university of groningen / faculty of science and engineering

Master's Thesis

Towards Classifying Bootstrap Percolation on Cayley Graphs

Author: Bart Marinissen BSc
Supervisor: prof. dr. Gerard R. Renardel de Lavalette
Supervisor: Daniel Rodrigues Valesin, PhD

Abstract

k-threshold bootstrap percolation is a very simple model of infection processes. We study how this model behaves on Cayley-graphs of amenable graphs. This is guided by the conjecture that on these graphs, either complete infection is guaranteed or impossible depending on the threshold k. To this end, we prove this conjecture holds for abelian groups of rank 2. We also find at which k complete infection is guaranteed. These results are essentially an extension of methods used in [Sch92] to prove the conjecture for Cayley-graphs on $\mathbb{Z}^n$ using canonical generators. Finally, we outline an approach that might extend these results to all Cayley graphs of abelian groups. Moreover, we test the conjecture experimentally on Cayley-graphs of Heisenberg and Lamplighter groups. Here we find evidence that the conjecture holds for the Heisenberg groups. The results for the lamplighter groups are inconclusive. However, we do find a very interesting phenomenon with the lamplighter group. The number of final infected nodes in these graphs seems to cluster around tenths of the total number of nodes in the graph.

Sunday 25th March, 2018

Contents

Chapter 1

Introduction

Take a locally finite graph $G = (V, E)$. That is, a graph where every node has finitely many neighbors. A vertex in this graph can either be occupied (value 1) or vacant (value 0). We then consider the following increasing discrete time dynamics. At time step t, a vacant vertex with more than k occupied neighbors becomes active in step $t+1$. This process is called k-threshold bootstrap percolation. Formally, we take A to be the set of occupied points at a given time step. We then define a single step by the following function:

$$\beta_k(A) = A \cup \{x \in V \mid \#E \cap (\{x\} \times A) \geq k\} \tag{1.0.1}$$

We are then interested in the behaviour of this system as we let time go to infinity. That is, we are interested in ${\beta_k}^\infty(A) = \bigcup_{i=1}^{\infty} {\beta_k}^i$.

This model was first studied in [CLR79] as a low temperature approximation to the Ising model, which models magnetic spins in a solid. In general, bootstrap percolation is a simple model for any infectious process. For example, the spread of disease through a population, the firing of neurons in a brain, the spread of opinions in a social network, or the seeping of water through coffee grounds against pressure. We will study what happens when the process is started from a random uniform configuration of occupied vertices A_0. That is, we pick some density $p \in [0, 1]$. Then, for any vertex $v \in V$ we have $\mathbb{P}(v \in A_0) = p$ and this is independent of all other vertices. We say an initial condition percolates if ${\beta_k}^\infty(A_0) = V$.

In [CLR79] the studied graphs were the $2n$-ary trees. Not much later, Van Enter [Ent87] studied this process on the 2D grid $\mathbb{Z}^2$ with $k = 2$. He found that for any positive density, percolation occurs with probability 1. This result sparked our interest, eventually leading to this thesis.

Besides k-threshold bootstrap percolation, there are other processes also referred to as bootstrap percolation. These are all increasing processes where vacant vertices become occupied as an increasing function of their neighborhood. We say one percolation process dominates another whenever the growth function of the former dominates the growth function of the latter. If a process A dominates another process B and B percolates, then this guarantees that A also percolates. This will mostly be used by defining a new form of percolation, proving results for the new form, and then using domination to apply those results to k-threshold bootstrap percolation.

Both kinds of graphs we mentioned before (the grids $\mathbb{Z}^d$ and the $2n$-ary tree) are examples of Cayley graphs. In this paper, we will study bootstrap percolation on Cayley graphs of amenable groups. Before we explain what an amenable group is, we first explain Cayley graphs. They are defined as follows:

Definition 1.0.1 (Cayley graph)**.** *Given a group G with operation $\circ$ and a finite set $S \subseteq G$ (called the generating set) we define the Cayley graph on G using generating set S as $\Gamma(G, S) = (V, E)$ taking:*

$$V = G$$
$$E = \{(x, x \circ s) \mid s \in S, x \in G\}$$

One can take a Cayley graph to be directed or undirected. We say a generating set S is a *symmetric set* if it is closed under inversion. That is if:

$$x \in S \implies x^{-1} \in S$$

If the generating set of a Cayley graph is symmetric, the directed and undirected graphs are essentially the same.

Cayley graphs are vertex transitive (defined below). This property together with their constructive nature really helps when studying these graphs. This is part of why we chose to study Cayley graphs.
Definition 1.0.2 (Vertex Transitive graphs). *Given a graph $G = (V, E)$ we say G is vertex transitive if $Aut(G)$ acts transitively on V. That is, given any $x, y \in V$ there exists an $\alpha \in Aut(G)$ such that $\alpha(x) = y$. Or equivalently, the image of any vertex x under $Aut(G)$ is all nodes V.*

Here, $Aut(G)$ is the group of all graph automorphisms of G. That is:

$$Aut(G) = \{F : V \to V \mid (x, y) \in E \iff (F(x), F(y)) \in E\}$$

Lemma 1.0.3. *All Cayley graphs are vertex transitive*

Proof. Given a Cayley graph $\Gamma(G, S)$ we have the following group of automorphisms:

$$\{x \mapsto gx \mid g \in G\}$$

One can see these are automorphisms because given any $g \in G$ an edge $(x, xs) \in E$ with $s \in S$ has image:

$$(x, xs) \mapsto (gx, gxs)$$

This pair (gx, gxs) is again an edge since $s \in S$. Now, given an arbitrary pair $a, b \in G$ we have the automorphism $x \mapsto ba^{-1}x$ such that $a \mapsto b$. ■

So given a graph G and a bootstrap percolation process $\beta : 2^V \to 2^V$ we are interested in the behavior of β when applied iteratively to some initial condition A_0 with density p. To this end, we introduce some variables that describe this behavior. Whenever we refer to the variables, it will be clear from context which graph G and bootstrap process β are meant. Here, we presume that β is an increasing and extensive function.

- T, the stopping time. This is a random variable, defined as:

$$T = \inf\{t \geq 0 \mid i \in \beta^t(A_0)\} \tag{1.0.2}$$

  Where i is the group identity.

- p_c, the critical point for T:

$$p_c = \inf\{p \in [0, 1] \mid \mathbb{P}_p(T < \infty) = 1\} \tag{1.0.3}$$

- $\gamma(p)$, the exponential growth rate:

$$\gamma(p) = \sup\{\gamma \geq 0 \mid \exists C < \infty \ \forall t : \mathbb{P}_p(T > t) \leq Ce^{-\gamma t}\} \tag{1.0.4}$$

- π_c, the critical point for exponential growth:

$$\pi_c = \inf\{p \in [0, 1] \mid \gamma(p) > 0\} \tag{1.0.5}$$

Now, there are some simple relations between these variables: $\gamma(p)$ is increasing in p; the expectation of T is decreasing in p; if $\gamma(p) > 0$ then $\mathbb{P}_p(T < \infty) = 1$ and thus $\pi_c \geq p_c$. We say percolation is exponential when $\gamma(p) > 0$.

Throughout this paper we will often use sequences. For these, we introduce some notation. By $[x_i]$ we mean a sequence $x_0, x_1 \ldots$. Such a sequence can be either finite or infinite. The difference will be clear from context. Moreover, given a set A, we define $[A]$ as the set of sequences with elements in A. For example, we would denote the sequence of all sums of squares as:

$$[x_i] \in [\mathbb{N}] :$$
$$x_0 = 0$$
$$x_{i+1} = x_i + (i + 1)^2$$

We also define the idea of a subset of nodes being internally spanned. Intuitively, this means we take the set of nodes as our full graph, and ask whether the set gets filled. Formally we have:
Definition 1.0.4 (Internally spanned sets)**.** *Take a bootstrap process $\beta : 2^V \to 2^V$ and an initial condition $A_0 \subseteq V$.*

We then say a set $X \subseteq V$ is internally spanned by A_0 if $X \cap A_0$ percolates using β on the graph $(X, E \cap (X \times X))$.

1.1 Aim of this thesis

As stated earlier, we mean to study Cayley graphs of amenable groups. An amenable group is defined as follows:
Definition 1.1.1 (Amenable group)**.** *A discrete group G is amenable if there exists a Følner sequence for G.*

That is, we have a sequence of finite subsets $[F_i] \in \left[2^G\right]$ such that:

$$\forall g \in G \ \exists i \ \forall j \geq i : g \in F_j$$
$$\lim_{i \to \infty} \frac{\#(gF_i \vartriangle F_i)}{\#F_i} = 0$$

Where $\vartriangle$ is the symmetric difference operator.

There are many other equivalent definitions as can be seen in [BPP06]. Informally, one could consider amenable groups as those that do not 'grow too quickly'.

We are interested in amenable groups due to the following conjecture from [BPP06]:
Conjecture 1.1.2. *A group G is amenable if and only if, for every finite symmetric $S \subseteq G$ and $k \in \mathbb{N}$ we have $p_c \in \{0, 1\}$ on $\Gamma(G, S)$ for k-threshold bootstrap percolation.*

We will seek to make progress towards this conjecture in two ways. The first is a theoretical approach, contained in Chapter 2. The second approach is experimental, covered in Chapter 3.

In the theoretical approach we will look at abelian groups, these are guaranteed to be amenable. There, we will work towards the following conjecture:
Conjecture 1.1.3. *Let H be a finitely generated abelian group H and $S \subseteq H$ be a symmetric set with $0 \notin S$. Then set k_S equal to half the number of non-periodic elements in S. That is, we set:*

$$k_S = \frac{\#\{s \in S \mid \forall n \in \mathbb{N}^+ : n \times s \neq 0\}}{2}$$

Then for β_k on $\Gamma(H, S)$ we have:

$$\pi_c = p_c = \begin{cases} 0 & \text{if} \quad k \leq k_S \\ 1 & \text{if} \quad k > k_S \end{cases}$$

The main result of this paper is Theorem 2.5.1 which states the above conjecture holds when H has rank 2 (the rank of an abelian group is defined in Theorem 2.1.8). We also have Theorem 2.2.12 which states the above conjecture holds for arbitrary abelian groups in the case $k > k_S$. In Section 2.5.1 we outline an idea for extending the methods of the proof of Theorem 2.5.1 to work for all finitely generated abelian groups. This would then prove Conjecture 1.1.3.

Most important to our theoretical work is [Sch92]. That paper proves a sub case of Conjecture 1.1.3. Specifically, it proves the case where $H = \mathbb{Z}^n$ and $S = \{e_1 \dots e_n\}$ where $[e_i]$ are the canonical generators for $\mathbb{Z}^n$. Our method uses the proof form that paper as a template. That paper was preceded by [AL88] and dealt with finite subsets (hypercubes) of $\mathbb{Z}^d$ rather than looking at the full space $\mathbb{Z}^d$.

Important to the conjecture itself is the paper in which it was posed: [BPP06]. That paper looks explicitly at non-amenable groups. For these groups, it seems that p_c takes intermediate values for k-threshold bootstrap percolation regardless of k. This contrasts sharply with the results known for specific amenable groups. Moreover, their results depend explicitly on groups being non-amenable.

The second approach is experimental. Here we intend to compute β_k on finite subgraphs of Cayley graphs of amenable groups. Specifically, we will be looking at the Heisenberg and lamplighter groups. Note that for any finite subgraph, we have $p_c = 1$ since the probability that $A_0 = \emptyset$ is technically positive. However, we hope to see (and do see) some form of critical behavior in these subgraphs. That is, at certain densities we see almost no growth from the initial condition. Then, for densities higher than some 'critical density', we see growth continues until almost the entire graph is occupied. Next, we study how this 'critical density' changes when we take ever larger subgraphs. If this 'critical density' converges to 0 that suggests that $p_c = 0$ for the full infinite graph.

Experiments like these were run previously for $\mathbb{Z}^2$. In these experiments they took squares and increased their side length. They ran simulations up to 350×350 squares. The critical point then seemed to converge to some small but positive value. Later, in [Hol03] we got theoretical results showing this is actually not the case. This was an expansion of the work done in [AL88]. The results show that the critical point converges to 0 as the side length L goes to infinity. However, convergence is very slow. In the case of $\mathbb{Z}^2$ it is $O(1/\log L)$ and in the case of $\mathbb{Z}^3$ it converges at a rate of $O(1/\log\log L)$. In fact, from [Bal+12] we know that in $\mathbb{Z}^d$ the critical point converges to 0 with order:

$$O\left(\frac{1}{\log^{d-1} L}\right) \tag{1.1.1}$$

where $\log^1 = \log$ and $\log^n = \log\log^{n-1}$. That is, $\log^n$ is log iterated n times. As such, finding results congruent with such convergence would suggest we have similar convergence.

For the Heisenberg group, our experiments found evidence for $O\left(\frac{1}{\log^3 L}\right)$ convergence. For the lamplighter group, our experiments were inconclusive. We did find a rather interesting phenomenon with the lamplighter group. On the binary lamplighter group, the size of the stable sets ${\beta_k}^\infty(A_0)$ seems to cluster around tenths of the full subgraph.

Chapter 2

Theoretical Work

2.1 First theoretical steps

In the section, we will lie the foundation for the remainder of this chapter. First, we define the k-fort and introduce the 0-1 law. After that, we define some basic concepts of discrete geometry and derive some propositions regarding these concepts; this will be instrumental in Section 2.4. Finally, we make use of the abelian structure theorem (2.1.8). This yields Theorem 2.1.13, which allows us to focus solely on free-abelian groups.

A very useful tool in analyzing k-threshold bootstrap percolation is the k-fort, introduced by [BPP06]:
Definition 2.1.1 (k-fort)**.** *Given a graph $G = (V, E)$, we call a subset $A \subseteq V$ a k-fort if it is connected and each vertex in A has fewer than k connections outside of A. Formally:*
A is connected and:

$$\forall a \in A : \#\big(E \cap (\{a\} \times (V \setminus A))\big) < k$$

Note that other papers often use $\leq k$ instead of $< k$ in the above definition. These are useful due to the following lemma
Lemma 2.1.2. *k-threshold bootstrap percolation fills the entire graph if and only if no k-fort is entirely vacant in the initial condition.*

Proof. Take A_0 to be an initial condition with density $p > 0$ and take V to be the set of vertices of our Cayley graph.

