Distributed Density Estimation

Amartya Shankha Biswas *

0.1 Domino Tiling

We will consider the problem of tiling a square grid with dominos. This problem has a long histor and various importtant applications in statistical physics. Specifically, we will focus on the local generation of domino tilings from the uniform distribution.

0.2 $2 \times n$ Domino Tiling

The simplest version of the problem is one where we are given a $2 \times n$ grid (Figure **??**. The queries will be as an index, and the generator should report the orientation of the domino at the i^{th} position in the grid.

It is a well known result that the number of tilings of a $2 \times n$ grid is exactly F_n. cite

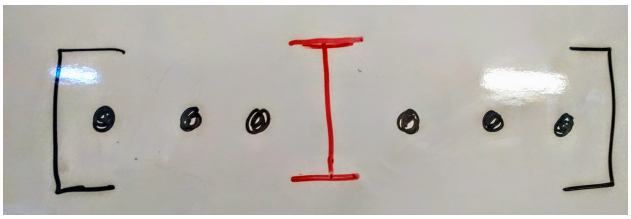
To aid with generalization, we will instead allow the generator to respond with the splitting boundary of the current tiling instead. For example, in Figure 1a, the boundary is a vertical line at the specified position. In Figure 1b, the boundary is horizontal, indicating that there are two horizontal dominos at that location. Note that Figure 1c is impossible for a $2 \times n$ grid. It should be clear that this query model is equivalent.

Now, consider a query to the location i, such that all positions between $i - a$ and $i + b$ have not been queriesd so far. So, there is a blank $2 \times (a + b)$ size sub-grid that we have to sample from. Let us consider the number of possible tilings resulting from each possible splitting boundary.

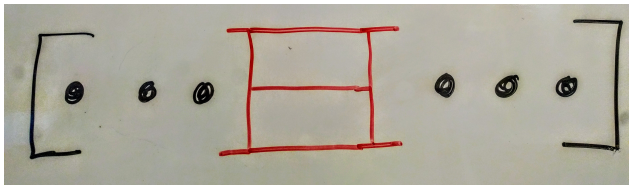
1. Vertical Boundary – This indicates that we divide the region into two sub-grids with sizes $2 \times a$ and $2 \times b$. So, the total number of possible tilings is exactly $F_a \cdot F_b$.
2. Horizontal Boundary – This indicates that we divide the region into two sub-grids with sizes $2 \times (a - 1)$ and $2 \times (b - 1)$. So, the total number of possible tilings is exactly $F_{a-1} \cdot F_{b-1}$.

So the probabilities are computed as $\frac{F_a \cdot F_b}{F_a \cdot F_b + F_{a-1} \cdot F_{b-1}}$ and $\frac{F_{a-1} \cdot F_{b-1}}{F_a \cdot F_b + F_{a-1} \cdot F_{b-1}}$. Now, we face the issue of approximating these fractions. If either of the values a or b are less than $\Theta(\sqrt{n})$, then we

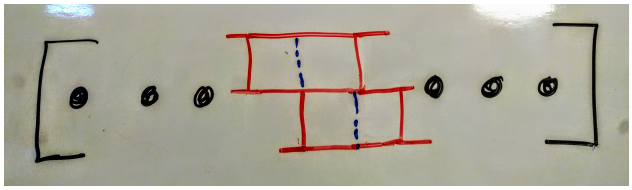
*MIT, Cambridge MA 02139. E-mail: asbiswas@mit.edu.



(a)



(b)



(c)

Figure 1: Caption for this figure with two images

can compute the exact value of the corresponding F_a or F_b. Otherwise, we use Lemma **??** to approximate $F_a = \phi \cdot F_{a-1}$ and $F_b = \phi \cdot F_{b-1}$. So, the probability of the vertical boundary becomes

$$\frac{F_a \cdot F_b}{F_a \cdot F_b + F_{a-1} \cdot F_{b-1}} = \frac{\phi^2}{\phi^2 + 1}$$

Similarly, the probability of a horizontal split with a top and bottom domino becomes $1/(\phi^2 + 1)$. Note that this also determines the two adjacent boundaries.

The only information we needed to make this query was the extent of the un-queried interval $[i - a, i + b]$. We can use any standard data-structure that allows insertion in positions $\{1, 2 \cdots, n + 1\}$, and provides successor and predecessor queries.

