A Deep Dive into Activated Random Walks and Layer Percolation

Adam Bougaev

May 15, 2024

Abstract

Self-organized criticality is a property of dynamical systems that is being studied as one of the means that complexity develops in nature. Model exhibiting self-organized criticality have been used to model certain events like forest fires, earthquake fault lines, and so on. The first of such models was the Abelian Sandpile Models are the model first presented by Physicists Per Bak, Chao Tang, and Kurt Weisenfeld in their 1987 paper (see [BTW87]). Essentially, models displaying self-organized criticality tend to reach a critical point between what is called a phase transition, which is effectively the same kind of phase transition we see in matter. Because of the possibility of applying this property to nature's mechanisms, the models displaying them are of interest to physicists and other natural scientists. In this paper we explore the Activated Random Walk, a model that in certain forms does exhibit self-organized criticality, and explore the mathematics behind a recently created stochastic process by my advisor coined Layer Percolation, and how we can use the mathematical findings from Layer Percolation to discover new properties relating back to Activated Random Walks in order to make new claims about the nature of the self-organized criticality exhibited in Activated Random Walks.

This paper demonstrates a lower bound as well as a generic lower bound for arbitrary distributions. This will be effective at further studying the models and the finding themselves will help us better understand the model and get us closer to acceptably bounding the critical density that tells us at what point phase transitions occur on this model.

1 Defining Activated Random Walks

In this section we will define the Activated Random Walk for clarity. Various properties of the model will be proven in later sections.

The Activated Random Walk (ARW) is defined by a grid of cells that are in a discrete number of states.

For the purpose of this paper, when referring to the Activated Random Walk, we will be referring to the cells as particles. We will also be referring to the specific instance of

the model where sites for particles can be labelled by an integer on $\mathbb{Z}$. For additional information, see [Rol20]

At every site $v \in \mathbb{Z}$ There are a certain number of particles ≥ 0.

Furthermore, every site contains an infinite set of instructions, of which the following are defined:

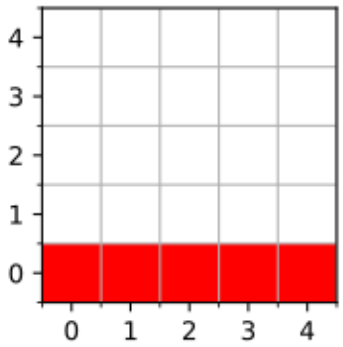
I. The left instruction removes one particle for site v and adds one particle to site $v - 1$.

II. The right instruction removes one particle for site v and adds one particle to site $v + 1$.

III. The sleep instruction will put the site to sleep only if there is one particle on the site. If there is more than one particle on the site when a sleep instruction is executed, the instruction does nothing. While the site is asleep, no instructions can be executed. The site will "wake up" if a particle from a right or left instruction on adjacent sites moves a particle to the sleeping site. Once this occurs, instructions can again be executed on the site.

When no particles are on a site, no instructions can be executed. Once a particle has moved to an empty site by executing instructions on other sites, execution on the empty site can resume.

We can visually represent this form of the Activated Random Walk in a manner like this:



R L R R L
L L L R R
S S R R R
S L R R L
L R S S L

Figure 1: ARW with 1 particle initially at all sites

In this example, we uniformly placed 1 particle at sites 0 through 4, as well as a set of instructions defined below. We can begin to execute instructions. Below is one instance of this action:

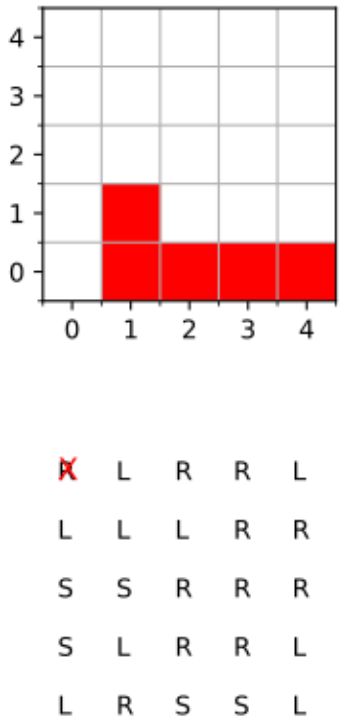


Figure 2: Execution of instruction at site 0

In this example, an instruction was executed at site 0, moving 1 particle to site 1, as the instruction was a right instruction. In this state, no instructions can be executed at site 0.