Now suppose that $A_\infty = {\beta_k}^\infty(A_0) \neq V$. Then we can find a connected component in ${A_\infty}^C$. This connected component must be a k-fort, as any vertex with k or more connections to the outside would not remain vacant. Now, since bootstrap percolation is an increasing process, this vacant k-fort must have been vacant in A_0.

What remains to show is that percolation implies that all k-forts were entirely vacant. We do this via contraposition. So, suppose we start with a vacant k-fort. Then, by definition of β_k, that k-fort will remain vacant. ■

Now consider what happens when we have a finite k-fort. If we have a density $p < 1$ then this finite k-fort has positive probability of being vacant. Thus, the probability of percolation is less then 1. The converse need not be true. If all k-forts are infinite, the probability that a given k-fort is vacant is 0, but if there are uncountably many k-forts, the probability that we could find a vacant one might still be positive.

We also have the following theorem, called the 0-1 law. This theorem follows from: [LP16, Prop. 7.3]; Cayley graphs being vertex transitive; the probability of complete percolation being invariant under the automorphisms of our Cayley graphs; and the event of complete occupation being translation invariant.

Theorem 2.1.3 (0-1 Law)**.** *For any infinite Cayley graph k-threshold bootstrap percolation we have*

$$\mathbb{P}_p(\textit{Complete occupation}) \in \{0, 1\}$$

We can use this theorem to deduce the following result about the existence of finite k-forts.
Lemma 2.1.4. *If a Cayley graph Γ generated by a finite S has finite k-forts then, for k-threshold bootstrap percolation, $p_c = 1$. Otherwise, it has $p_c \leq 1 - q$ where $q > 0$ is the critical probability for site percolation on Γ.*

Proof. These results are due to [BPP06].

First, we consider the case where finite k-forts exist. In this case, unless the initial density $p = 1$, the probability that a given finite k-fort is vacant is positive. Therefore, the probability of complete occupation is lower than 1. By the 0-1 law, it must then be 0.

For the case of no finite k-forts existing, if we have no complete occupation, there must exist a vacant k-fort. Since this k-fort is connected and infinite, this can only occur when the density of vacant sites is at least q. This follows directly from the definition of q. Moreover, as S is finite, $q > 0$. ■

Having looked at k-forts, we now consider the effect of growing S.
Lemma 2.1.5. *Let H be a finitely generated abelian group, and let $S, T \subseteq H$ be finite symmetric subsets of H.*

Now, if $S \subseteq T$ it follows that β_k on $\Gamma(H, S)$ dominates β_k on $\Gamma(H, T)$.

Proof. Let $\beta_{k,S}$ be β_k on $\Gamma(H, S)$ and let $\beta_{k,T}$ be β_k on $\Gamma(H, T)$. Both $\beta_{k,S}$ and $\beta_{k,T}$ are of the type: $2^G \to 2^G$.

We need to show that $\beta_{k,S}(A) \supseteq \beta_{k,T}(A)$ for any $A \subseteq G$. Now, let E_S be all edges of $\Gamma(H, S)$ and E_T be all edges of $\Gamma(H, T)$. It follows from $S \subseteq T$ that $E_S \subseteq E_T$. Thus, given any $x \in A$ we have:

$$E_S \cap (\{x\} \times A) \subseteq E_T \cap (\{x\} \times A)$$ ■

We will use this lemma to ignore periodic generators in S. Specifically, to prove Conjecture 1.1.3 in the case $k \leq k_S$ it now suffices to prove the conjecture holds when S contains no periodic elements. In this case we have $k_S = \frac{\#S}{2}$.

2.1.1 Basics of discrete geometry

Our proofs will often concern convex polytopes and other discrete geometry. Here, we introduce the discrete geometry we need for our proves.

We start with the most basic operations. Given a set $A \subseteq \mathbb{R}^n$ a point $x \in \mathbb{R}^n$ and a scalar $\alpha \in \mathbb{R}$ we define the following operations:

$$\begin{aligned} x + A &= \{x + a \mid a \in A\} = A + x \\ \alpha A &= \{\alpha a \mid a \in A\} \\ -A &= -1A \\ A - x &= A + (-x) \\ x - A &= x + (-A) \end{aligned}$$

The main subject of this section is the convex polytope. Convex polytopes have many different descriptions. We will use the following three:

- A set of the form
$$\{x \in \mathbb{R}^n \mid Mx < b\}$$
for some $M \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. Here, comparison of vectors is component-wise and we require all components to satisfy the inequality.

- A finite intersection of shifted half-spaces

$$\bigcap_i H(u_i) + b_i$$

Here, we define a half-space as:

$$H(u) = \{x \in \mathbb{R}^n \mid x \cdot u \leq 0\}$$

where $u \neq 0$. We do not define $H(0)$ as that would not be a half-space. This later saves us from needing to exclude the special case $u = 0$.

For the shifts we take: $b_i \in \mathbb{R}^n$. So we take a finite intersection of half spaces with normal u_i shifted so their border contains the point b_i.

- Bounded polytopes can be written as: $\text{Conv}(V)$ for some finite set $V \subseteq \mathbb{R}^n$. That is, the convex hull of some finite set of points. Here, the convex hull is defined as follows:

$$\text{Conv}(A) = \left\{ \sum_{a \in A} \alpha(a) a \;\middle|\; (\forall a \in A : \alpha(a) \geq 0) \wedge \sum_{a \in A} \alpha(a) = 1 \right\}$$

Besides the half-space $H(u)$ we also define the border of that half space, also known as the normal complement:

$$N(u) = \{x \in \mathbb{R}^n \mid x \cdot u = 0\}$$

where $u \neq 0$. Again, we exclude $u = 0$ to prevent needing to treat it as a special case every time we use $N(u)$ or $H(u)$. It should be noted that for shifted half-spaces of the form $H(u) + b$ different values of b can give the same shifted half-space. If we take some $d \in b + N(u)$ then $H(u) + d = H(u) + b$.

We then define the *faces* of a polytope P as follows. A set $F \subseteq \mathbb{R}^n$ is a face of P if there exist some u and b such that

$$F = P \cap (H(u) + b) = P \cap (N(u) + b)$$

Every such face F has a dimension, this is defined as $\dim(\text{span}(F - b))$ where b is the base of the shifted half-space that defines F. We call the faces with dimension $n - 1$ the *facets* of P. The union of all facets forms the boundary of P.

We then also consider the polytope P and $\emptyset$ to be faces with dimension n and -1 respectively. In this case, the faces form a lattice. That is, they form a partial order (under set inclusion) where any two elements have a unique least upper bound and greatest lower bound.

There is a special case of an unbounded polytope called the conical sum. It is defined as:

$$\text{Coni}(A) = \left\{ \sum_{a \in A} \alpha_a a \mid \forall a \in A : \alpha_a \geq 0 \right\}$$

One can see this as the union of all scaled versions of $\text{Conv}(A)$. The conical sum is closed under addition and positive scaling. Note that unlike the convex hull, the conical sum is not invariant under affine transformations. If a conical sum is not equal to the full space, it can be written in either of the following forms:

- $\{x \in \mathbb{R}^n \mid Mx \leq 0\}$ for some $M \in \mathbb{R}^{m \times n}$
- $\bigcap_i H(u_i)$

These are convex polytopes with a single vertex at the origin.

Finally, we have the Minkowski sum and difference. These are also known as dilation and erosion. The Minkowski sum $\oplus$ has three equivalent definitions.

$$\begin{aligned} A \oplus B &= \{a + b \mid a \in A \wedge b \in B\} \\ &= \bigcup_{b \in B} A + b \\ &= \bigcup_{a \in A} a + B \end{aligned}$$

One key property of the Minkowski sum is that it commutes with taking the convex hull. That is: $\text{Conv}(A \oplus B) = \text{Conv}(A) \oplus \text{Conv}(B)$. Moreover, the Minkowski sum of two convex sets is itself convex.

The Minkowski difference $\ominus$ is defined as:

$$A \ominus B = \{x \in A \mid a + B \subseteq A\}$$

Note that $\ominus$ is not the inverse of $\oplus$. Moreover, if A and B are convex, so is $A \ominus B$. In fact, this holds even if B is not convex. To see this, consider that $a + B \subseteq A$ implies that $\text{Conv}(a + B) \subseteq A$ as A is convex.

Now, let A and B be convex polytopes with $0 \in B$. Then any $x \in A \oplus B$ has many different decompositions $x = a + b$ with $a \in A$ and $b \in B$. However, we will define a unique one:

$$\begin{aligned} x' &= \arg\min \|b\| : b \in (x - A) \cap B \\ \tilde{x} &= x - x' \\ x &= x' + \tilde{x},\ x' \in B,\ \tilde{x} \in A \end{aligned} \tag{2.1.1}$$

To see this is unique, $(x - A) \cap B$ is clearly a convex polytope. Then, the following proposition proves x' is unique, from which $\tilde{x}$ must be unique.

Proposition 2.1.6. *Given a convex polytope P, the following is uniquely defined:*

$$\arg\min_{x \in P} \|x\|$$

Proof. Suppose we have two distinct points $x, y \in P$ with $\|x\| = \|y\|$. Then, by the triangle inequality and $x \neq y$ we have:

$$\left\| \frac{x}{2} + \frac{y}{2} \right\| < \|x\|$$

and by convexity we have:

$$\frac{x}{2} + \frac{y}{2} \in P$$

Thus, the above arg min must be unique. ■

Now, given any polytope B with a facet F we define

$$u(F) = u \in \mathbb{R}^2 \text{ such that } \forall b \in F : F = b + H(u) \cap B$$

This is unique up to scaling by positive real numbers. With this, we can introduce the following lemma:

Lemma 2.1.7. *Given A, B convex polytopes with $0 \in B$, let $\mathcal{F}$ be the set of facets of A then:*

$$A \oplus B = A \cup \bigcup \left\{ F \oplus B \;\middle|\; F \in \mathcal{F} \wedge u(F) \in \bigcup_{v \in B} H(v)^C \right\}$$

Proof. It is trivial to see that the right-hand side is a subset of the left-hand side. This follows from $0 \in B$ and $F \subseteq A$ for all $F \in \mathcal{F}$.

Given any $x \in A \oplus B$, take the unique decomposition $x = \tilde{x} + x'$ from Equation 2.1.1. This has $\tilde{x} \in A$, $x' \in B$ and x' is the smallest element in b that allows such a decomposition. Clearly, if $x' = 0$ then $x \in A \subseteq A \oplus B$.

Now, suppose $x' \neq 0$. Then $x = \tilde{x} + x' \notin A$. Moreover we know that:

$$\text{Conv}(\tilde{x}, \tilde{x} + x') \cap A = \{\tilde{x}\}$$

For otherwise, x' would not be minimal. But then $\tilde{x}$ must lie on the border of A, and thus lies in some facet $F \in \mathcal{F}$. Thus if $x' \neq 0$ we have:

$$x = \tilde{x} + x' \in F \oplus B$$

Moreover if $\tilde{x} \in F$ then $u(F) \in H(x')^C$. This is again a consequence of the minimality of x'.

As such, every $x \in A \oplus B$ either lies in A or lies in $F \oplus B$ for some $F \in \mathcal{F}$ with $u(F) \in \bigcup_{v \in V} H(v)^C$. ■

The above lemma means that when taking the Minkowski sum, we only need to 'dilate' at the facets. Note that in some cases, our extra requirement on the facet $u(F) \in \bigcup_{v \in B} H(v)^C$ means we can leave out some of the dilated facets.

2.1.2 Using the abelian structure theorem

As we focus on finitely generated abelian groups, the abelian structure theorem will be used often. We state it here for the sake of completeness:
Theorem 2.1.8 (Abelian structure theorem)**.** *Any finitely generated abelian group is isomorphic to a product of the form $\mathbb{Z}^r \times T$ where T is a finite abelian group. T is called the 'toroidal component' of the group and r is called the 'rank'.*

Note that in this decomposition all periodic elements lie in $S \cap \{0\} \times T$. Furthermore, as each subgroup of an abelian group is normal, we have the quotient group $(\mathbb{Z}^r \times T)/(\{0\} \times T) \simeq \mathbb{Z}^r$. This means there exists a surjective group homomorphism $\phi : \mathbb{Z}^r \times T \to \mathbb{Z}^r$. Such homomorphisms can be seen as just discarding the part in T.

One application of the abelian structure theorem is the following result.
Theorem 2.1.9. *Given a finitely generated abelian group H and a finite symmetric set $S \subseteq H \setminus \{0\}$ we set:*

$$k_S = \frac{\#\{s \in S \mid \forall n \in \mathbb{N}^+ : n \times s \neq 0\}}{2}$$

Then, all k_S-forts in $\Gamma(H, S)$ are infinite.

Proof. First, we name the set of all non-periodic elements in S

$$Q = \{s \in S \mid \forall n \in \mathbb{N}^+ : n \times s \neq 0\}$$

Note that $\#Q = 2k_S$. By the abelian structure theorem there exists a surjective homomorphism:

$$\phi : H \to \mathbb{Z}^d$$

We know that for any $x \in Q$ we have $\phi(x) \neq 0$ because x is not periodic.

Now, we can create a weak ordering (total order allowing for ties) on H by simply extending a lexicographical ordering on $\mathbb{Z}^d$. That is, if $x, y \in H$ then $x \geq y$ if and only if $\phi(x) \geq \phi(y)$. Where we write $x \equiv y$ if $x \geq y \wedge y \geq x$. This only occurs if $x - y \in T$. To see this, consider how the generators are ordered. Notably, our ordering is well behaved w.r.t. addition. That is, if $s > 0$ then $x + s > x$. This is because addition on $\mathbb{Z}^n$ also has this property under the lexicographical ordering.

As S is symmetric, Q is also symmetric . This, combined with the fact that no elements in Q are ordered equally with 0, means that exactly half of Q is larger than 0. We shall call this half Q^+. Note that $\#Q^+ = k_S$. Now, consider any finite subset $A \subseteq H$. We wish to show that such an A cannot be a k_S-fort.