Here's we can be fancy and use Van-Emde-Boas trees to get a $\mathcal{O}(\log \log n)$ query time. However, in some cases, the exact value of a Fibonacci number still needs to be computed, and this takes $\mathcal{O}(\log n)$ time. The faster queries only work when the new query is "far enough" ($\mathcal{O}(\log n)$ distance) away from all previous queries.

0.3 Permutations

Anak: Did we try to support more information that just computing $\pi(i)$?

[Specification]

In addition to a table (dictionary) containing all assigned values $\pi(i)$, we maintain the following binary trees, whose nodes are generated on-the-fly in response to queries.

T_1: Each node of T_1 corresponds to a specific range of indices, where the root represents the entire range $\{1, \ldots, N\}$, and its two children represents each (approximately) half of the parent's range.

Each node counts the indices i in the range, such that $\pi(i) = \perp$. Initially T_1 only contains the root node, and the number of unassigned indices are N. The children are only generated when we need to traverse down from the root; as these nodes are generated, all of the indices in their ranges are unassigned. Once an index becomes assigned, we simply update the information along the path in $\mathcal{O}(\log n)$ time.

T_2: T_2 is similar to T_1 but instead of maintaining the number of indices in the range that are still unassigned, it maintains the number of values in the range that are still unused (have not been assigned to an index). Similarly, we may sample an unused value or mark it as used within $\mathcal{O}(\log n)$ time.

To compute $\pi(i)$, first we check the table for $\pi(i)$ and return its value if $\pi(i) \neq \perp$. Otherwise, sample an unused value j from T_2 and mark that value as used. Add $\pi(i) = j$ to the table, and mark index i as used on T_1.

If we wish to support $\pi^{-1}(i)$, then also store the table of $\pi^{-1}(i)$. To assign $\pi^{-1}(i)$, sample an unassigned index j from T_1 then mark it as assigned, add $\pi(j) = i$ and $\pi^{-1}(i) = j$ to the table, and mark the value i as used on T_2.

0.4 Generating Permutations with given Cyclic Structure

We will use the technique from [NR02] to locally generate a random permutation with a given cyclic structure. This algorithm uses two permutations π and σ, where π is a uniformly random permutation and σ is a fixed permutation with the given structure. The resulting random permutation is formed by the composition $\pi^{-1}(\sigma(\pi(\cdot)))$. We will generate the permutation π as described in the previous section.;

We receive as input a list of cycle sizes $\{c_1, c_2, \cdots, c_k\}$ with the restriction that $\sum c_i = n$. Now we need to locally generate the permutation σ with the prescribed structure. Define the indices $C_j = \sum_{i=1}^{j} c_i$ with $C_0 = 0$. We will construct the cycle corresponding to c_i as all the elements in the interval $\{c_{i-1} + 1, c_{i-1} + 2, \cdots, c_i\}$.

Of course we will not be computing σ explicitly. Instead, we will pre-compute the C_j indices, and when given a query $\sigma(x)$, we binary search amongst C_j to find the cycle that x belongs to. Then we can report the value of $\sigma(x)$ accordingly.

So, we can now compose the generator oracles for π, σ, and π^{-1} to get the full generator.

1 Catalan Objects and Dyck Paths

Dyck paths are one interpretation of the Catalan numbers. Here, we will instead consider a more general form of Dyck Paths, which correspond to numbers in the *Catalan Trapezoid*.

A Dyck path can be constructed as a $2n$ step random walk on the $\mathbb{Z}^2$ lattice, starting at the origin $(0, 0)$ and ending at (n, n) (Figure 2). Each step in the walk moves one unit either along the positive x-axis or the positive y-axis. Given these restrictions, we would obtain a 1D random walk pinned to zero on both sides. A Dyck path also has the additional restriction that for any position (x, y) on the path, $y - x > 0$ i.e. the walk is always on one side of the diagonal.

The number of possible Dyck paths is the n^{th} Catalan number –

$$C_n = \frac{1}{n+1} \cdot \binom{2n}{n}$$

We will attempt to support queries to a uniformly random instance of a Dyck path. Specifically, we will want to query the position of the i^{th} index.