We can now execute another instruction:

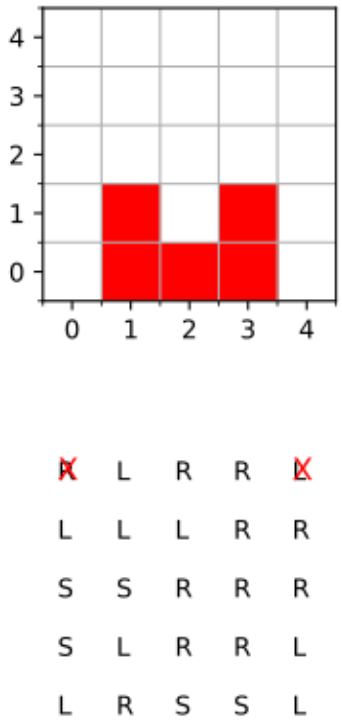


Figure 3: Execution of instruction at site 4

In this example, an instruction was executed at site 4, moving 1 particle to site 3, as the instruction was a left instruction. In this state, no instructions can be executed at site 4.

This example can be taken to an arbitrary number of executed instructions.

As alluded to by its name, Activated Random Walks have a property wherein the order of executing instructions is irrelevant to a certain state of the system. Only the number of instructions executed at each site (given the list of instructions for these sites) is important to encode the state of the system. That state is known as a configuration.

Definition 1. *By taking the number of instructions we execute at each site to get a certain configuration, we can get a function known as an odometer function, where our input is a site and the output is the number of instructions executed.*

It must also be noted that the instructions are commonly generated via the following parameters:

I. Right and left instructions are created with equal probability $\frac{1}{2}$

II. Sleep instructions are inserted between right and left instructions at a rate of $\frac{\lambda}{1+\lambda}$, where λ is a parameter from 0 to ∞ to modulate how often sleep instructions

occur.

You also place a cell at each site with probability ρ at the beginning of the process. The example displayed above has this parameter set at 1, which means all sites have a particle placed.

The version that takes place on $\mathbb{Z}$ as described above is the *fixed-energy model*, and a critical property of this model is that it can undergo what is called a *phase transition*

When ρ goes beyond what is called the critical density p_c, the model has been shown to never fall asleep, and inversely is guaranteed to fall asleep below this parameter.

A version of the model that takes place on a finite boundary exists, wherein particles that fall beyond the boundary are destroyed. When the system reaches a state where all particles are destroyed or asleep, we add a new particle. This is called the *driven-dissipative model*.

One of the defining features of the driven-dissipative model is that it exhibits self-organized criticality. In this model, the process naturally converges to the critical density.

[BGH18], [HRR23], [RS12] have shown that the critical density must be strictly between 0 and 1 with certainty, but finding the critical density is nontrivial, and so work is being done to discover bounds for this critical density. We will discuss finding new lower bounds in later sections.

2 Least Action Principle

One of the basic principles that can be proven about ARWs is the Least Action Principle. This principle is associated with the property of sites being "stable". When site v is stable, There are either no particles on the site or 1 sleeping particle. In either case, instructions cannot be executed at the site, and so it is stable.

2.1 Definitions

I. For an ARW with sites on the integer line, let a legal execution of an instruction be one executed on a site with at least one nonsleeping particle.