As A is finite, it has 'maximal' points with respect to our order. Formally, there exist $m \in A$ such that for all $a \in A$ $m \geq a$. Now, take such an m and consider the set of points $m + Q^+$. These are all neighbours of m, because $Q^+ \subseteq S$. Moreover, all of $m + Q^+$ is larger than m in our ordering and thus lies outside A. Therefore $m \in A$ has at least $\#Q^+ = k_S$ neighbors outside of A. As such A cannot be a k_S-fort. ∎

This immediately allows us to apply Lemma 2.1.4 to get the following corollary:
Corollary 2.1.10. *Given a finitely generated abelian group H and a symmetric set of generators S pick:*

$$k = k_s = \frac{\#\{s \in S \mid \forall n \in \mathbb{N}^+ : n \times s \neq 0\}}{2}$$

Then k-threshold bootstrap percolation on $\Gamma(H, S)$ has $p_c \leq 1 - q$ where $q > 0$ is the critical probability for site percolation.

To better make use of the abelian structure theorem, we define a new form of bootstrap percolation called modified bootstrap percolation.

Definition 2.1.11 (Modified bootstrap percolation)**.** *Let G be an abelian group and take $S \subseteq G \setminus \{0\}$ to be a finite symmetric subset. We call a set $X \subseteq S$ a symmetric half of S if $X \cup -X = S$. Modified bootstrap percolation is then defined by the following discrete step, based on symmetric halves:*

$$\mu_S(A) = A \cup \{x \in A \mid \exists X \subseteq S : x + X \subseteq A \ \wedge \ X \cup -X = S\}$$

Simply stated, in order for x to become occupied by μ_S, each symmetric pair $x+s, x-s$ given an $s \in S$ has to have at least one occupied node. Now compare modified bootstrap percolation to k-threshold bootstrap percolation with $k = \frac{\#S}{2}$. We see that modified bootstrap percolation is a lot like k-threshold bootstrap percolation, only we have some more geometric requirements. Instead of allowing any half of the neighbors of a point to be occupied, we require a symmetric half to be occupied. Note that the 0-1 law (Theorem 2.1.3) still holds for modified bootstrap percolation. The underlying probability distribution remains ergodic, and the event of percolation under modified bootstrap percolation is also translation invariant.

For now, we wish to show that modified bootstrap percolation is dominated by k-threshold bootstrap percolation when $k = {}^{\#S}/_2$. This follows quite readily from the fact that $0 \notin S$. For then, if X is a symmetric half of S then $\#X \geq {}^{\#S}/_2$. This yields:

$$\mu_S(A) \subseteq \beta_{{}^{\#S}/_2}(A) \tag{2.1.2}$$

Thus if we prove that modified percolation leads to complete percolation, then we also prove that k-threshold bootstrap percolation leads to complete percolation at $k = {}^{\#S}/_2$.

We can combine the above with Lemma 2.1.5 to get the following proposition:

Proposition 2.1.12. *Let H be a finitely generated abelian group, and $S \subseteq H \setminus \{0\}$ be a finite symmetric subset. Moreover, set T to be all non-periodic elements of S.*

Then it follows that $k_S = \frac{\#T}{2}$ and:

$$\mu_T(A) \supseteq \beta_{k_S}(A)$$

Next, we introduce a theorem that takes modified bootstrap percolation without periodic generators on any abelian group and reduces it to modified bootstrap percolation on a free abelian group (i.e. a group with no toroidal part).

Theorem 2.1.13. *Let H be any abelian group isomorphic to $\mathbb{Z}^r \times T$ where T is finite. Let $S \subseteq H$ be a finite symmetric set that does not include periodic elements. Finally, let $\phi : H \to \mathbb{Z}^r$ be a surjective homomorphism.*

In that case, if $p_c = 0$ for modified bootstrap percolation on $\Gamma(\mathbb{Z}^r, \phi(S))$ then $p_c = 0$ for modified bootstrap percolation on $\Gamma(H, S)$. The same holds for π_c.

Proof. The basis of this proof is to show that modified bootstrap percolation on H dominates modified bootstrap percolation on $\mathbb{Z}^r$ in some sense.

Note that, if $H = \mathbb{Z}^r \times T$ we can simply take:

$$\phi : (z, t) \mapsto z$$

Any other surjective homomorphism $\psi : H \mapsto \mathbb{Z}^r$ can be factored as $\psi = f \circ \phi \circ g$ where $f : H \to \mathbb{Z}^r \times T$ and $g : \mathbb{Z}^r \to \mathbb{Z}^r$ are isomorphisms. Nowhere in the proof do these isomorphisms affect the reasoning. As such, once can essentially presume the above form of ϕ for the entire proof.

Now, given an initial condition $A_0 \subseteq H$ we will transform it to an initial condition $B_0 \subseteq \mathbb{Z}^r$ as follows: $B_0 = \{x \in \mathbb{Z}^r \mid \phi^{-1}(x) \subseteq A_0\}$. That is, B_0 corresponds to all totally occupied translates of T in A_0. Now for any $x \in \mathbb{Z}^r$ we have $P_p(x \in B_0) = p^{\#T}$ and these are fully independent of any other points. Thus B_0 can be seen as an initial condition with density $p^{\#T}$.

We then define a second generating set $S' = \phi(S)$. Note that the requirement that no generators be periodic means that no generator lies in T so $0 \notin \phi(S)$. We then define two sequences $[A_i]$ and $[B_i]$:

$$A_i = (\mu_S)^i(A_0)$$
$$B_i = (\mu_{S'})^i(B_0)$$

Now, we want to show that $[A_i]$ dominates $[B_i]$ in some sense, but they are sequences in different spaces. So instead we will prove the following:

$$\phi^{-1}(B_i) \subseteq A_i$$

We still call this 'domination' because as B_i grows, this also require A_i to grow. Moreover, if $B_i = \mathbb{Z}^r$ then it follows that $A_i = H$. We will prove this using induction on i. By definition of B_0 this holds for $i = 0$. What remains is to show that:

$$\phi^{-1}(B_i) \subseteq A_i \implies \phi^{-1}(B_{i+1}) \subseteq A_i$$

To this end, it suffices to show that:

$$x \in B_{i+1} \setminus B_i \implies \phi^{-1}(x) \subseteq A_{i+1} \tag{2.1.3}$$

By definition of modified bootstrap percolation, the left hand side of (2.1.3) implies there exists a symmetric half X' of S' such that:

$$x + X' \subseteq B_i$$

Now take y to be any point in $\phi^{-1}(x)$. Moreover, take $X = S \cap \phi^{-1}(X')$. Note that X is a symmetric half of S. Then the above implies:

$$y + X \subseteq \phi^{-1}(B_i) \subseteq A_i$$

Thus, by definition of μ_S we have $y \in \mu_S(A_i) = A_{i+1}$. ■

Now, suppose that modified bootstrap percolation on $\Gamma(\mathbb{Z}^r, S)$ has $p_c = 0$ for any finite symmetric S. Then the above theorem means we have $p_c = 0$ for modified bootstrap percolation on any $\Gamma(H, Q)$ as long as H has rank r and Q does not contain periodic generators. Now, by Proposition 2.1.12 we know that modified bootstrap percolation on $\Gamma(H, Q)$ dominates k_Q-threshold bootstrap percolation on the same graph. Moreover, if we take $S = Q \cup P$ where all elements in P are periodic, we have $k_Q = k_S$. Moreover, the graph $\Gamma(H, S)$ is an extension of $\Gamma(H, Q)$. Thus β_{k_S} on $\Gamma(H_S)$ dominates β_{k_Q} on $\Gamma(H, Q)$.

Now, suppose we have some abelian group H with rank r. Next take $S \subseteq H$ to be a finite symmetric subset and set $Q \subseteq S$ to contain all non-periodic elements in S. This means that $k_Q = k_S$. Moreover, we know that β_{k_S} on $\Gamma(H, S)$ dominates β_{k_Q} on $\Gamma(H, Q)$.

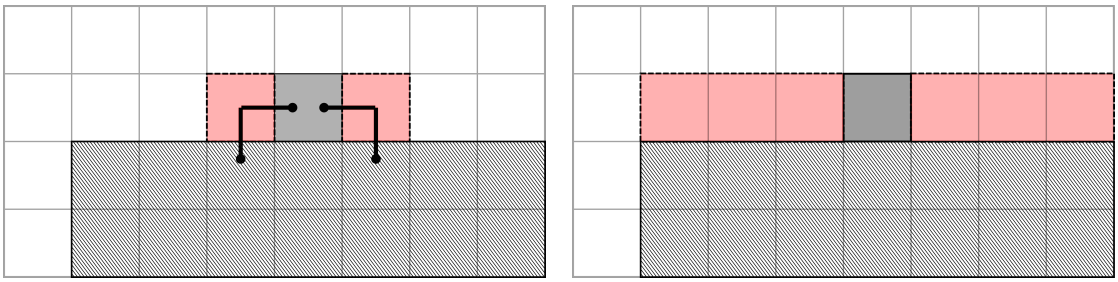
Then by the above theorem, if $p_c = 0$ for modified bootstrap percolation on all Cayley graphs $\Gamma(\mathbb{Z}^r, S')$ for finite symmetric S', then we also have $p_c = 0$ for k_Q-threshold bootstrap percolation on $\Gamma(H, Q)$. By the domination we derived above, this also means $p_c = 0$ for k_S-threshold percolation on $\Gamma(H, S)$.

Thus, to prove the case $k \leq k_S$ in Conjecture 1.1.3 for groups of rank r it suffices to only deal with modified bootstrap percolation on the free abelian group $\mathbb{Z}^r$. As such, we now focus on modified bootstrap percolation on free abelian groups.

2.2 Free abelian case

Theorem 2.1.13 shows that to prove Conjecture 1.1.3 for $k \leq k_S$ we need only consider Cayley graphs over the free abelian groups $\mathbb{Z}^n$. In this section, we get some general results for such graphs. These represent first steps towards proving we have $p_c = 0$ for modified bootstrap percolation. Sadly, later we need steps that only work in $\mathbb{Z}^2$ but in this section.

Our results here are inspired by the proofs in [Ent87] and [Sch92]. These proofs are for Cayley graphs of $\mathbb{Z}^2$ and $\mathbb{Z}^n$ using a minimal generating set, we will call these the canonical graphs. We also take some inspiration from [GG93], though we consider the proofs from that paper to be suspect. First, we sketch the ideas behind these proofs.



(2.1-a) the first step of growth **(2.1-b)** final state

Figure 2.1: How an edge grows by adding one point adjacent to the edge.

The proofs for the canonical graphs depend on rectangles or more generally, boxes. These are stable shapes at $k = n \geq 2$ (n is the dimension). That is they are sets B that satisfy:

$$B = \beta_k(B)$$

For any point outside a box has at most 1 connection to a point inside the box, and $k > 1$.

However, in some sense such boxes are only barely stable. To see this, first consider the 2D case of a rectangle of occupied points. Then, add a single occupied point along one of the edges. This activates the points along the edge adjacent to this one extra point. These added points then repeatedly activate their neighboring points until we reach the end of the edge. Once this process has finished, the edge has grown by one line. This is illustrated in Figure 2.1 The proof from [Sch92] extends this reasoning to n-dimensional boxes in their lemma 3.1. Summarized, the reasoning is as follows: consider again a box of occupied points. We now take a facet of this box and ask, "what does it take for this facet to grow". Here, by growing we mean occupying all points adjacent to this facet. This set of vacant points forms a $n-1$ dimensional box. Moreover, these points all already have 1 occupied neighbor. As such, the facet will grow when this $n-1$ dimensional box would fill itself under $(k-1)$-threshold percolation.

Our idea is to apply the proof ideas sketched above to modified bootstrap percolation. To this end, we introduce even more geometry by viewing $\mathbb{Z}^n$ as a subset of $\mathbb{R}^n$. This embedding plays a key role in this paper. It allows us to use norms (euclidean unless stated otherwise), inner-products, and discrete geometry as described in Section 2.1.1.

We then use this embedding in $\mathbb{R}^n$ to introduce a new form of bootstrap percolation on our free abelian Cayley graphs: convex bootstrap percolation. Recall that in modified bootstrap percolation, a point became occupied when a symmetric half of it's neighbors were occupied. In convex bootstrap percolation, we replace the 'symmetric half' with a 'convex half'.

Definition 2.2.1 (Convex Half)**.** *Given a finite symmetric set $S \in \mathbb{Z}^n \setminus \{0\}$ we say C is a convex half of S if:*

$$\exists u \in \mathbb{R}^n : C = S \cap H(u)$$

A single step of convex bootstrap percolation is then defined as follows:

$$\beta_S(A) = A \cup \{x \in \mathbb{Z}^n \mid \exists u \in \mathbb{R}^n : x + (S \cap H(u)) \subseteq A\} \tag{2.2.1}$$

Note that it is possible that a convex half contains more than half of all points from S. This will be resolved when we define 'minimal convex halves' (Definition 2.2.4).

Convex halves of S are all symmetric halves of S. Thus, μ_S dominates β_S. In the case of canonical generators, convex and modified bootstrap percolation coincide. For convex bootstrap percolation, like modified bootstrap percolation, a necessary condition for growth of a point x is as follows. For every $s \in S$ either $x + s$ or $x - s$ needs to be occupied.

Now, we will only use generating sets with a few reasonable properties. To prevent needlessly repeating these properties, we define a generating set as a set that has the properties we want.

Definition 2.2.2 (Generating set)**.** *We say a set $S \subseteq \mathbb{Z}^n$ is a generating set if it has all the following properties:*

- *S is finite*
- *$S = -S$ (i.e. S is symmetric)*
- *$0 \notin S$*

We call a half-space $H(u)$ stable if:

$$\beta_S(H(u)) = H(u)$$

We then ask which half-spaces are stable. In the canonical case for $\mathbb{Z}^2$ under 2-threshold bootstrap percolation, the only half-spaces that are stable are $H((1,0))$ and $H((0,1))$ and their negative counterparts. This is why rectangles are stable shapes in the canonical case. They are intersections of such (shifted) half-spaces. In arbitrary dimension, the same holds when S consists only of the canonical basis vectors and their negative counterparts. This is why, in that case, boxes are stable shapes.

We will see later that stable half-spaces are only barely stable in a sense similar to how the facets of boxes are barely stable. This is the subject of Section 2.2.1 which forms the basis of Section 2.3.

First though, we want to classify which half spaces are stable. To this end, we define V_S as the set of all stable directions:

$$V_S = \{u \in \mathbb{R}^n \mid \beta_S(H(u)) = H(u)\}$$

We then have the following alternate description for V_S.

