple empty 2×2 spaces that haven't been tiled yet. We now fill these empty spaces by uniformly choosing between the two possible rank 1 matrices. This step is called creation
5. We finally swap the square colors for the dominoes to get our uniformly random tiling.
6. We repeat steps $2 - 5$ until we get our desired uniformly random tiling of rank-N.

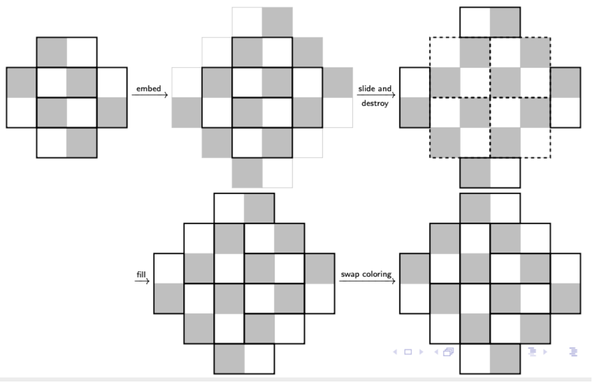

Figure 6: Steps $2 - 5$ for a rank-2 to a rank-3 AD

Unlike the Metropolis-Hastings algorithm the Domino Shuffling algorithm is less general and much more specific to our model. This makes it more efficient and faster. We implement this code in python in section (A).

0.2 The Height Function

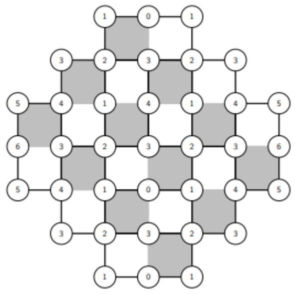

Figure 7: A height function for a rank 3 Aztec Diamond

There is a bijection between the tilings of the Aztec Diamond and the height function H : {vertices of Aztec Diamond} $\to \mathbb{Z}$. The bijection is described in Figure (7). Notice that the height function satisfies:

- H_T takes successive values when moving along the boundary counterclockwise $(0, 1, ..., 2n+1, 2n+2, 2n+1, ...0, ...)$
- if two vertices v_1, v_2 are adjacent, then $H_T(v_2) = H_T(v_1) + 1 v H_T(v_1) - 3$ depending on if they are connected in the tiling or not.

Given the random tiling of a large rank Aztec Diamond, the height function approaches to a deterministic surface, as shown in the Figure (8).

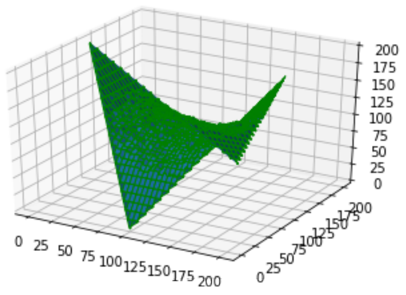

Figure 8: Height function of a rank 100 Aztec Diamond

In the limit of rank n, we have that the height function at a point (x, y) of the

Aztec Diamond in the disordered region is:

$$h^*(x,y) = \frac{2}{\pi}((\arg(z) - \arg(w))x + (\arg(z) + \arg(w))y + (\arg(z-1) - \arg(z) + \arg(z-1)) \tag{3}$$

$$z(x,y) = \frac{x - y + i\sqrt{1 - 2(x^2 + y^2}}{1 + x + y} \tag{4}$$

$$w(x,y) = -\frac{1 + z(x,y)}{1 - z(x,y)} \tag{5}$$

Further, the fluctuations of the observed rank n height function around its average should satisfy:

$$\lim_{N\to\infty} \frac{h_N(N_x, N_y) - \mathbb{E}[h_N(N_x, N_y)]}{(\frac{\log N}{2\pi^2})^{1/2}} \sim \mathcal{N}(0,1) \tag{6}$$

0.3 Numerical Results

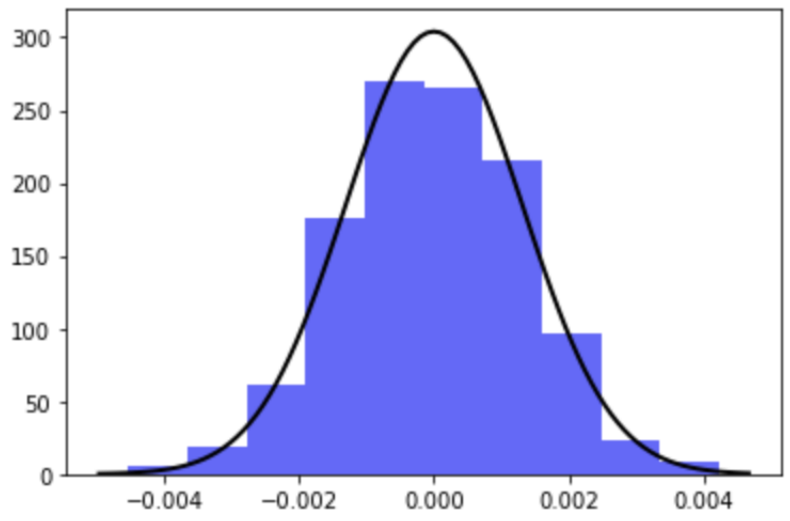

Figure 9: Histogram of height functions of a rank 100 Aztec Diamond

We numerically tested the fluctuations of the average height at a point described in equation (6). We generated 2000 random-tilings of rank 100 and cop muted their average around a 3×3 box of points with the height function. We chose several points close to each other instead of just one point to give us better results since we are limited by computing power to generate more tilings of higher ranks. Our results shown in figure (9) appear to be normally distributed with slight discrepancies that can be attributed to not a large enough rank and using a box of points instead of a single point. As a result, we can expect that as the rank and number of samples increase, the fluctuations in height will get closer to a normal distribution.