II. $\sigma(v)$ is a function defined by stating the number of particles on a site before executing any instructions

III. $u_{left}(v)$ is a function defined as the number of left instructions executed on a site within the first $u(v)$ instructions, where a particle on a site is moved to the left adjacent site

IV. $u_{right}(v)$ is a function defined as the number of right instructions executed on a site within the first $u(v)$ instructions, where a particle on a site is moved to the right adjacent site

V. Let the odometer resulting in a stable configuration $u(v)$ be one that meets the following requirements,

(i) $\sigma(v) - u_{left}(v) + u_{left}(v+1) - u_{right}(v) + u_{right}(v-1) \in \{0, 1\}$

(ii) If the quantity defined in (i) is 1, than instruction $u(v)$ was a sleep instruction

We can call this a stabilizing odometer.

The Least Action Principle Claims that if we were to execute instructions until we reached a stable configuration, we must have executed a smaller or equal number of instructions at every site than any stabilizing odometer.

Theorem 2.1 (Least Action Principle)**.** *Let $u^*(v)$ be the number of instructions executed at site v for some sequence of legal instructions executed that results in a stable configuration*

Let the odometer resulting in a stable configuration be called $u(v)$

For any $u(v)$, $u^*(v) \leq u(v)$

Proof. In this model we begin executing instructions until each site is either stable or we have executed $u(v)$ instructions. Call the odometer function that counts the number of instructions executed this way for each site v $w(v)$

Assume we have a site α where we have executed $u(\alpha)$ instructions but have not satisfied (i) and (ii) for stability. So, $u(\alpha) = w(\alpha)$.

Because it is known how many instructions we have executed, we know that

$$u_{left}(\alpha) = w_{left}(\alpha)$$

$$u_{right}(\alpha) = w_{right}(\alpha)$$

Because we have executed $u(\alpha)$ instructions on site α, we know $u_{left}(\alpha)$ and $u_{right}(\alpha)$ While we are unsure of the number of commands executed by site $\alpha - 1$ and $\alpha + 1$, we know that the total number of commands executed by those sites is bounded by $u(\alpha - 1)$ and $u(\alpha + 1)$, respectively.

$$w(\alpha - 1) \leq u(\alpha - 1)$$

$$w(\alpha + 1) \leq u(\alpha + 1)$$

By this fact, we also know,

$$w_{left}(\alpha - 1) \leq u_{left}(\alpha - 1)$$
$$w_{right}(\alpha + 1) \leq u_{right}(\alpha + 1)$$

Thus,

$$\begin{aligned}&\sigma(\alpha) - w_{left}(\alpha) + w_{left}(\alpha+1) - w_{right}(\alpha) + w_{right}(\alpha-1)\\ &\quad\leq \sigma(\alpha) - u_{left}(\alpha) + u_{left}(\alpha+1) - u_{right}(\alpha) + u_{right}(\alpha-1) \qquad (1)\end{aligned}$$

We know u(v) is stabilizing, and thus satisfies (i) for site α

We claim:

$$\sigma(\alpha) - u_{left}(\alpha) + u_{left}(\alpha+1) - u_{right}(\alpha) + u_{right}(\alpha-1) \in \{0,1\}$$

Because $\sigma(\alpha) - w_{left}(\alpha) + w_{left}(\alpha+1) - w_{right}(\alpha) + w_{right}(\alpha-1)$ counts the number of particles on site α after executing a valid sequence of instructions, it is nonnegative. $\sigma(\alpha) - w_{left}(\alpha) + w_{left}(\alpha+1) - w_{right}(\alpha) + w_{right}(\alpha-1) \geq 0$. By inequality (1), the value of this equation gives us 0 or 1, satisfying (i).

If there are 0 particles on the site, then (ii) is true for $w(v)$

Because $u(\alpha)$ instructions have been executed at site α, and the configuration represented by $u(v)$ is stable at all sites, then assuming the number of particles at site α is 1, then instruction $u(\alpha)$ was a sleep instruction by (ii). Thus the site α satisfies (ii) if there is 1 particle at α for $w(v)$.

In both cases, the conditions at site α satisfy conditions (i) and (ii) for $w(v)$, the site is stable, which contradicts our assumption that site α was not stable.