Proposition 2.2.3. *Given a generating set $S \subseteq \mathbb{Z}^n$, we have:*

$$V_S = \{u \in \mathbb{R}^n \mid N(u) \cap S \neq \emptyset\}$$

For the proof of this proposition we need a definition we will use throughout this section:

$$\sigma_S(u) = \min\{x \cdot u \mid x \in (S \setminus H(u))\} \tag{2.2.2}$$

This is the 'shift' of a direction u. Note that by the symmetry of S we have $\sigma_S(u) = \sigma_S(-u)$. Moreover, as S is finite we know that $\sigma_S(u) > 0$.

Proof. We will prove the set equality by proving mutual inclusion.

First, we prove the inclusion $\supseteq$.
To this end, suppose we have a direction $u \in \mathbb{R}^n$ such that:

$$N(u) \cap S \neq \emptyset$$

Then, we can find a pair points: $s, -s \in N(u) \cap S$. Moreover, if C is a convex half of S then either $s \in C$ or $-s \in C$ (or both). Now, take any point $x \notin H(u)$, then by definition of u it follows that:

$$x + s \notin H(u) \ \wedge \ x - s \notin H(u)$$

So, for any convex half $C \subseteq S$ we have:

$$x + C \nsubseteq H(u)$$

Thus, we can conclude that $x \notin \beta_S(H(u)$; which means that $u \in V_S$.

Next we prove the inclusion $\subseteq$.
This will be done using contraposition, so suppose that we have a direction $u \in \mathbb{R}^n$ for which:

$$N(u) \cap S = \emptyset$$

Then note that $H(u) \cup H(-u) = \mathbb{R}^n$ and $H(u) \cap H(-u) = N(u)$. We can combine that with our assumption on u to get the following convex half C:

$$C = S \cap H(u) = S \setminus H(-u)$$

Now, take $x = \sigma_S(u)\, u$ noting that $x \notin H(u)$. Further, pick any $y \in C$ Then, by the above equation for C and the definition of σ_S we know that:

$$\begin{gathered} 0 < x \cdot u \leq -y \cdot u \\ (x+y) \cdot u \leq 0 \\ (x+y) \in H(u) \end{gathered}$$

Therefore, $x + C \subseteq H(u)$ and thus $x \in \beta_S(H(u))$ whilst $x \notin H(u)$. This means u is not a stable direction:

$$\begin{gathered} N(u) \cap S = \emptyset \implies u \notin V_S \\ V_S \subseteq \{u \in \mathbb{R}^n \mid N(u) \cap S \neq \emptyset\} \end{gathered}$$ ■

Thus, we can view V_S as the union of all orthogonal complements to some $s \in S$. In this sense, V_S has dimension $n - 1$.

Our set V_S finds another use in the following definition of minimal convex halves:
Definition 2.2.4 (Minimal convex half)**.** *Given a generating set S we call $C \subseteq S$ a minimal convex half of S if:*

$$\exists u \notin V_S : C = S \cap H(u)$$

We define the family of all minimal convex halves of S:

$$\mathcal{C}_S = \{C \subseteq S \mid \exists u \notin V_S : C = S \cap H(u)\}$$

The following proposition explains why these specific convex halves are called minimal:
Proposition 2.2.5. *Given a generating set S and any convex half $C = S \cap H(u)$, there exists a minimal convex half $M \in \mathcal{C}_S$ such that $M \subseteq C$.*

Moreover, given any minimal convex half $M \in C_S$ there exists no other convex half $C = S \cap H(u)$ such that $C \subsetneq M$.

Proof. We first prove the second claim. Let $M = S \cap H(u) \in \mathcal{C}_S$ be a minimal convex half. So $u \notin V_S$. Then, by Proposition 2.2.3 we have: $M \cap -M = S \cap N(u) = \emptyset$. Therefore, $\#M = \frac{\#S}{2}$. This means that if we leave out any point from M we have too few elements to be a convex half. So M cannot have a convex half as a proper subset.

Now, for the first claim, consider an arbitrary convex half $C = S \cap H(u)$. If $u \notin V_S$ then C is a minimal convex half and we are done. So we only need to consider the case where $u \in V_S$. In that case, we know that $S \cap N(u) \neq \emptyset$. Now set:

$$B = (S \cap H(u)) \setminus N(u)$$

We will extend B to a minimal half of S by adding a minimal convex half of $S \cap N(u)$. To this end, pick some

$$u' \in N(u) \setminus V_{S \cap N(u)}$$

Then $S \cap N(u) \cap H(u')$ is a minimal convex half of $S \cap N(u)$.

Next, as S is finite, there exists some $\epsilon > 0$ such that $S \setminus H(-u) = S \setminus H(-v)$ whenever $\|u - v\| < \epsilon$. We can then find a factor $0 < \alpha < \epsilon$ such that $v = u + \alpha\, u' \notin V_S$. It then follows that

$$S \cap H(v) = B \cup (S \cap N(u) \cap H(u')) \subseteq C$$

Thus, C contains a minimal convex half. ■

Corollary 2.2.6. *We have the following equivalent definition for convex bootstrap percolation:*

$$\beta_S(A) = A \cup \{x \in \mathbb{Z}^n \mid \exists\, C \in \mathcal{C}_S : x + C \subseteq A\} \tag{2.2.3}$$

Next, we define a finite set related to V_S that contains the 'most stable' directions:

$$U_S = \{u \mid \|u\| = 1 \wedge \text{span}(N(u) \cap S) = N(u)\}$$

That is, U_S contains all unit vectors u for which $N(u) \cap S$ spans a hyper-plane. Given a $u \in U_S$, consider the half space $H(u)$. We say this half-space is 'most stable' because if $A_0 = H(u)$ then any point $x \notin H(u)$ has vacant neighbours in a set of directions that span a full hyperplane.

Now, as S is finite, it can only span finitely many hyper-planes. Therefore, U_S is finite. Like V_S was the union of all points that are orthogonal to some $s \in S$, U_S consists of all points that are orthogonal to $n-1$ linearly independent elements from S.

Below we give a quick recap of newly introduced names. Here, we presume that S is a generating set (Definition 2.2.2).

- The set of all stable directions:
$$\begin{aligned} V_S &= \{u \in \mathbb{R}^n \mid \beta_S(H(u)) = H(u)\} \\ &= \{u \in \mathbb{R}^n \mid N(u) \cap S \neq \emptyset\} \end{aligned}$$
- The set of the most stable directions:
$$U_S = \{u \mid \|u\| = 1 \wedge \text{span}(N(u) \cap S) = N(u)\}$$
- The shift of a direction $u \in \mathbb{R}^n$:
$$\sigma_S(u) = \min\{x \cdot u \mid x \in (S \setminus H(u))\}$$
- The set of all minimal convex halves:
$$\mathcal{C}_S = \{C \subseteq S \mid \exists u \notin V_S : C = S \cap H(u)\}$$
- The convex bootstrap percolation process in terms of minimal convex halves:
$$\beta_S(A) = A \cup \{x \in \mathbb{Z}^n \mid \exists C \in \mathcal{C}_S : x + C \subseteq A\}$$
- We also introduce a new definition, the radius of S:
$$R_S = \max\{\|x\| \mid x \in S\}$$

2.2.1 Facet growth

Now that we have a nice overview of what constitutes a stable direction, we start looking at what it takes for growth to occur in such a stable direction. Specifically, we look at this in the context of facets of polytopes. Note that half-spaces are just a special case of polytopes. The idea here is that we might replaces boxes in the proof of the canonical case with polytopes.

To proceed, we want a nice way to refer to polytopes with facet normals in U_S. To this end, we take some enumeration of U_S, which yields:

$$U_S = \{u_1 \dots u_m\} \tag{2.2.4}$$

We then use this to define a matrix M_S. Considering each u_i to be a column vector we define:

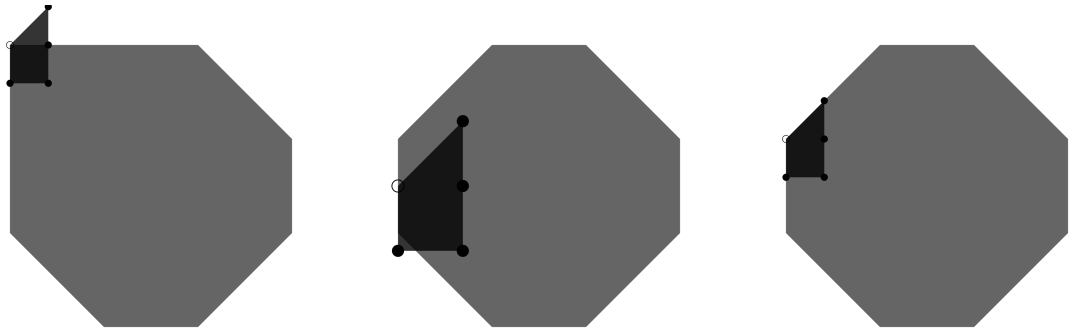
$$M_S = (\sigma(u_1)\, u_1 \mid \sigma(u_2)\, u_2 \mid \dots \mid \sigma(u_m)\, u_m)$$

Where the vertical bars are just visual separation of the columns. Our polytopes can then by defined as solutions to $M_S\, x \leq b$ for some b. Our scaling by $\sigma(u_i)$ for M_S means that integer choices for b correspond to polytopes whose bounding hyperplanes are supported by points from $\mathbb{Z}^n$. We then identify vectors b with polytopes as follows:

$$\text{Poly}_S(b) = \{x \in \mathbb{R}^n \mid M_S\, x \leq b\}$$

As these polytopes are the intersection of stable sets and β_S is increasing, these polytopes are stable as well.

Now, to generalize the proof ideas from the canonical case, we need some way of describing a 'grown' facet. We then define the growing function G and the 'facet' that gets added F. These functions take a



(2.2-a) An example of a polygon that is degenerate because it has a corner that is 'too sharp'. Thus, that vertex will never fit a convex half

(2.2-b) With this polygon, every vertex fits a convex half. However, it is degenerate due to the mid-point of the edge. The chosen minimal convex half does not fit, neither does its vertical mirror image.

(2.2-c) Finally, this polygon is non-degenerate. It is essentially a scaled-up version of the previous case.

Figure 2.2: Various polygons and whether or not they are degenerate. We take S containing all points within a 1×1 square (excluding 0). The polytopes are shown in light-gray. The minimal convex half is shown in (translucent) black. The origin of the convex half is the open circle, whilst the actual elements are shown as full circles. Note that all minimal convex halves of this S are rotated and mirrored versions of the shown minimal convex half.

polytope $P = \text{Poly}_S(b)$ and a direction $u_i \in U_S$ (we take the index of u so we can find the corresponding canonical basis vector e_i).

$$\begin{aligned} G(P, u_i) &= \text{Poly}_S(b + e_i) \\ F(P, u_i) &= G(P, u_i) \setminus P \end{aligned} \tag{2.2.5}$$

We call $F(P, u)$ the facet in direction $u \in U_S$. Note that technically, F has volume and thus is not a facet of P.

Next, we introduce the concept of degeneracy.

Definition 2.2.7 (Degenerate Polytope)**.** *Given a generating set S, we say a polytope P is degenerate (with respect to S) if*

$$\exists x \in P : \forall\, C \in \mathcal{C}_S : x + C \not\subseteq P$$

If a polytope is not degenerate, we say it is non-degenerate.

The issue with degenerate polytopes is that they contain points that cannot be activated without help from outside the polytope. The concept is illustrated in Figure 2.2.

Using the growth functions from (2.2.5) and the concept of degeneracy, we can introduce the following lemma:

Lemma 2.2.8. *Take $P = \text{Poly}_S(b)$ to be fully occupied. Take $u \in U_S$ and presume $G(P, u)$ is non-degenerate.*

Then $G(P, u)$ is internally spanned by S if $F(P, u)$ is internally spanned by S.

Proof. Our proof is based on the decomposition $G(P, u) = P \cup F(P, u)$. We set $A \subseteq \mathbb{Z}^d$ to be the set of active points and then define $\bar{A} = A \cap F(P, u)$, which is the set of active points inside the facet.

Now, suppose we have a point $x \in F(P, u)$ that gets activated by the dynamics using generating set $S \cap N(u)$. That is,

$$x \in \beta_{S \cap N(u)}(\bar{A})$$

If we then show that this implies $x \in \beta_S(A)$ the proof can be finished with a trivial inductive argument, as $F(P, u)$ was internally spanned.

If $x \in \bar{A}$ it follows immediately that $x \in \beta_S(A)$. So we proceed under the assumption that $x \notin \bar{A}$. Then, we need to find some convex half $C \in \mathcal{C}_S$ such that $x + C \subseteq A$.

Now, by definition of x, we know there exists some minimal convex half $\bar{C} \in \mathcal{C}_{S \cap N(u)}$ such that $x + \bar{C} \subseteq \bar{A}$. Moreover, as $G(P, u)$ is non-degenerate, there exists some minimal convex half $D \subseteq S$ such that $x + D \subseteq G(P, u)$. Now, we take:

$$C = (D \setminus N(u)) \cup \bar{C}$$

Now, by definition $x + \bar{C} \subseteq \bar{A} \subseteq A$. Moreover, by $x \in F(P, u)$ and the assumption that $P \subseteq A$ we get:

$$x + (D \setminus N(u)) \subseteq P \subseteq A$$

Thus indeed we have $x + C \subseteq A$. All that remains is to show that C is indeed a convex half of S. To see this we set $\bar{u}$ to be the direction that creates $\bar{C}$ and u to be the direction that creates D. That is, pick $\bar{u} \in N(u)$ such that $\bar{C} = S \cap N(u) \cap H(\bar{u})$ and $u \in \mathbb{R}^n \setminus V_S$ such that $D = S \cap H(u)$.