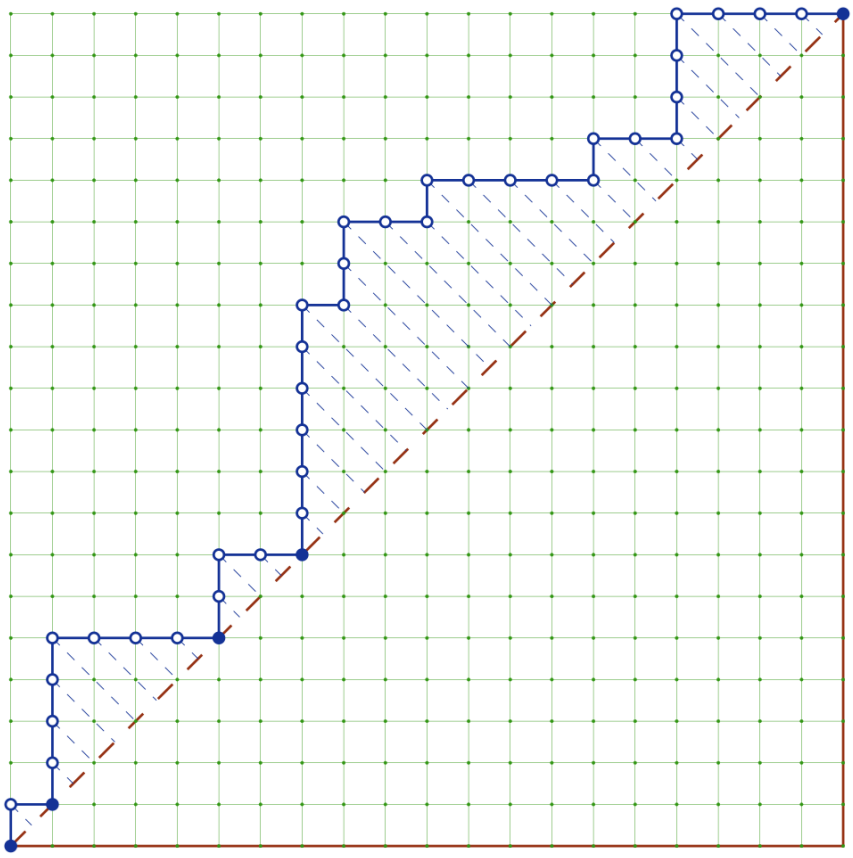


Figure 2: Simple Dyck path from $(0, 0)$ to $(20, 20)$

1.1 Catalan Trapezoids and Generalized Dyck Paths

First, we define Catalan trapezoids as presented in [Reu14]. Let $C_k(n, m)$ be the $(n, m)^{th}$ entry of the Catalan trapezoid of order k, where $C_1(n, m)$ corresponds to the Catalan triangle.

The interpretation is as follows: Consider a random walk from $(0,0)$ to (m,n) (n steps along the y-axis, and m steps along the x-axis), such that for any position (x,y) along the walk, $y-x > 1-k$ (Figure 3) i.e. the walk is always on one side of the shifted diagonal. The total number of such paths is exactly $C_k(n,m)$. For $k=1$, we obtain the definition of the simple Dyck path (Figure 2).

Now, we state a result from [Reu14] without proof

$$C_k(n,m) = \begin{cases} \binom{n+m}{m} & 0 \le m < k \\ \binom{n+m}{m} - \binom{n+m}{m-k} & k \le m \le n+k-1 \\ 0 & m > n+k-1 \end{cases}$$