Thus it must be that the odometer function $u^*(v)$ represented by executing instructions legally until we reach a stable configuration is bounded by an arbitrary stable odometer function $u(v)$ that satisfies the conditions for stability. $u^*(v) \leq u(v)$ for all sites $v \in \mathbb{Z}$ and for any arbitrary $u(v)$ □

Definition 2. *If an odometer satisfies the Least Action Principle, we call it a true odometer function*

Corollary 2.1.1 (True Odometer Functions are Unique)**.** *Let $u^*(v)$ be the number of instructions executed at site v for some sequence of legal instructions executed that results in a stable configuration, also known as our true odometer function. There is only one $u^*(v)$*

Proof. Assume there are two odometers $u^*(v)$ and $u^{**}(v)$ that satisfy the Least Action Principle.

Then,

$$u^*(v) \leq u(v)$$
$$u^{**}(v) \leq u(v)$$

The above is true for any arbitrary odometer $u(v)$. Since $u(v)$ is an arbitrary stable odometer function, it can be $u^*(v)$ or $u^{**}(v)$. Thus it must be that,

$$u^*(v) \leq u^{**}(v)$$

$$u^{**}(v) \leq u^*(v)$$

Thus,

$$u^{**}(v) = u^*(v)$$

□

3 An Algorithm for Generating Stabilizing Odometers

We choose to explore generating stabilizing odometers given some basic information about our system as a means of applying the knowledge of the system as well as set up the connection between ARW and Layer Percolation.

If we have the instructions for sites 0 to n, as well as the number of instructions executed to reach stability at site 0 and the number of particles that have moved from site 0 to site 1 (defined as our net flow), as well as $\sigma(v)$ for all sites defined, we can construct the possible stable odometers, starting with the single stable site at 0 and applying our algorithm to construct a tree of possible stabilizing odometers for the sites we have.

To concretely explain this, imagine we have the following instructions on sites 0 through 2

Site 0: SRSSLLRLRLRRLRRLRSLRRLSRR...

Site 1: RLSLRRSSRLRSLLRRLRLRSRSLR...

Site 2: SLRRLSLSRLLRSLLSLSRRSLLSRRSRRSLSLRR...

We give ourselves starting conditions to establish the stability conditions of site 0, moving to subsequent sites to stabilize each site one at a time. The conditions we give ourselves are that $\forall v \in \{0, 1, \ldots, n\}(\sigma(v) = 1)$. We give ourselves a net flow of 2 and $u(0) = 20$.

Because we have a net flow of 2, it must be the the particles we give to the next site must be 2 more than the particles the next site gives us, so:

$$u_{right}(0) - u_{left}(1) = 2$$

Since $u(0) = 20$, we know exactly how many right instructions were executed by counting, which in our example is 9, and so the number of left instructions is 7.

Going to site 1, we must look at the instruction number that is associated with 7 left instructions. By counting at site 1, we can easily deduce that $u(1) \in \{19, 20, 21, 22, 23\}$

We can then analyze the state of the system for each instruction count.

At $u(1) = 19$, the instruction executed is a left instruction. Since we are generating a stabilizing odometer, we need to be stable at site 1, and since the last instruction is a left, we must have 0 particles remaining at site 1. Snce we know we started at 1 particle and gained 2 by our net flow, we must lose 3 particles to the next site. So:

$$u_{right}(1) - u_{left}(2) = 3$$

Since we know we have executed 19 instructions in this scenario, we can count the number of right instruction executed at site 1, and so we get $u_{right}(1) = 8$.

Knowing this, it is obvious that $u_{left}(2) = 5$ to balance our equality.

For another scenario, we can look at $u(1) = 21$, in this case, the last instruction executed was a sleep instruction, which means we must have had at least 1 particle remaining. We received 2 particles from site 0, and so our net flow at site 1 is now 2.

$$u_{right}(1) - u_{left}(2) = 2$$

Counting right instructions, we get $u_{right}(1) = 9$. From there, $u_{left}(2) = 7$.

Apply the algorithm recursively, we can generate stabilizing odometers for a finite set of sites in the form of a tree.

4 Introducing Layer Percolation

Layer Percolation is a stochastic process that is distinct from ARW but can be used to derive important details about ARW.