Next, as S is finite, there exists some $\epsilon > 0$ such that $S \setminus H(-u) = S \setminus H(-v)$ whenever $\|u - v\| < \epsilon$. We can then find a factor $0 < \alpha < \epsilon$ such that $v = u + \alpha\, \bar{u} \notin V_S$. It then follows that

$$S \cap H(v) = (D \setminus N(u)) \cup \bar{C} = C$$

Thus, C contains a minimal convex half. ■

This lemma is very useful, but it depends on non-degenerate polytopes. Thus, we are interested in determining whether a polytope is non-degenerate. To help determine this, we introduce some new functions. Given a polytope P, let $\mathcal{F}$ be its face lattice. We then define the following set given a face $F \in \mathcal{F}$:

$$\mathcal{T}(F) = \{G \in \mathcal{F} \mid F \subseteq G \wedge \dim G = n - 1\}$$

This is the family of all facets that contain the face F. Then given a point in a face $x \in F \in \mathcal{F}$, we define the margin of that point in that face as:

$$\epsilon_F(x) = \inf \left\{ \|x - y\| \;\middle|\; y \in \bigcup \left(\mathcal{F} \setminus \left(\{P\} \cup \mathcal{T}(F)\right)\right) \right\}$$

This captures how far x lies from the nearest facet not in $\mathcal{T}(F)$. Then, let $O : \mathcal{F} \to \mathbb{R}^n$ be a function that selects a point from every face. That is, $O(F) \in F$. Associated with this selection O is the margin:

$$\epsilon_O = \min\{\epsilon_F(O(F)) \mid F \in \mathcal{F} \wedge \dim F > 0\}$$

Note that, if we pick every $O(F)$ to lie in the interior of F we have $\epsilon_O > 0$. Moreover, we can say that ϵ_O scales linearly with P. To be precise, if we scale P by factor α and define the scaled selection function $\alpha\, O$ as $F \to \alpha O(F)$ then we have:

$$\epsilon_{\alpha\, O} = \alpha \epsilon_O$$

Using these definitions we then introduce the following theorem. Note that this theorem has no requirements on the face normals of our polytope. That is, it works for all polytopes, not just those of the form $\text{Poly}_S(b)$.

Theorem 2.2.9. *Let $S \subseteq \mathbb{Z}^n$ be a generating set and let $P \subseteq \mathbb{R}^n$ be a convex polytope. Set V to be the vertices of P.*

Now assume there is a function $C : V \to \mathcal{C}_S$ such that $v + C(v) \subseteq P$. Moreover, suppose we have a selection of points from facets O such that $\epsilon_O > R_S$.

Then P is non-degenerate with respect to S.

To prove this, we first introduce a lemma. The proof of this lemma relies heavily on the face lattice as defined in Section 2.1.1.

Lemma 2.2.10. *Let P be a convex polytope with vertices V and face lattice $\mathcal{F}$. Now, assume there is a function $C : V \to C_S$ such that $v + C(v) \subseteq P$.*

Then, for any $F \in \mathcal{F}$ and $x \in F$ we have:

$$\epsilon_F(x) > R_S \implies \forall v \in F \cap V : C(v) + x \subseteq P$$

That is, if the margin of a point x in a facet F is large enough, the convex halves that fit at vertices of F fit at x.

Proof. First, given any face $F \in \mathcal{F}$ except $F = P$ we define $u(F)$ and $b(F)$ as the direction vector and shift vector that create F. That is:

$$F = (N(u(F)) + b(F)) \cap P$$

Using this description, we define the 'cone' of a face:

$$T(F) = \bigcap_{G \in \mathcal{T}(F)} H(u(G))$$

This is the smallest cone that 'fits over' a face. Specifically, given any $b \in F$ we have $b + T(F) \supseteq P$. If F is a facet, $T(F)$ is just $H(F)$. In general, for a face of dimension $n - k$ $T(F)$ is an intersection of $k - 1$ half-spaces. For example, if F is an edge in $\mathbb{R}^3$ then $T(F)$ is an intersection of two half spaces that hug the two facets on either side of the edge.

So, if we take a vertex $v \in F \cap V$ we have:

$$v + C(v) \subseteq P \subseteq v + T(F)$$

From this it follows that for any $v \in F \cap V$ we have $C(v) \subseteq T(F)$.

Now we fix a face $F \in \mathcal{F}$, and pick a point $x \in F$ such that $\epsilon_F(x) > R_S$. Then, we pick a vertex $v \in F$. We will then show by contradiction that $x + C(v) \subseteq P$.

Suppose there exists a $a \in C(v)$ such that $y = x + a \notin P$. Then, draw a line between x and y. As $x \in P$ and $y \notin P$ there must exist a $z \in \text{Conv}(x, y)$ that lies on the border of P. Since the border of P is the union of all facets of P, there must then be some facet G such that $z \in G$. Formally, we take

$$z \in \text{Conv}(x, y) \cap \bigcup (\mathcal{F} \setminus \{P\})$$
$$G \in \{F \in \mathcal{F} \mid z \in F \wedge \dim(F) = n - 1\}$$

Now, we know $G \notin \mathcal{T}(F)$ because $a \in C(v) \subseteq T(F)$ and so $x + a \in x + T(F)$. But this leads to a contradiction:

$$\|x - z\| \geq \epsilon_F(x) > R_S \qquad \text{(By } G \notin \mathcal{T}(F) \text{ and choice of } x\text{)}$$
$$\|x - z\| \leq \|x - y\| = \|a\| \leq R_S \qquad \text{(by } a \in C(v) \subseteq S\text{)}$$

■

Now, with this lemma proven we can proceed to prove Theorem 2.2.9. This proof will use a construction similar to z and G used above.

Proof of Theorem 2.2.9. By the assumption $\epsilon_O > R_S$ and Lemma 2.2.10 we have:

$$\forall v \in F \cap V : O(F) + C(v) \subseteq P \tag{2.2.6}$$

Now, we fix some $x \in P$. We will then produce some v such that $C(v) + x \in P$. Or equivalently, some v such that $x \in P \ominus C(v)$. Note that as P is convex, $P \ominus C(v)$ is also convex.

Now, we define 2 sequences recurrently. $[X_i]$ will be a sequence of points and $[F_i]$ will be a sequence of facets containing X_i. The dimension of $[F_i]$ will be decreasing.

$$X_0 = x, \quad X_{i+1} = L(O(F_i), X_i) \cap \bigcup \{G \in \mathcal{F} \mid G \subseteq F_i\}$$
$$F_0 = P, \quad F_{i+1} \in \max\{G \in \mathcal{F} \mid G \subseteq F_i \wedge X_{i+1} \in G\}$$

Here $L(a, b)$ is a half line through b originating at a. That is, $L(a, b) = \{a + \alpha(b - a) \mid \alpha \in [0, \infty)\}$. So at every step, we take X_i and project it from $O(F_i)$ onto the border of F_i to get X_{i+1}. We then set F_{i+1} equal to a facet of F_i in which X_{i+1} lies.

Note that the sequences stop when F_i is a vertex because then X_{i+1} is no longer defined. This happens at $i = n$ because $\dim(F_i) = n - i$. Also, note that the definition of X_{i+1} is abuse of notation. Technically, the right hand side is a singleton set, not an element from $\mathbb{R}^n$. That it is a singleton set follows from $\epsilon_0 > 0$.

Next, we define a derived sequence of convex hulls $[D_i]$:

$$D_i = \text{Conv}(\{O(F_0) \dots O(F_i), X_i\})$$

And claim the following invariant:

$$j \leq i \implies D_j \subseteq D_i$$

We will prove this by induction. The base case of $i = 0$ is vacuously true. Now presume that for some k we have $j \leq k \implies D_j \in D_k$. Then, from the definition of X_{k+1} we have

$$X_{k+1} \in \text{Conv}(O(F_k), X_k) \subseteq D_{k+1}$$

By definition of D_{k+1} we have $O(F_j) \subseteq D_{k+1}$ for all $j \leq k$ so we can conclude that $D_k \subseteq D_{k+1}$. This finishes the inductive proof of the invariant.

Since $X_0 \in D_0$ the invariant gives us:

$$x \in D_n$$

We know that the face F_n is a vertex. Thus, we set $v = X_n \in V$ and can conclude that: $F_n = \{v\}$ and $O(F_n) = v$. Thus we have:

$$D_n = \text{Conv}(O(F_1) \dots O(F_n))$$

Moreover, from $F_{i+1} \subseteq F_i$ and then applying Equation 2.2.6 we get:

$$\forall i \leq n : v \in F_i$$
$$\forall i \leq n : O_i + C(v) \subseteq P$$

By the previous two equations, $x \in D_n$ and the convexity of P we thus have:

$$x + C(v) \subseteq D_n \oplus C(v) \subseteq P$$ ■

Next, given any generating set S we will construct a polytope P_S that allows a function $C(v)$ as above. We could then scale this polytope sufficiently to get a non-degenerate polytope. This polytope is defined as:

$$P_S = \text{Conv}\left(\sum \{C \cup \{0\} \mid C \in \mathcal{C}_S\}\right) \tag{2.2.7}$$

That is, we take P_S to be the Minkowski sum of all minimal convex halves of $S \cup \{0\}$. For this, we have the following lemma:

Lemma 2.2.11. *For every vertex v of P_S there exists some convex half $C(v) \in \mathcal{C}_S$ such that*

$$v + C(v) \subseteq P_S$$

Proof. Take v to be any vertex of P_S. Then there exists some u_v such that $P_S \cap (N(u_v) + v) = \{v\}$. This is also the unique solution to:

$$v = \arg\max_{x \in P_S} x \cdot u_v$$

From the definition of P_S we can write the above arg max as a sum of unique arg max expressions:

$$v = \sum \left\{ \arg\max_{x \in C \cup \{0\}} x \cdot u_v \;\middle|\; C \in \mathcal{C}_S \right\}$$

From this, it follows that $u_v \notin V_S$. For otherwise, one of the elements of the above sum would not be unique. Now, we set

$$C(v) = H(u_v) \cap S$$

We just showed $u_v \notin V_S$ so it is given that $C(v) \in \mathcal{C}_S$. That is $C(v)$ is a minimal convex half. We then need to show that $v + C(v) \subseteq P_S$. To this end, define:

$$Q = \sum \big\{ C \cup \{0\} \mid C \in \mathcal{C}_S \setminus \{C(v)\} \big\}$$

That is, we take the Minkowski sum that defines P_S, but leave out the minimal convex half $C(v)$. It follows that $Q \oplus C(v) \subseteq P_S$. Thus, it suffices to show that $v \in Q$. By definition of $C(v)$ we have

$$\underset{x \in C(v) \cup \{0\}}{\arg\max} \; x \cdot u_v = 0$$

Thus, we can leave out the component from $C(v)$ in the sum that defines v. This yields:

$$v = \sum \left\{ \underset{x \in C \cup \{0\}}{\arg\max} \, x \cdot u_h \;\middle|\; C \in \mathcal{C}_S \setminus \{C(v)\} \right\} = \underset{x \in Q}{\arg\max} \, x \cdot u_v$$

Thus, it follows that $v \subseteq Q$. ■

Interestingly, when scaled up sufficiently Theorem 2.2.9 implies this polytope forms a $(\frac{\#S}{2}+1)$-fort. This allows us to prove the case $k > k_S$ in Conjecture 1.1.3.

Theorem 2.2.12. *Let H be a finitely generated abelian group of rank r and let $S \subseteq H$ be a finite symmetric set with $0 \notin S$. We set k_S as follows:*

$$k_S = \frac{\#\{s \in S \mid \forall n \in \mathbb{N}^+ : n \times s \neq 0\}}{2}$$

Then, if $k > k_S$ we have $\pi_c = p_c = 1$ for β_k on $\Gamma(H, S)$.

Proof. Since $\pi_c \geq p_c$ it suffices to show that $p_c = 1$ for $k > k_S$. Now from Lemma 2.1.4 we know that $p_c = 1$ for β_k if there exist finite k-forts. So it suffices to produce a finite k-fort (where $k > k_S$).

As H has rank r, there exists a surjective group homomorphism $\phi : H \mapsto \mathbb{Z}^r$. We use this to set $Q = \phi(S) \setminus \{0\}$ and consider $P_Q \subseteq \mathbb{Z}^r$ as defined in Equation 2.2.7. Next we pick $n \in \mathbb{N}$ large enough that Theorem 2.2.9 ensures nP_Q is non-degenerate. We will then show that the set $F = \phi^{-1}(nP_Q) \subseteq H$ is a k-fort.

As nP_Q is non-degenerate we know that for every element $y \in nP_Q$ there exists a convex half $C(y) \in \mathcal{C}_Q$ such that $y + C(y) \subseteq nP_Q$. Now consider any point $x \in F$. From the above we can conclude that:

$$\phi(x) + \big(\{0\} \cup C(\phi(x))\big) \subseteq nP_Q \tag{2.2.8}$$

In order to make use of the above, we define a new set $D_x \subseteq S$:

$$D_x = S \cap \phi^{-1}\big(\{0\} \cup C(\phi(x))\big)$$

Then, based on (2.2.8) we can conclude that:

$$x + D_x \subseteq F$$

As we wanted a k-fort, we want an upper bound on $\#(S \setminus D_x)$. Since $C(\phi(x))$ is a convex half of Q, we know that D_x contains a symmetric half of S. (See Definition 2.1.11 for the meaning of a symmetric half.) Moreover, $\phi^{-1}(0)$ contains all periodic elements in H. Thus D_x also contains all periodic generators in S. As such, we know that:

$$\#(S \setminus D_x) \leq k_S < k$$

■

2.2.2 An upper bound on π_c for β_S

For Cayley graphs on $\mathbb{Z}^r$, we can actually use a so called 'Peierls estimate' to prove an upper bound on π_c for β_S. Through domination, this also holds for modified and k-threshold bootstrap percolation. We make the argument for convex bootstrap percolation, but through domination it also holds for modified- and k-threshold bootstrap percolation. The methods of the proof are somewhat disjoint from the rest of this paper.

Later, we will combine this theorem with a process called renormalization to prove that $\pi_c = 0$ when working in $\mathbb{Z}^2$.

Theorem 2.2.13. *For any generating set $S \in \mathbb{Z}^n$ using β_S as our bootstrap process, we have:*

$$\pi_c \leq 1 - \frac{1}{\#S - 1}$$

Proof. First, we need to construct a norm p_V on $\mathbb{Z}^n$. To this end, we define $\sigma'(x)$ as the minimal positive inner product of x with an element from $\mathbb{Z}^n$:

$$\sigma'(x) = \inf\{x \cdot y \mid y \in \mathbb{Z}^n \setminus H(x)\}$$

For $x \in \mathbb{Z}^n$ it follows that $\sigma(x) > 0$. Then, we pick n linearly independent vectors $v_i \dots v_n \notin V_S$. We then define the set:

$$V = \{\sigma'(v_0)v_0, \dots, \sigma'(v_n)v_n\}$$

As $v_i \in \mathbb{Z}^n$ we know that $\sigma(v_i) > 0$ so $0 \notin V$. We can then define the norm p_V and the corresponding radius r_V as follows:

$$p_V(x) = \max_{v \in V} |v \cdot x|$$
$$r_V(A) = \max_{a \in A} p_V(a)$$

Note that due to our scaling by σ' for $x \in \mathbb{Z}^n$ we always have $p_V(x) \in \mathbb{N}$. Moreover, as all elements from V are linearly independent, we know that $p_V(x) = 0$ implies $x = 0$.