Appendix A

Implementation in Python

We can generate any AD of rank-N with the shuffling algorithm below by simply calling domino_shuffle(n)

```
"""
The Aztec Diamond is represented as a dictionary
where the key is the coordinate location of the
bottom left point of each tile and the value is the
orientation/sliding direction of the tile
u:up,d:down,l:left,r:right,None:empty tile
The coordinate plane I use to map the tiles
has the bottom left point of the middle block centered
at 0,0

Lets take the rank 1 tiling for example.
This tiling has 2 possible reprsentations
The dictionary representation therefore can either be
{-1,-1:d, 0,-1:d, -1,0:u, 0,0:u} or
{-1,-1:l, 0,-1:r, -1,0:l, 0,0:r}
"""

import random

def domino_shuffle(n):
    tiling = {}
    tiles = []

   #iterate through the ranks
   and generate boards until we get
   desired rank
   for rank in range(1,n+1):
       tiling,tiles = create_board(rank,tiling, tiles)
    return tiling

"""
Each time I call create_board, I create a new coordinate
system for the new rank
I then create an empty dictionary and populate its keys with
the coordinates I just generated
I then call destroy,slide,fill to generate the orientations
for the new board
"""
def create_board(rank,prev_tiling,prev_tiles):
    tiles = []
    #generate coordinate system
    for i in range(-rank, rank):
        m = min(rank + 1 + i, rank - i)
        for j in range(-m, m):
            tiles.append((j, i))

    tiling = {tile: None for tile in tiles}

    if(rank != 1):
        prev_tiling = destroy(prev_tiles,prev_tiling)
        tiling = slide(prev_tiles,tiling,prev_tiling)

    return fill(tiles,tiling),tiles

"""
Based on the tiles orientation from the previous board, I
slide the tiles and update
accordingly in the new board
"""
def slide(prev_tiles,tiling,prev_tiling):
    for (i,j) in prev_tiles:
        if prev_tiling[(i, j)] == "u":
            tiling[(i, j + 1)] = "u"
        if prev_tiling[(i, j)] == "d":
            tiling[(i, j - 1)] = "d"
        if prev_tiling[(i, j)] == "l":
            tiling[(i - 1, j)] = "l"
        if prev_tiling[(i, j)] == "r":
            tiling[(i + 1, j)] = "r"

    return tiling
```

```
"""
I check if any tiles in the previous tiling are oriented
towards each other, in other words
they will slide into each other during the slide step. If so
I delete all these tiles by setting
their values to None
"""
def destroy(tiles, tiling):
    for i, j in tiles:
        try:
            if check_block(i, j, tiling,["u", "u", "d", "d"])
            or check_block(
                    i, j, tiling,["r", "l", "r", "l"]):
                    fill_helper(i, j, tiling,[None] * 4)
        except KeyError:
                pass
    return tiling


"""
Helper method to check if block or 4 tiles have a certain set
of values
"""
def check_block(i,j,tiling,domino_values):
    if (tiling[(i,j)] == domino_values[0] and tiling[(i+1,j)]
    == domino_values[1] and
        tiling[(i,j+1)] == domino_values[2] and
        tiling[(i+1,j+1)] == domino_values[3]):
        return True
    return False

"""
I fill the remaining empty tiles by checking for groups of
empty 4 tiles or blocks
If I find an empty block I randomly pick a rank 1 tiling to
fill into the block
"""
def fill(tiles,tiling):
    for i, j in tiles:
            try:
                rint = random.randrange(2)
                if check_block(i, j, tiling, [None] * 4):
                    if rint == 0:
                        tiling = fill_helper(i, j, tiling,
                        ["d", "d", "u", "u"])
                    else:
                        tiling = fill_helper(i, j, tiling,
                        ["l", "r", "l", "r"])
            except KeyError:
                pass
    return tiling

"""
Helper method for fill
"""
def fill_helper(i,j,tiling,domino_values):
    tiling[(i,j)] = domino_values[0]
    tiling[(i+1,j)] = domino_values[1]
    tiling[(i,j+1)] = domino_values[2]
    tiling[(i+1,j+1)] = domino_values[3]

    return tiling
```

Bibliography

[1] Noam Elkies et al. "Alternating-Sign Matrices and Domino Tilings (Part I)". en. In: *Journal of Algebraic Combinatorics* 1.2 (Sept. 1992), pp. 111–132. ISSN: 1572-9192. DOI: 10.1023/A:1022420103267. URL: https://doi.org/10.1023/A:1022420103267.