Layer Percolation operates in steps involving a grid of cells (much like ARW) but follows a new rule set.

4.1 Overview

Start with a graph defined by 2 axes, the R axis and the S_n axis. Start with one cell at $(0, 0)$.

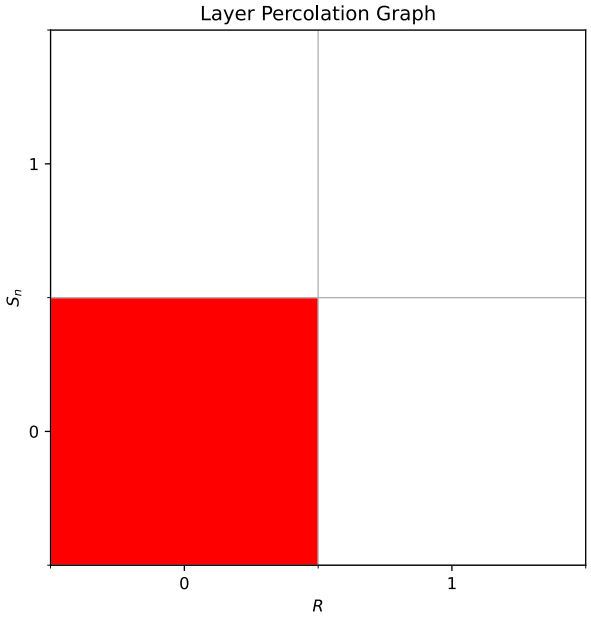


Figure 4: Initial State of Layer Percolation Model (Step 0)

Subsequently, we will define a layer shape for our next step, which we will label step 1. In order to generate layer shapes, we build shapes of cells by the following two conditions:

I. Designate the width of your shape by sampling a random variable from the Distribution $1 + Geo(\frac{1}{2})$. The number sampled results in a bottom layer of cells.

II. For each cell on this new bottom layer, sample from $Ber(\frac{\lambda}{1+\lambda})$, if 1 then add a cell on top of the cell you are currently sampling for from the bottom layer. Otherwise do nothing.

To demonstrate this in effect, here is a basic layer shape as defined by these rules:

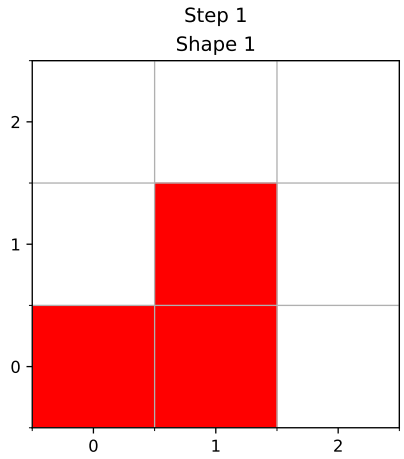


Figure 5: Example Layer shape

Insert this layer shape in place of the original cell, we get the following result:

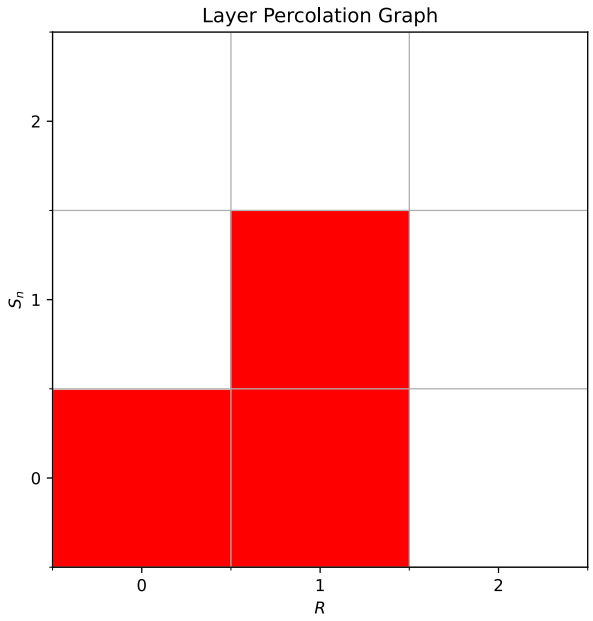


Figure 6: Step 1 Percolation Graph

Now for the next step, we define layer shapes as before, but we must define multiple for the coming step.