Now, take A to be the cluster of vacant vertices connected to the origin in some configuration $B \subseteq \mathbb{Z}^n$. Moreover set A' to be the same cluster in the configuration $\beta_S(B)$. That is, A' is what remains of A after a single step of convex bootstrap percolation.

Then we claim our choice of V gives the following:

$$r_V(A') \leq r_V(A) - 1 \tag{2.2.9}$$

If A is infinite, this is trivially true. To see this for finite A, pick $x \in A$ and $v_x \in V$ such that $|v_x \cdot x| = r_V(A)$. As A is the connected cluster of vacant points it follows that all points in

$$x + (S \setminus H(v_x))$$

are occupied. After all, any vacant point y in the above set would be part of the connected cluster, and have $p_V(y) > p_V(x) = r_V(A)$. Moreover, as $v_x \notin V_S$ it follows that

$$S \setminus H(v_x) \in \mathcal{C}_S$$

Therefore, by definition of convex bootstrap percolation, $x \notin A'$. This shows that $r_V(A') < r_V(A)$ because every point $x \in A$ with $p_V(x) = r_V(A)$ becomes occupied. The full from of (2.2.9) follows from $r_V(A) \in \mathbb{N}$.

Now, from Equation 2.2.9 we can conclude that:

$$r_V(A) \geq T \tag{2.2.10}$$

(T is the stopping time as defined in equation 1.0.2). Thus, an exponential bound for $r_V(A)$ gives an exponential bound for T. We can get such a bound via the radius of A in terms of the graph-distance from the origin.

Specifically, let $l(x)$ be the length of the shortest path from x to the origin. And let $L(A)$ be the radius of A with respect to $l(x)$. That is:

$$L(A) = \max\{l(x) \mid x \in A\}$$

Now, we want a relation between $L(A)$ and $r_V(A)$. To this end we set:

$$\alpha = \max\left\{|v \cdot s| \mid v \in V, s \in S\right\}$$

Now, let $b_1 \dots b_{l(x)} \in S$ form such a shortest path to x. That is, $x = \sum_{i=1}^{l(x)} b_i$. We then get the bound:

$$\begin{aligned} \alpha\, l(x) &\geq \max_{v \in V} \sum_{i=1}^{l(x)} |v \cdot b_i| \\ &\geq \max_{v \in V} \left| v \cdot \sum_{i=1}^{l(x)} b_i \right| = p_V(x) \end{aligned}$$

From the bound above, the definition of $L(A)$ and $r_V(A)$ and Equation 2.2.10 we then get:

$$\alpha L(A) \geq r_V(A) \geq T$$

Writing the above bound in terms of probabilities, we get:

$$\mathbb{P}_p(T > t) \leq \mathbb{P}_p\left(L(A) > \frac{t}{\alpha}\right)$$

We can then use a Peierls-estimate to get an exponential bound on $L(A)$. By definition of $L(A)$ if we have $L(A) > t$ then there is at least one non-backtracking path of length larger than t. Now, the number of non-backtracking paths with length l is at most $\#S(\#S-1)^{l-1}$. Thus, we get:

$$\mathbb{P}_p(L(A) \geq t) \leq \sum_{l=t}^{\infty} (1-p)^{l+1} \#S(\#S-1)^{l-1} \leq Ce^{-\gamma t} \tag{2.2.11}$$

Now, presume:

$$p > 1 - \frac{1}{\#S - 1}$$

In that case, in Equation 2.2.11 we have $C < \infty$ and $\gamma > 0$. Thus, we have:

$$\mathbb{P}_p(T > t) \leq \mathbb{P}_p\left(L(A) > \frac{t}{\alpha}\right) \leq C \exp\left(-\frac{\gamma}{\alpha} t\right)$$

Which means we have $\gamma(p) \geq \gamma/\alpha > 0$ (with $\gamma(p)$ as defined in Equation 1.0.4). From this we can conclude that:

$$\pi_c \leq 1 - \frac{1}{\#S - 1}$$

■

2.3 The seed argument in $\mathbb{Z}^2$

Next, we use the general results from Section 2.2 to get stronger results in the specific case of $\mathbb{Z}^2$. The 2D case is more easier than the general case. This is mostly due to the ease of describing polygons and their facets (that is, their edges). In this section, we first introduce the 'seed' argument, and use it to show we have $p_c = 0$. In the next section, we will combine results from this section with a process called renormalization and the upper bound on π_c from Theorem 2.2.13 to show that, in fact we also have $\pi_c = 0$. (p_c and π_c are defined in (1.0.3) and (1.0.5).)

Our first step is to make non-degeneracy of polygons easy to determine. Given a polygon $P = \text{Poly}_S(b)$ let $E(P)$ be the shortest edge of P. Formally:

$$E(P) = \min\{\|v_1 - v_2\| \mid \exists u \in U_S : \text{Conv}(v_1, v_2) = P \cap (v_1 + H(u))\}$$

If $v_1 \neq v_2$ in the above expression then (v_1, v_2) is an edge of P orthogonal to $u \in U_S$. If $v_1 = v_2$, it means that $v_1 = v_2$ is a vertex of P and there is no edge normal to $u \in U_S$. Thus $E(P) = 0$ if and only if there is some $u \in U_S$ such that P does not have any edge normal to u.

Next, we call a polygon obtuse if all internal angles are greater than $90°$. If $E(P) > 0$ we can always find a linear (skewing) transform f such that $f(P)$ is obtuse. This follows from $\#U_S \geq 4$ and the symmetry of U_S. We can then get the following result for degeneracy:

Lemma 2.3.1. *Let $P = \text{Poly}_S(b) \subseteq \mathbb{R}^2$ be obtuse. Then P is non-degenerate if $E(P) > 2R_S$.*

Proof. We will prove this using Theorem 2.2.9. To do this, need to prove two things:

- For every vertex $v \in P$, there exists a convex half $C(v)$ such that $v + C(v) \subseteq P$.
- We can find a function O, which selects an element for every face of F such that $\epsilon_O > R_S$.

For the first requirement, given any vertex v there exists a $u_v \notin V_S$ such that $v + N(u_v) \cap P = \{v\}$. We then take the convex half:

$$C(v) = S \cap H(u_v)$$

Then, let $u_l, u_r \in U_S$ be the closest elements to the left and right of u_v. This gives us

$$\text{Coni}(C(v)) = H(u_l) \cup H(u_r)$$

Moreover, u_l and u_r are the edge normals of the edges incident on v. If we combine that with $E(P) > 2R_S > R_S$ we get:

$$v + (B(R_S) \cap H(u_l) \cap H(u_r) \subseteq P \tag{2.3.1}$$

Where, $B(r) = \{x \in \mathbb{R}^2 \mid \|x\| \leq r\}$ is the ball of radius r. From the conical sum expression above we get:

$$\begin{aligned} C(v) &= S \quad \cap \text{Coni}(C(v)) \\ &= S \quad \cap H(u_l) \cap H(u_r) \\ &\subseteq B(R_S) \cap H(u_l) \cap H(u_r) \end{aligned}$$

Combining the above equation with (2.3.1) gives us the first requirement.

Now, for the second claim, let $\mathcal{F}$ be the face lattice of P. Then $\mathcal{F}$ consists of P, all edges of P and all vertices of P. We will explicitly construct our selection function O.

- For $O(P)$ let (v_1, v_2) be an edge with length $E(P)$. Then, pick $O(P) \in P$ such that $\{v_1, v_2, O(P)\}$ forms an isosceles right triangle, with the right angle at $O(P)$. As P is obtuse, and all edges are longer than $2R_S$, no edge is closer to $O(P)$ than (v_1, v_2). The distance to that edge is $\sqrt{2}R_S > R_S$. So $\epsilon_P(O(P)) > R_S$.
- For $O(F)$ where F is an edge set $O(F)$ to be the middle of that edge. Then $\epsilon_F(O(F)) > R_S$. Thus, we have constructed a selection O such that $\epsilon_O(P) > R_S$.
- For $O(\{v\})$ where v is a vertex, we have no option but to set $O(\{v\}) = v$.

It then follows that $\epsilon_O(P) > R_S$. ■

With such an easy description of non-degeneracy, we look towards making use of Lemma 2.2.8. In 2D, our growth sets $F(P, u)$ are 1-dimensional, as is the corresponding generating set. Specifically, our generating set is $S \cap N(u)$. We then take $g(u)$ to be the smallest element in $S \cap N(u)$ (which is unique up to sign). Moreover we also define $m(u)$ as the 'radius' of $S \cap N(u)$. Formally we set:

$$\begin{aligned} m(u) &= \frac{\#S \cap N(u)}{2} \\ g(u) &= \underset{x \in S \cap N(u)}{\arg\min} \|x\| \end{aligned}$$

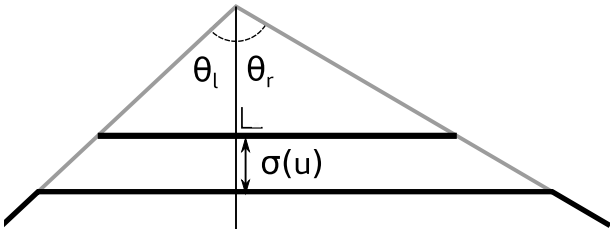


Figure 2.3: Illustration of l_u. It is the difference in length between the two horizontal lines.

We can then write one convex half of $S \cap N(u)$ as:

$$C_u = \{1g(u), 2g(u), \ldots, m(u)g(u)\}$$

The entire generating set can then be expressed as:

$$S \cap N(u) = C_u \cup -C_u$$

This generating set only has the two convex halves C_u and $-C_u$. Thus, $F(P, u)$ is internally spanned if and only if it contains $m(u)$ consecutive occupied points.

We can then get a lower bound on the probability that $F(P, u)$ is internally spanned by $S \cap N(u)$. The question of a set being internally spanned will come up more often. Hence we introduce the following definition:

$$R_S(P, p) = \mathbb{P}_p(P \text{ is internally spanned})$$

Our lower bound on $R_{S\cap N(u)}(F(P, u), p)$ will be based on $E(P)$, the length of the shortest edge of P. Consider any edge $F(P, u)$. This edge contains

$$\left\lfloor \frac{\#F(P, u)}{m(u)} \right\rfloor$$

non-overlapping intervals of length $m(u)$. If one of these intervals is fully occupied, $F(P, u)$ is internally spanned. Given such an interval, the probability that it is fully occupied is $(1-p^{m}(u))$. As these intervals do not overlap, the events of them being fully occupied are independent. Using $\lfloor a/b \rfloor > a/b - 1$, we then get the following lower bound:

$$R_{S\cap N(u)}(F(P, u), p) \geq 1 - (1 - p^{m(u)})^{\frac{\#F(P,u)}{m(u)}-1} \tag{2.3.2}$$

We have an estimate here because we did not consider overlapping intervals[1]. However, what is important here is the exponential bound in terms of $\#F(P, u)$. Next, we set:

$$C_e(S) = \min\{\|g(u)\| \mid u \in U_S\}$$

We can then use this to get a relation between $\#F(P, u)$ and $E(P)$:

$$\#F(P, u) > C_e(S)\,(E(P) - 1) \tag{2.3.3}$$

Next, we consider $E(G(P, u))$. If P is obtuse, when we compare the edges of $G(P, u)$ to those of P only the edge orthogonal to u shrinks. All other edges either grow or are unchanged. Presuming the edge orthogonal to u does not disappear this edge shrinks by a constant amount, we call this amount l_u. We set θ_l to be the angle between the current edge and the left incident edge, and similarly set θ_r for the right incident edge. This situation is sketched in Figure 2.3. We then get:

$$l_u = \sigma(u)(\tan(\theta_l) + \tan(\theta_r))$$

We can thus say:

$$E(G(P, u)) > E(P) - l_u \tag{2.3.4}$$

[1]See stack exchange for an exact formulation: math.stackexchange.com/a/59749

On the other hand, for any $\alpha \in [0, \infty)$ we have

$$E(\text{Poly}_S(\alpha b)) = \alpha E(\text{Poly}_S(b) \tag{2.3.5}$$

The equation above, (2.3.5) is close to what we need. Specifically, we need edges to grow as we grow the polytope. However, the second to last equation, (2.3.4), causes some difficulties as it states that some intermediary steps might have smaller edges. Thus there is some tension between the two equations. In general though, we can interpret (2.3.4) as stating there is no monotonic growth in edge lengths on the short term. Whilst (2.3.5) states that in the long term the edge lengths do grow larger. In the canonical case, treated in [Sch92], we have $l_u = 0$ which simplifies the arguments a lot. Not having that luxury, we will proceed to formalize the idea that edge lengths grow larger in the long term.

To reconcile the two formulas, we take a 'seed' of the form $\text{Poly}_S(kb)$ and then describe how to grow that seed to something of the form $\text{Poly}_S\big((k+1)b\big)$. For the intermediate stages of this growing, we can only use (2.3.4), but for the final result we can use (2.3.5). We can then iteratively describe growing $\text{Poly}_S(kb)$ to any $\text{Poly}_S\big((k+n)b\big)$.

Now, we fix some b such that $E(\text{Poly}_S(b)) > 0$. We then get a sequence $v_1 \dots v_J \in U_S$ where that grows $\text{Poly}_S(kb)$ to $\text{Poly}_S\big((k+1)b\big)$. We set $J = (1, \dots, 1) \cdot b$. That is, J is the element-wise sum of b. To formally define what it means for a sequence $[v_i] \in [U_S]$ to grow a polytope, we define two related sequences:

$$\begin{aligned} G'_0(P) &= P \\ G'_{i+1}(P) &= G(G_i(P), v_i) \\ F'_{i+1}(P) &= F(G_i(P), v_{i+1}) \end{aligned} \tag{2.3.6}$$

Now, given a sequence $[v_i] \in [U_S]$, we define a corresponding sequence $[n_i] \in [\mathbb{N}]$ such that $v_i = u_{n_i}$ where we reuse the enumeration of U_S from Equation 2.2.4. Then, it suffices to find a sequence $[v_i]$ such that:

$$b = \sum_{i=1}^{J} e_{n_i}$$

where e_i are the canonical basis vectors. It then follows from the definition of $G(P, u)$ (Equation 2.2.5) that:

$$G'_J(\text{Poly}_S(kb)) = \text{Poly}_S\big((k+1)b\big)$$

Note that we can reorder the sequence $[v_i]$ and still get the same end result.