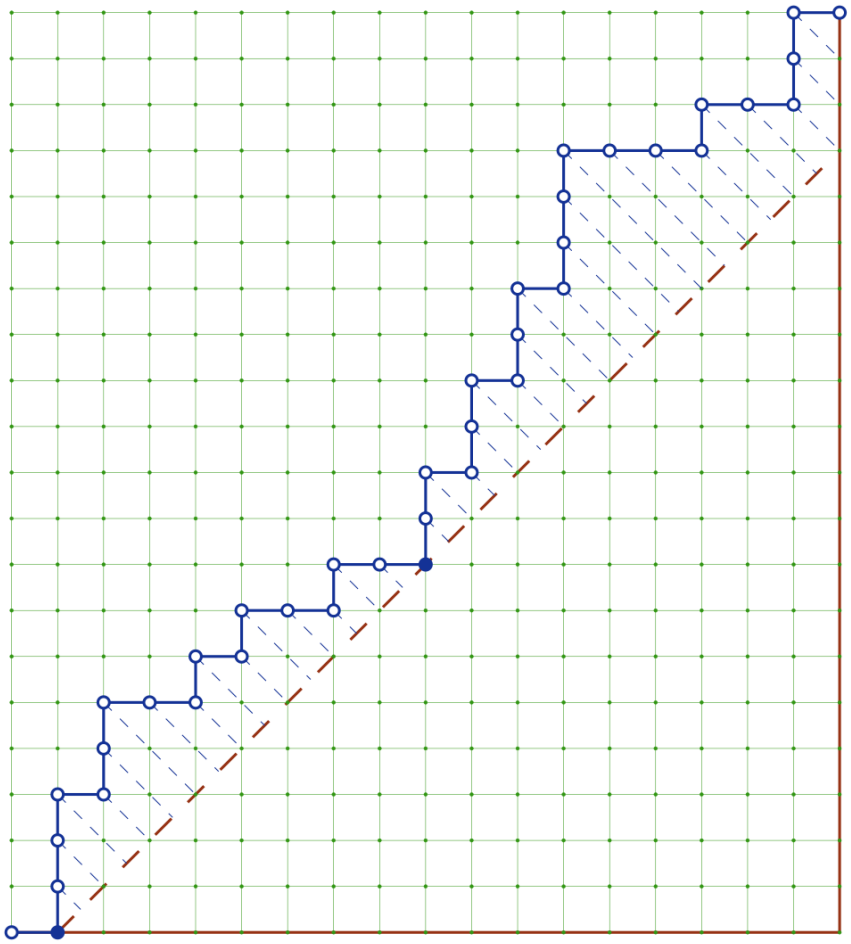


Figure 3: Complex Dyck path from $(0,0)$ to $(18,20)$ with $k=2$. Notice that the diagonal is shifted.

1.2 Generating Dyck Paths

Our general recursive step is as follows. We consider a sequence of length $2S$ comprising of $2U$ up moves $(+1)$ and $2D$ down moves (-1). Additionally, the sum of any initial sequence prefix? canon

be less than $k-1$. Without loss of generality, let's assume that $2D \leq S$. If this were not the case, we could simply flip the sequence and negate the elements. This essentially means that the overall Dyck path is non-decreasing.

Lemma 1. $S - 2D = \mathcal{O}(\log n\sqrt{S}) \implies U - D = \mathcal{O}(\log n\sqrt{S})$

We want to sample the height of this path after S steps. This is the same as sampling the number of $(+1)$s that get assigned to the first half of the elements in the sequence. We define p_d as the probability that exactly $D-d$ (-1)s get assigned to the first half. This means that exactly $U+d$ $(+1)$s get assigned to the first half. Consequently, the second half will contain exactly $D+d$ (-1)s and $U-d$ $(+1)$s.

Let us first compute this probability.

$$p_d = \frac{D_{left} \cdot D_{right}}{D_{tot}}$$

Where D_{left} denotes the number of valid starting sequences (first half) and D_{right} denotes the number of valid ending sequences. Here, *valid* means that each half sequence gets the appropriate number of ups and downs and the initial sums never drop below $1-k$. For, D_{right}, we will start the Dyck path from the end of the $2S$ sequence. In this case the invalidation threshold will be a different k'. This k' is the final height of the $2S$ sequence. So, $k' = k + 2U - 2D = k + 4S - 2D$. We will use this fact extensively moving forward.

Also, D_{tot} is the total number of possible sequences of length $2S$, given the initial conditions. This value is considered by constructing paths in the original direction i.e. the value of k is the same.

1.3 The Simple Case

The problem of sampling reduces to the binomial sampling case when $k > \mathcal{O}(\log n)\sqrt{S}$ for some constant c. In this case, the we can simply approximate the probability as

$$\frac{\binom{S}{D-d} \cdot \binom{S}{D+d}}{\binom{2S}{2D}}$$

This is because the random paths will not have initial sums less than $1-k$ with high probability. Note that this uses the assumption that we have an increasing path.

1.4 Path Segments Close to Zero

The problem arises when we $k < \mathcal{O}(\log n)\sqrt{S}$. In this case we need to compute the actual probability, Using the formula from [Reu14], we find that.

$$D_{left} = \binom{S}{D-d} - \binom{S}{D-d-k} \qquad D_{right} = \binom{S}{U-d} - \binom{S}{U-d-k'} \tag{1}$$

Here, $k' = k + 2U - 2D$, and so $k' = \mathcal{O}(\log n)\sqrt{S}$.

prove using Lemma 1

Finally, we compute the total number of Dyck paths as

$$D_{tot} = \binom{2S}{2D} - \binom{2S}{2D-k}$$

Now, we are going to use the following Lemma from [GGN10].