We must first separate the grid into diagonals. Cells belong to the same diagonal n if the sum of their coordinates on the S_n and R are n.

As an example if there are two cells in a graph that have coordinates $(3, 2)$ and $(5, 0)$, they both would belong to the same diagonal 5.

For each diagonal that contains at least 1 cell, define a layer shape. For the next step in our example, we must define 3 layer shapes.

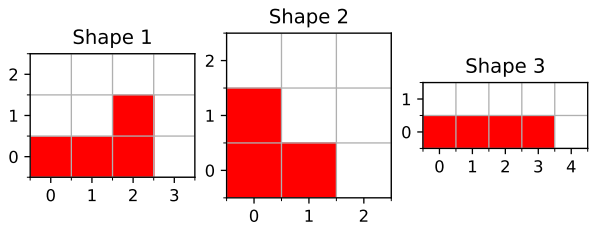


Figure 7: Layer Shapes for Step 2

For each cell on a diagonal, insert a shape starting at the same S_n height as the cell that is inserting the layer shape. As before, diagonal 0 contains only the first cell, which means inserting the first layer shape at $(0, 0)$. Subsequent diagonals insert their layer shapes at the final R coordinate where the layer shapes of the last diagonal fell. Thus each layer shape overlaps with the layer shape that came before and after it by 1 column. Here is what step 2 would look like once performing these steps:

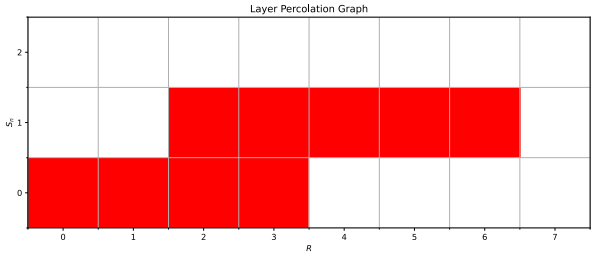


Figure 8: Step 2 Percolation Graph

This process is repeated as many times as you would want to simulate.

4.2 Significance

Earlier research has found that these layer percolation graphs can provide insight into ARW. Particularly, there is a connection between Layer Percolation and the algorithm performed in the last section to find stabilizing odometers. For example, you can see the bottom layer of cells in our layer shapes as the number of right instructions before the next left instruction, as knowing the number of right instructions following the amount of left instructions known for the next site in our stabilizing odometer generating algorithm allows us to trace how many paths each level takes in the tree of possible stabilizing odometers.

In terms of the upper layer of cells which can be generated over the base layer of the layer shape, this corresponds directly to a sleep instruction following the number of right instructions specified by the base layer up to and including the column of the relevant cell.

Understanding the layer shapes as a result of the trees generated by our algorithm allows us to understand why the layer shapes are defined by diagonals of cells rather than individual cells. A sleep instruction forces you to leave a single cell, requiring an extra cell to maintain the balanced equation. The adjacent diagonals would all correspond to the same segment of instructions at the next site.

From this, we could see the layer shapes as a segment of instructions. As an example, looking back at 4.1, we can derive the set of instructions as being "LRRS" (where every segment starts with a left instruction by the nature of how we get our segments).

In [HJJ], it has been shown that the critical density of the driven-dissipative model is equal to a constant ρ^* that is defined as the limit of the highest row reached at step n (worded differently, the limiting growth rate). We also can show that various other models of activated random walk relate to this constant in the same way. For example, the fixed-energy model on the integer line has a phase transition at some critical density ρ_c. it has been proven that this critical density ρ_c is equal to ρ^*.

4.3 Finding a Better Lower Bound

Knowing all this, if we can bound the probability at which the model will rise, we can bound the critical density.