We then set $L = J \times (\max l_u)$ and give the very crude bound:

$$E(G'_i(P)) \leq E(P) - L \tag{2.3.7}$$

Note that this bound cannot possibly be tight, as $E(G'_J(P)) > E(P)$ by Equation (2.3.5). An actual tight bound would depend on the ordering of v_i.

Now, pick k sufficiently high such that $P = \text{Poly}_S(kb)$ has $E(P) > 2R_S + L$. Then by Lemma 2.3.1 all of $G'_0(P_k) \dots G'_J(P_k)$ are guaranteed to be non-degenerate. This P shall be our 'seed'. We then set the expanded polytope $P' = \text{Poly}_S\big((k+1)b\big)$. We then consider the question: "what is the probability that a fully occupied P grows to P'". Here, we consider the set $P' \setminus P$ to be the 'shell' of P. Now, this shell is partitioned by $F'_i(P)$. Moreover, by Lemma 2.2.8 and an induction argument if all $F'_i(P)$ are internally spanned, then P grows to P'.

First, we get a bound on $F'_i(P)$ being internally spanned: For this, we set

$$m = \max_{u \in U_S} m(u)$$

and then get:

$$\begin{aligned}
&R_{S\cap N(u)}(F_i'(P),p) && \geq && \text{(by (2.3.2))}\\
&1-(1-p^m)^{\frac{\#F(P,u)}{m(u)}-1} && = &&\\
&1-\exp\left(\ln(1-p^m)\left(\frac{\#F(P,u)}{m(u)}-1\right)\right) && \geq && \text{(by (2.3.4))}\\
&1-\exp\left(\ln(1-p^m)\left(\frac{C_e(S)}{m}(E(G_i'(P))-1)-1\right)\right) && \geq && \text{(by (2.3.7))}\\
&1-\exp\left(\ln(1-p^m)\frac{C_e(S)}{m}\left(E(P)-L-1-\frac{m}{C_e(S)}\right)\right) = 1-e^{\gamma(E(P)-K)} &&&&
\end{aligned}$$

where $\gamma=\ln(1-p^m)\frac{C_e(S)}{m}$ and $K=L+1+\frac{m}{C_e(S)}$.

Now, we can use the above to get a lower bound on P growing to P':

$$\begin{aligned}
&\mathbb{P}_p(P' \text{ internally spanned} \mid P \text{ internally spanned}) \geq\\
&\quad \mathbb{P}_p(\forall i\in(1\dots J): F_i'(P) \text{ internally spanned}) =\\
&\qquad \prod_{i=1}^{J} R_{S\cap N(v_i)}(F_i'(P),p) \geq \left(1-e^{\gamma(E(P)-K)}\right)^J
\end{aligned} \tag{2.3.8}$$

Using this approximation we have an easy proof of percolation in 2 dimensions.

Theorem 2.3.2. *Let $S\subseteq\mathbb{Z}^2$ be a generating set. Then β_S has critical probability $p_c=0$.*

Proof. By the 0-1 law (Theorem 2.1.3) all we need to show is that for any initial density $p>0$ we have positive probability of percolation.

Like in previous proofs, we take $P=\text{Poly}_S(kb)$ such that $E(P)>2R_S+L$. As P is finite the probability that P is internally spanned is positive. Specifically we have the crude lower bound:

$$\mathbb{P}_p(P \text{ internally spanned}) > p^{\#P} > 0$$

Now, we are guaranteed percolation if as $i\to\infty$ the polytopes $\text{Poly}_S(ib)$ remain internally spanned. By Equation 2.3.8 and $E(\text{Poly}_S(ib))=iE(P)$ we then get the following estimate:

$$\begin{aligned}
&\mathbb{P}_p(\text{percolation} \mid P \text{ internally spanned}) && \geq\\
&\mathbb{P}_p(\forall i>k: \text{Poly}_S(ib) \text{ internally spanned} \mid P \text{ internally spanned}) && \geq\\
&\prod_{i=k}^{\infty}\left(1-\exp(\gamma(iE(P)-K)\right)^J && > 0
\end{aligned} \tag{2.3.9}$$

■

The core of any seed argument lies in Equation 2.3.8. Specifically, the exponential form it gives for the probability that a shell around a seed grows. We can then use that iteratively to bound the probability of any seed growing to multiple shells. However, seed arguments vary in the estimate of an initial seed existing. For the above result, we used finiteness of the shell to show the probability was merely positive. For the lemma below, we will present a proof using ergodicity to estimate the probability of a seed being internally spanned.

This next lemma is a first step towards using Theorem 2.2.13. We will later combine it with renormalization to show we have exponential percolation for any positive initial density.

Lemma 2.3.3. *Working in $\mathbb{Z}^2$ given a vector $b\in\mathbb{R}^m$ such that $E(\text{Poly}_S(b))>0$ and $\text{Poly}_S(b)$ is symmetric we have:*

$$\lim_{k\to\infty} R_S(\text{Poly}_S(kb),p)=1$$

for any $p>0$.

Proof. We take the finite sequences $[v_i], [G_i'], [F_i']$ from Equation 2.3.6 and extend them to infinity. This is done by making the sequence $[v_i]$ repeat itself. That is, we set:

$$v_i = v_{i \bmod J}$$

We then set n to be the smallest number such that $nE(\text{Poly}_S(b)) > 2R_S + L$. We then define the smallest possible seed $T = \text{Poly}_S(nb)$.

We then set our outer polygon $P = \text{Poly}_S(kb)$ where k is sufficiently high to have:

$$m = \left\lfloor \frac{k-n}{3} \right\rfloor > n$$

and use this m to define the inner polygon:

$$Q = \text{Poly}_S(mb)$$

This polygon will be our seed. Based on our seed we then define 2 events:

- Q' is the event that Q gets filled by the dynamics restricted to P. That is, Q' is the event ${\beta_S}^{\infty}(A_0 \cap P) \supseteq Q$.
- W is the event that all facets $F_j'(Q)$ for $j = 0 \dots J(k-m)$ are internally spanned.

By Lemma 2.2.8 the events Q' and W together imply that P is internally spanned. Therefore:

$$R_S(P, p) \geq \mathbb{P}_p(Q') \cdot \mathbb{P}_p(W)$$

From Equation 2.3.9 we can deduce that: $\lim_{k\to\infty} \mathbb{P}_p(W) = 1$ We then want to show that this can be extended to:

$$\lim_{k\to\infty} P_p(Q') = 1$$

To this end, We say the origin is a good site if T (the smallest possible seed) is occupied and so are all sets $F_i(T)$ for $i \in \mathbb{N}$. In this case, we can immediately apply Equation 2.3.9 to get:

$$\mathbb{P}_p(\text{origin is a good site}) > 0$$

Next, we say a point x is a good site in A_0 if the origin is a good site in $A_0 - x$. By vertex transitivity, this again has positive probability.

Now we claim that if Q contains a good site $x \in Q$, then Q gets filled by the dynamics restricted to P. To see this, consider the set:

$$X = x + \text{Poly}_S(n + 2m) = x + T \oplus Q \oplus Q$$

By definition of x being a good site we know that X is internally spanned. Moreover, as $x \in Q$ we have:

$$X \subseteq Q \oplus T \oplus Q \oplus Q = \text{Poly}((3m + n)b) \subseteq \text{Poly}(kb) = P$$

So X gets filled by the dynamics restricted to P. At the same time, $x \in Q$ and $Q = -Q$ imply that $0 \in x + Q$. From this we can conclude that:

$$Q \subseteq (x + Q) \oplus Q \subseteq x + Q \oplus Q \oplus T = X$$

So indeed, if Q contains a good site, Q gets filled by the dynamics restricted to P.

Then, by ergodicity we have:

$$\lim_{k\to\infty} \mathbb{P}_p(\text{There is a good site inside } Q) = 1$$

and thus:

$$\lim_{k\to\infty} \mathbb{P}_p(Q') = 1$$

■

2.4 Renormalization

In this section, based on Lemma 2.3.3 and the lower bound on π_c from Theorem 2.2.13 we seek to prove the following theorem:

Theorem 2.4.1. *Given any generating set $S' \subseteq \mathbb{Z}^2$ we have $\pi_c = 0$ for $\beta_{S'}$.*

In this section, we will first sketch the proof of this theorem to motivate the introduction of the required propositions, lemmas. After these are all introduced and proven, we proceed to a terse proof of this theorem based on the rest of this section.

From a bird's eye view, the proof proceeds as follows: We start by formally defining the renormalization procedure for which the section is named. This is a mapping $\mathrm{Re} : 2^{\mathbb{Z}^2} \to 2^{\mathbb{Z}^2}$. Then, we show that convex bootstrap percolation after the renormalization dominates (in a loose sense) convex bootstrap percolation before the renormalization procedure. Next, we use the bound on π_c from Theorem 2.2.13 to show that β_S starting from $\mathrm{Re}(A)$ percolates exponentially fast. Then, using the (loose sense of) domination, we conclude that $\beta_S(A)$ must also percolate exponentially fast.

Throughout this section, we will often presume S to be square. That is, we assume that

$$S = (\mathrm{Sq}_r \cap \mathbb{Z}^2) \setminus \{0\}$$
$$\mathrm{Sq}_r = \{x \in \mathbb{R}^2 \mid \|x\|_\infty \leq r\}$$

This is needed in many of our proofs for various reasons. However, this need not be an obstacle, since for any generating set Q, we can find some square $S = \mathrm{Sq}_r \cap \mathbb{Z}^2 \supseteq Q$. Thus, β_Q dominates β_S. So exponential percolation for square generating sets means exponential percolation for all generating sets.

Now we define the renormalization procedure. Informally, we partition $\mathbb{Z}^2$ into a grid of squares. Then, to apply our work on polygons, we replace each square by a polygon containing that square. This means we have a covering instead of a partitioning since neighboring polygons may overlap. We call the original $\mathbb{Z}^2$ the 'base grid' and the grid of polygons the 'renormalized grid'. A node in the renormalized grid thus corresponds to a shifted polygon. We will consider a renormalized node to be occupied whenever the corresponding polygon in the base grid is fully occupied. The renormalization procedure is parameterized by a scale factor m. This simply scales the polygons and underlying grid we use for our covering.

As a base for our covering polygon, we set Sq_k equal to the smallest square that contains the polygon P_S defined in Equation 2.2.7. That is we take:

$$k = \min\{n \in \mathbb{N} \mid \forall s \in S \mid \|s\|_\infty \leq n\}$$

and define our base polygon as $P_S \oplus \mathrm{Sq}_k$. Our choice for P_m allows us to use the results from Lemma 2.2.11 regarding P_S. Then, for a scaling factor $m \in \mathbb{N}$ we define the scaled polygon P_m. In turn, we use that to define the shifted polygons $P_m(x)$ and those are used to define the renormalization and normalization maps:

$$\begin{aligned} P_m &= m(P_S \oplus \mathrm{Sq}_k) \\ P_m(x) &= 2km\, x + P_m \\ \mathrm{Re}_m(A) &= \{x \in \mathbb{Z}^d \mid P_m(x) \cap \mathbb{Z}^d \subseteq A\} \\ \mathrm{Nm}_m(A) &= \bigcup_{x \in A} P_m(x) \cap \mathbb{Z}^d \end{aligned}$$

Note that Nm_m is a right-inverse of Re_m. That is:

$$\mathrm{Re}_m(\mathrm{Nm}_m(A)) = A$$

In the following theorem, we provide the 'loose sense of domination' we mentioned earlier.

Theorem 2.4.2. *Let $S = (\mathrm{Sq}_r \cap \mathbb{Z}^2) \setminus \{0\}$ for some $r \in \mathbb{N}^+$.*

Then, for any $m \in \mathbb{N}^+$ there exists a τ such that, for any $A \subseteq \mathbb{Z}^2$:

$$\beta_S{}^t(\mathrm{Re}_m(A)) \subseteq \mathrm{Re}_m\big(\beta_S{}^{\tau\, t}(A)\big) \tag{2.4.1}$$

Before we can proceed to the proof of this theorem, we need two lemmas. First up is a lemma that describes how a single convex half of S affects growth of entire sets, rather than single points.

Lemma 2.4.3. *Given $C = S \cap H(u)$ a minimal convex half of S, take a set of occupied vertices A, and an arbitrary finite set of vertices E.*

Then

$$E \oplus C \subseteq E \cup A \tag{2.4.2}$$

implies:

$$E \cup A \subseteq {\beta_S}^{\#(E \setminus A)}(A)$$

Proof. We call A the active set, and E the extended set. Moreover, we label $B = E \setminus A$ the inactive set. Now, given any $b \in B$ we define the rank of b recursively:

$$r(b) = \begin{cases} 0 & \text{if } b + C \cap B = \emptyset \\ 1 + \max_{x \in C} r(b + x) & \text{otherwise} \end{cases}$$

For any $b \in B$ the recursive definition above is guaranteed to terminate. For, were it not to terminate, there would exist an infinite sequence $[b_i] \in [B]$ with $b_0 = b$ and $b_{i+1} - b_i \in C$. Now, recall that $\sigma_S(u) > 0$ (defined in Equation 2.2.2). So we have:

$$b_{i+1} \cdot u \geq b_i \cdot u + \sigma_S(u)$$

So the sequence $[b_i]$ could never repeat. This leads to a contradiction, since B is finite. We then get the bound:

$$r(b) \leq \#B = \#(E \setminus A)$$

Using this bound, it suffices to show the following:

$${\beta_S}^n(A) \supseteq \{b \in B \mid r(b) < n\}$$

We will prove this by induction on n. The base case $n = 0$ is vacuously true. Now suppose it holds up to some value k. Then consider a $b \in B$ with $r(b) = k$. By definition of $r(b)$ we know that for all $v \in b + C$ we have $r(v) < n$. So by the induction hypothesis $b + C \in {\beta_S}^k(A)$. Therefore $b \in {\beta_S}^{k+1}(A)$. ∎

For the next lemma we need to define two new sets: quadrants Q and bounding boxes Bb.

$$Q^{\pm_1 \pm_2} = \left\{(x, y) \in \mathbb{R}^2 \mid \pm_1 x \geq 0 \wedge \pm_2 y \geq 0\right\}$$
$$\mathrm{Bb}(A) = \bigcap \{x + Q \mid x \in A \wedge Q \text{ is a quadrant} \wedge A \subseteq x + Q\}$$

There are four quadrants: Q^{++}, Q^{+-}, Q^{-+} and, Q^{--}. As every rectangle can be written as an intersection of shifted quadrants, our bounding box of a set A is the smallest rectangle that contains A. We will often write Bb as a function taking multiple points as an argument as abuse of notation for the bounding box of those points. That is $\mathrm{Bb}(x, y \ldots z) = \mathrm{Bb}(\{x, y \ldots z\})$.