Lemma 2. *Let $\{p_i\}$ and $\{q_i\}$ be distributions satisfying the following conditions*

1. *There is a poly-time algorithm to approximate p_i and q_i up to $\pm n^{-2}$*
2. *Generating an index i according to q_i is closely implementable.*
3. *There exists a poly(logn)-time recognizable set S such that*
   - $1 - \sum_{i \in S} p_i$ *is negligible*
   - *There exists a constant c such that for every i, it holds that $p_i \leq \log^{\mathcal{O}(1)} n \cdot q_i$*

Then, generating an index i according to the distribution $\{p_i\}$ is closely-implementable.

In this process, we will first disregard all values of d where $|d| > \Theta(\sqrt{S})$. The probability mass associated with these values can be shown to be negligible .

bound variance of path

Next, we will construct an appropriate $\{q_i\}$ and show that $p_d < \log^{\mathcal{O}(1)} n \cdot q_d$ for all $|d| < \Theta(\sqrt{S})$ and some constant c. We will use the following distribution

$$q_d = \frac{\binom{S}{D-d} \cdot \binom{S}{D+d}}{\binom{2S}{2D}} = \frac{\binom{S}{D-d} \cdot \binom{S}{U-d}}{\binom{2S}{2D}}$$

It is shown in [GGN10] that this distribution is closely implementable.

Lemma 3. *First we show that $D_{left} \leq \frac{c_1 \cdot k}{\sqrt{S}} \cdot \binom{S}{D-d}$ for some constant c_1.*

Proof. This involves some simple manipulations.

$$D_{left} = \binom{S}{D-d} - \binom{S}{D-d-k} \tag{2}$$

$$= \binom{S}{D-d} \cdot \left[1 - \frac{(D-d)(D-d-1)\cdots(D-d-k+1)}{(S-D-d+k)(S-D-d+k-1)\cdots(S-D-d+1)}\right] \tag{3}$$

$$\leq \binom{S}{D-d} \cdot \left[1 - \left(\frac{D-d-k+1}{S-D+d+k}\right)^k\right] \tag{4}$$

$$\leq \binom{S}{D-d} \cdot \left[1 - \left(\frac{U+d+k-(U-D+d+k-1)}{U+d+k}\right)^k\right] \tag{5}$$

$$\leq \binom{S}{D-d} \cdot \left[1 - \left(\frac{U+d+k-\mathcal{O}(\sqrt{U})}{U+d+k}\right)^k\right] \tag{6}$$

$$\leq \frac{k}{\Theta(\sqrt{S})} \cdot \binom{S}{D-d} \tag{7}$$

□

Lemma 4. *Similarly, we show that* $D_{right} < \frac{c_2 \cdot k'}{\sqrt{S}} \cdot \binom{S}{U-d}$ *for some constant* c_2*.*

Proof.

$$D_{right} = \binom{S}{U-d} - \binom{S}{U-d-k'} \tag{8}$$

$$= \binom{S}{U-d} \cdot \left[1 - \frac{(U-d)(U-d-1)\cdots(U-d-k'+1)}{(S-U-d+k')(S-U-d+k'-1)\cdots(S-U-d+1)}\right] \tag{9}$$

$$\leq \binom{S}{U-d} \cdot \left[1 - \left(\frac{U-d-k'+1}{S-U+d+k'}\right)^{k'}\right] \tag{10}$$

$$\leq \binom{S}{U-d} \cdot \left[1 - \left(\frac{2D-U-d-k+1}{2U-D+k+d}\right)^{k'}\right] \tag{11}$$

$$\leq \binom{S}{U-d} \cdot \left[1 - \left(\frac{U+k+d-(2U-2D+2d+2k-1)}{U+k+d}\right)^{k'}\right] \tag{12}$$

$$\leq \binom{S}{U-d} \cdot \left[1 - \left(\frac{U+k+d-\mathcal{O}(\sqrt{U})}{U+k+d}\right)^{k'}\right] \tag{13}$$

$$\leq \frac{k'}{\Theta(\sqrt{S})} \cdot \binom{S}{U-d} \tag{14}$$

□

Finally, we need to lower bound the value of D_{tot}.

Lemma 5. *We claim that* $D_{tot} < \frac{c_3 \cdot k \cdot k'}{S} \cdot \binom{2S}{2D}$ *for some constant* c_3.