We can start our search by realizing that every layer shape is a chance to rise up a level. Given that there is at least 1 cell at a certain level of S_n, we know that there will be at least one layer shape generated at that level. So a bound is minimally the probability that a single layer shape has a height of 2 given the parameters for creating a layer shape.

We know that the width is generated by sampling from $1 + Geo(\frac{1}{2})$ and each of these cells has a second cell above with probability $\frac{\lambda}{1+\lambda}$.

It suffice to take 1 minus the probability the shape does not have a height of 2.

By summing the probabilities of each width generating no second layer, and subtracting

this value from 1, we can achieve a better lower bound for the critical density:

$$1 - \sum_{i=1}^{\infty} (\frac{1}{2})^i (1 - \frac{\lambda}{1+\lambda})^i = \frac{2\lambda}{1+2\lambda}$$

Since $\frac{2\lambda}{1+2\lambda} - \frac{\lambda}{1+\lambda} = \frac{\lambda}{(1+\lambda)(1+2\lambda)}$, it is obvious that $\frac{2\lambda}{1+2\lambda} - \frac{\lambda}{1+\lambda} \geq 0$ for all $\lambda \geq 0$, thus we have achieved a better lower bound.

4.4 Generalizing the Bound

We can take our lower bound and extend it to any probability distribution $1 + Geo(\frac{1}{n})$ where $n \geq 1$ by solving for:

$$1 - \sum_{i=1}^{\infty} (\frac{1}{n})^i (1 - \frac{\lambda}{1+\lambda})^i = \frac{n + n\lambda - 2}{n + n\lambda - 1}$$

4.5 Further Work

We can utilize what we know about conditional probability to uncover the biased distribution for when the shape generated does not rise. Intuitively, this distribution should result in smaller shapes than the distribution that makes no assumptions about rising.

To derive this distribution, we need to calculate the probability that the width of a shape $R = n$ given that we do not rise. We can utilize Bayes' Theorem to make this calculation:

$$P[R = n | S = 0] = \frac{(\frac{1}{1+\lambda})^n (\frac{1}{2})^n}{\frac{1}{1+2\lambda}} = \left(\frac{1}{2(1+\lambda)}\right)^n (1 + 2\lambda) \tag{2}$$

Equation 2 can be written as and corresponds to a geometric distribution $1 + Geo\left(\frac{1+2\lambda}{2+2\lambda}\right)$. This gives the biased size distribution of the shapes assuming we don't rise.

Furthermore, we can attempt to uncover the probability that at the second step of layer percolation, when we create layer shapes for each diagonal of cells from the first layer shape created, we do not rise:

$$\sum_{i=1}^{\infty} \left(\frac{1}{1+2\lambda}\right)^i (p) (1-p)^{i-1} = \frac{p}{p+2\lambda}$$

Where p is the arbitrary probability associated with the geometric distribution of our first layer shape width from step 1.

References

[BGH18] Riddhipratim Basu, Shirshendu Ganguly, and Christopher Hoffman, *Non-fixation for conservative stochastic dynamics on the line*, Comm. Math. Phys. **358** (2018), no. 3, 1151–1185. MR 3778354

[BTW87] P. Bak, C. Tang, and K. Wiesenfeld, *Self-organized criticality: An explanation of the* $1/f$ *noise*, Phys. Rev. Lett. **59** (1987), 381.

[HJJ] Christopher Hoffman, Tobias Johnson, and Matthew Junge, *Universality for activated random walk*, work in preparation.

[HRR23] Christopher Hoffman, Jacob Richey, and Leonardo T. Rolla, *Active phase for activated random walk on* $\mathbb{Z}$, Comm. Math. Phys. **399** (2023), no. 2, 717–735. MR 4576759

[Rol20] Leonardo T. Rolla, *Activated random walks on* $\mathbb{Z}^d$, Probab. Surv. **17** (2020), 478–544. MR 4152668

[RS12] Leonardo T. Rolla and Vladas Sidoravicius, *Absorbing-state phase transition for driven-dissipative stochastic dynamics on* $\mathbb{Z}$, Invent. Math. **188** (2012), no. 1, 127–150. MR 2897694