Proof.

$$D_{tot} = \binom{2S}{2D} - \binom{2S}{2D-k} \tag{15}$$

$$= \binom{2S}{2D} \cdot \left[1 - \frac{(2D)(2D-1)\cdots(2D-k+1)}{(2S-2D+k)(2S-2D+k-1)\cdots(2S-2D+1)}\right] \tag{16}$$

$$\geq \binom{2S}{2D} \cdot \left[1 - \left(\frac{2D-k+1}{2S-2D+1}\right)^k\right] \tag{17}$$

$$\geq \binom{2S}{2D} \cdot \left[1 - \left(\frac{2U-(2U-2D+k-1)}{2U+1}\right)^k\right] \tag{18}$$

$$\geq \binom{2S}{2D} \cdot \left[1 - \left(\frac{(2U+1)-k'}{2U+1}\right)^k\right] \tag{19}$$

$$\geq \frac{k \cdot k'}{\Theta(S)} \cdot \binom{2S}{2D} \tag{20}$$

□

Theorem 1. *We can now put these lemmas together to show that* $p_d/q_d \leq c = c_1 \cdot c_2/c_3 = \Theta(1)$. *This satisfies all the conditions of Lemma 2 from [GGN10]. We simply need to set the accept probability less than* $p_d/(c \cdot q_d)$.

2 Random Coloring of a Graph

We wish to locally sample an uniformly random coloring of a graph. A q-coloring of a graph $G = (V, E)$ is a function $\sigma : V \to [q]$, such that for all $(u, v) \in E$, $\sigma_u \neq \sigma_v$. We will consider only bounded degree graphs, i.e. graphs with max degree $\leq \Delta$. Otherwise, the coloring problem becomes NP-hard. cite

Using the technique of path-coupling, Vigoda showed that for $q > 2\Delta$, one can sample an uniformly random coloring by using a MCMC algorithm. cite

2.1 Glauber Dynamics

The Markov Chain proceeds in T steps. The state of the chain at time t is given by $\mathbf{X}^t \in [q]^{|V|}$. Specifically, the color of vertex v at step t is $\mathbf{X}_v^t$.

In each step of the Markov process, a pair $(v, c) \in V \times [q]$ is sampled uniformly at random. Subsequently, if the recoloring of vertex v with color c does not result in a conflict with v's neighbors, i.e. $c \notin \{X_u^t : u \in \Gamma(v)\}$, then the vertex is recolored i.e. $X_v^{t+1} \leftarrow c$.

After running the MC for $T = \mathcal{O}(n \log n)$ steps we reach the stationary distribution (ϵ close), and the coloring is an uniformly random one.

2.2 Local Coloring Algortihm

Given a vertex v, the local-access generator has to output the color of v after running $T = \mathcal{O}(n \log n) = k \cdot n \log n$ steps of Glauber Dynamics where k is a constant. For this algorithm to work, we will take $q > 2k\Delta \log n = \mathcal{O}(\Delta \log n)$.

We can consider T iterations of the above MC, and the corresponding vertex and color samples.

$$\langle (v_1, c_1), (v_2, c_2), (v_3, c_3), \cdots, (v_T, c_T) \rangle \sim_{\mathcal{U}} (V \times [q])^T$$

A position i in the sequence is labeled *"ACCEPT"* if at the i^{th} step, v_i was recolored to c_i (no conflicts with neighbors). Otherwise, position i is marked *"REJECT"*.

Given a vertex v, we consider all instances of $(v, *)$, where $* \in [q]$. Let the last such occurrence be (v, c_t). We now need to compute whether position i was marked *"ACCEPT"* or *"REJECT"*.

2.3 Modified Glauber Dynamics

Now we define a modified Markov Chain, with each step called an epoch. In the i^{th} epoch, denoted by $\mathcal{E}_i$,

- Pick a random permutation $\pi^{(i)}$ of the vertices V.
- Sample $n = |V|$ colors $\langle c_1, c_2, \cdots, c_n \rangle$ from $[q]$.
- Perform the standard update using the pairs $\left\langle (\pi_1^{(i)}, c_1), (\pi_2^{(i)}, c_2), \cdots, (\pi_n^{(i)}, c_n) \right\rangle$.

Theorem 2. *After $k \log n$ epochs, the Markov Chain is mixed.*

References

[GGN10] Oded Goldreich, Shafi Goldwasser, and Asaf Nussboim. On the implementation of huge random objects. *SIAM Journal on Computing*, 39(7):2761–2822, 2010.

[NR02] Moni Naor and Omer Reingold. Constructing pseudo-random permutations with a prescribed structure. *Journal of Cryptology*, 15(2):97–102, 2002.

[Reu14] Shlomi Reuveni. Catalan's trapezoids. *Probability in the Engineering and Informational Sciences*, 28(03):353–361, 2014.