With these definitions, we can introduce the following lemma. This lemma allows us to say certain bounding boxes are contained in $\mathrm{Nm}(C)$ when C is a minimal convex half of a square generating set.

Lemma 2.4.4. *Take S to be a square generating set, Q some quadrant, and $u \in Q \setminus V_S$. Then set the minimal half C and normalized minimal half A:*

$$C = H(u) \cap S \in \mathcal{C}_S$$
$$A = \mathrm{Nm}_m(C)$$

Then, if $a', b' \in C$ and $\tilde{a}, \tilde{b} \in P_m$ are chosen such that $a' - b' \in Q$ and $\mathrm{Bb}(\tilde{a}, \tilde{b}) \in P_m$. It follows that:

$$\mathrm{Bb}(2mk\, a' + \tilde{a}, 2mk\, b' + \tilde{b}) \subseteq A$$

Proof. As $a', b' \in C \subseteq S$ and S is square, we have:

$$\mathrm{Bb}(a', b') \cap \mathbb{Z}^2 \subseteq S \cup \{0\}$$

Furthermore, from $a' - b' \in Q$ we can conclude that:

$$\mathrm{Bb}(a', b') \subseteq a' - Q$$

Moreover, from $a' \in C$ we know $a' \neq 0$. Then we take $a' \in C \subseteq H(u)$ and $u \in Q$ to conclude that:

$$a' - Q \subseteq H(u) \setminus \{0\}$$

This leads us to:

$$\mathrm{Bb}(a', b') \cap \mathbb{Z}^2 \subseteq S \cap H(u) = C$$

And so, finally, we have the following deduction:

$$\begin{aligned} A &= \mathrm{Nm}_m(C) \\ &\supseteq \mathrm{Nm}_m(\mathrm{Bb}(a', b') \cap \mathbb{Z}^2) \\ &= 2mk\,\mathrm{Bb}(a', b') \oplus P_m \\ &\supseteq 2mk\,\mathrm{Bb}(a', b') \oplus \mathrm{Bb}(\tilde{a}, \tilde{b}) \\ &\supseteq \mathrm{Bb}(2mk\,a + \tilde{a}, 2mk\,b + \tilde{b}) \end{aligned}$$

■

Now that we have shown these two lemmas, we can proceed to prove Theorem 2.4.2:

Proof of Theorem 2.4.2. First, we define the discrete version of P_m:

$$\tilde{P}_m = P_m \cap \mathbb{Z}^2$$

It suffices to show that for each minimal convex half $C \in \mathcal{C}_S$ we can find a τ_C such that for any $X \subseteq \mathbb{Z}^2$ we have:

$$C \subseteq \mathrm{Re}_m(X) \Rightarrow \tilde{P}_m \subseteq {\beta_S}^{\tau_C}(X)$$

We can then set $\tau = \max\{\tau_C \mid C \in \mathcal{C}_S\}$ and, using translation invariance, conclude that:

$$\beta_S(\mathrm{Re}_m(A)) \subseteq \mathrm{Re}_m({\beta_S}^{\tau}(A))$$

The full form of theorem 2.4.2 then follows by a trivial inductive argument.

Thus, we fix the minimal convex half C and set $u_C \notin V_S$ such that $C = H(u_C) \cap S$. We also set:

$$A = \mathrm{Nm}_m(C) = \tilde{P}_m \oplus 2mkC$$

So, for any set X such that $C \subseteq \mathrm{Re}_m(X)$ we have $A \subseteq X$. Now, we need to show that

Now, we can proceed in terms of A and P_m and only work in the base grid. Here, we look to apply Lemma 2.4.3. This means that, in addition to our A, we need to produce an E that meets the following two requirements:

$$E \oplus C \subseteq E \cup A \tag{2.4.3}$$

$$\tilde{P}_m \subseteq E \cup A \tag{2.4.4}$$

Lemma 2.4.3 would then yield:

$$\begin{aligned} \tau_C &= \#(E \setminus A) \\ \tilde{P}_m &\subseteq E \cup A \subseteq {\beta_S}^{\tau_C}(A) \end{aligned}$$

Which would prove our theorem.

To produce this E, we now change from discrete to continuous sets by taking the convex hull. That is, we define the following sets:

$$\bar{A} = \text{Conv}(A)$$
$$\bar{C} = \text{Conv}(C, \{0\})$$

Next, we define the 'top' of P_m:

$$t_m = \arg\max_{x \in \bar{P_m}} x \cdot u_C$$

This is a vertex of P_m and per Lemma 2.2.11 it follows that $t_m + \bar{C} \subseteq P_m$.

Next, we make use of the conical sum of C. As we are working in 2 dimensions, conical sums are easily described. We can find two unit vectors $u_1, u_2 \in V_S$ such that:

$$\text{Coni}(C) = H(u_1) \cap H(u_2)$$

We then use this to define 'border points' of C as:

$$b_1 = \arg\max_{b \in C \cap N(u_1)} \|b\| \qquad b_2 = \arg\max_{b \in C \cap N(u_2)} \|b\|$$

Now we pick $c, c_t \in \mathbb{Z}^2$ such that:

$$\text{Bb}(b_1, b_2) = \text{Conv}(b_1, c, b_2, c_t)$$

As S is square, we have $c, c_t \in S$. Moreover, one of c, c_t lies inside $\text{Coni}(C)$. We call this one c and the other c_t.

Now, we can finally define our E. This is based on the convex polygon $\bar{E}$.

$$\begin{aligned}\bar{E} &= t_m + (2mk \,\text{Bb}(b_1, b_2) \cap \text{Coni}(C)) \\ &= t_m + 2mk \,\text{Conv}(b_1, 0, b_2, c) \\ E &= \bar{E} \cap \mathbb{Z}^2\end{aligned}$$

Note that all vertices of $\bar{E}$ are the 'top' of some $P_m(x)$.

Now, to prove Equation 2.4.3 we change to a continuous form. The discrete form follows simply by taking the intersection with $\mathbb{Z}^2$. Thus, we seek to prove:

$$\bar{E} \oplus \bar{C} \subseteq \bar{E} \cup \bar{A}$$

Now, note that by definition of u_1, u_2 we have: $u_1, u_2 \notin \bigcup\{H(u)^C \mid u \in \bar{C}\}$. Moreover, we have $0 \in \bar{C}$. Then, by Lemma 2.1.7 it follows that:

$$\bar{E} \oplus \bar{C} = \bar{E} \cup t_m + \left(\left(2mk \,\text{Conv}(b_1, c) \oplus \bar{C}\right) \cup \left(2mk \,\text{Conv}(b_2, c) \oplus \bar{C}\right)\right) \tag{2.4.5}$$

Now, by definition of c we know that $\text{Conv}(b_1, c)$ is either vertical or horizontal. Therefore, $\text{Conv}(b_1, c) = \text{Bb}(b_1, c)$. Moreover, if we take Q to be the quadrant such that $u_c \in Q$ we have: $c - b_1 \in Q$. Finally, we have $t_m + \bar{C} \subseteq P_m$ and $b_1, c \in C$. Then, by the definition of $\oplus$ and Lemma 2.4.4 we get:

$$\begin{aligned}(t_m + \bar{C}) \oplus 2mk \,\text{Conv}(b_1, c) &= \bigcup_{\tilde{a} \in t_m + \bar{C}} \tilde{a} + 2mk \,\text{Conv}(b_1, c) \\ &= \bigcup_{\tilde{a} \in t_m + \bar{C}} \text{Bb}(\tilde{a} + 2mk\, b_1, \tilde{a} + 2mk\, c) \\ &\subseteq A\end{aligned}$$

The same argument holds when we replace b_1 by b_2. Combining the above with Equation 2.4.5 we get:

$$\bar{E} \oplus \bar{C} \subseteq \bar{E} \cup A$$

Thus, we have proven Equation 2.4.3.

All that remains is to prove Equation 2.4.4. To do this, we again change to a continuous form. Thus, we seek to prove:

$$P_m \subseteq \bar{E} \cup \bar{A}$$

The discrete form follows simply by taking the intersection with $\mathbb{Z}^2$.

First, note that as $P_S \subseteq \mathrm{Sq}_k$ we have $P_m \subseteq \mathrm{Sq}_{2mk}$. Moreover, we have $P_m \subseteq t_m + \mathrm{Coni}(C)$. Thus:

$$P_m \subseteq \mathrm{Sq}_{2mk} \cap (t_m + \mathrm{Coni}(C))$$

We will construct an intermediate set D. Without loss of generality, we assume that $u_c \in Q^{++}$. Then we set:

$$\begin{aligned} v_1 &= t_m + 2mk\, b_1 \\ v_2 &= t_m + 2mk\, b_2 \\ z &= -2mk\, r\, (1,1) \end{aligned}$$

Where r is the presumed 'radius' of our generating set. That is: $S = (\mathrm{Sq}_r \setminus \{0\}) \cap \mathbb{Z}^2$. Note that we can write $z = 2mk\, z'$ with $z' = (-r, -r) \in C$. That is, z is the center of the farthest translate of P_m in A. Then we set:

$$\begin{aligned} D &= E \cup \mathrm{Bb}(v_1, z) \cup \mathrm{Bb}(v_2, z) \\ &= (t_m + \mathrm{Coni}(C)) \cap \mathrm{Bb}(v_1, v_2) \cup \mathrm{Bb}(v_1, z) \cup \mathrm{Bb}(v_2, z) \\ &\supseteq (t_m + \mathrm{Coni}(C)) \cap \mathrm{Bb}(v_1, v_2, z) \end{aligned}$$

Note that by square symmetry of P_m we have $\mathrm{Bb}(t_m, 0) \subseteq P_m$. Moreover, we have $z - v_1 \in Q^{++}$ and $z - v_1 \in Q^{++}$. Then, by Lemma 2.4.4 it follows that $\mathrm{Bb}(v_1, z) \subseteq A$ and $\mathrm{Bb}(v_2, z) \subseteq A$, and thus:

$$D \subseteq E \cup A$$

Next we will show that

$$D \supseteq (t_m + \mathrm{Coni}(C)) \cap \mathrm{Bb}(v_1, v_2, z) \overset{*}{\supseteq} \mathrm{Sq}_{2mk} \cap (t_m + \mathrm{Coni}(C)) \supseteq P_m \tag{2.4.6}$$

Everything except for the inclusion marked by $*$ is already known. Now recall c_t was the vertex opposite c in $\mathrm{Bb}(b_1, b_2)$. We can then write:

$$\mathrm{Bb}(v_1, v_2, z) = \mathrm{Bb}(t_m + 2mk\, c_t, z) = (t_m + 2mk\, c_t - Q^{++}) \cap (z + Q^{++})$$

Note that, by definition of z, we always have: $(-1, -1) \in z - Q^{++}$.

Now we distinguish two cases. Either 0 lies on the interior of $c_t - Q^{++}$ or 0 lies on an edge of $c_t - Q^{++}$. It is not possible that 0 lies outside $c_t - Q^{++}$ or $0 = c_t$ because S is square, and so conical angle of $\mathrm{Coni}(C)$ is larger than $90°$.

In the first case, as $c_t \in \mathbb{Z}^2$ we must have:

$$(1,1) \in c_t - Q^{++}$$

and thus:

$$\begin{aligned} \mathrm{Bb}(v_1, v_2, z) &= (t_m + 2mk\, c_t - Q^{++}) \cap (z + Q^{++}) \\ &\supseteq \mathrm{Bb}\big(2mk(1,1)\,,\, -2mk(1,1)\big) \\ &= \mathrm{Sq}_{2mk} \end{aligned}$$

which proves Equation 2.4.6 in the first case.

In the second case, one of b_1 or b_2 is vertical or horizontal. Without loss of generality, we presume $b_1 \in N(e_1)$. Where e_1 is either of the normal basis vectors. Then, $c_t \in \mathbb{Z}^2$ yields the first equation below, and the definition of t_m yields the second.

$$\begin{aligned} e_2 &\in c_t - Q^{++} \\ t_m &\in N(e_1) \end{aligned}$$

And thus:

$$\begin{aligned} \mathrm{Bb}(v_1, v_2, z) &= (t_m + 2mk\, c_t - Q^{++}) \cap (z + Q^{++}) \\ &\supseteq \mathrm{Bb}\big(2mk\, e_2\,,\, -2mk(1,1)\big) \\ &= \mathrm{Sq}_{2mk} \cap (t_m + H(e_1)) \end{aligned}$$

Furthermore, note that from $b_2 \in N(e_1)$ we can conclude that:

$$\mathrm{Coni}(C) \subseteq (t_m + H(e_1))$$

We can then combine the above two equations to get:

$$\mathrm{Bb}(v_1, v_2, z) \cap \mathrm{Coni}(C) = \mathrm{Sq}_{2mk} \cap \mathrm{Coni}(C)$$

Which proves Equation 2.4.6 in the second case. Thus, we have proven Equation 2.4.6 in either case. ■

So, now we have proven our loose form of domination. Now, we still need to conclude the dominating percolation is exponential, and use that to conclude the dominated percolation is also exponential.

To that end we define the starting time t_0 as:

$$t_0 = \#P_m \tag{2.4.7}$$

We then use that to define an initial condition and use Theorem 2.4.2 to get:

$$\begin{aligned} A_{t_0} &= {\beta_S}^{t_0}(A_0) \\ {\beta_S}^{t}(\mathrm{Re}_m(A_{t_0})) &\subseteq \mathrm{Re}_m({\beta_S}^{\tau\, t + t_0}(A_0)) \end{aligned}$$

Now suppose that bootstrap percolation starting from the stochastic initial condition $\mathrm{Re}_m(A_{t_0})$ percolates exponentially, say with exponent $\gamma > 0$. We could then reason as follows:

$$\begin{aligned} Ce^{-\gamma t} &\geq \mathbb{P}_p(0 \notin {\beta_S}^{t}(\mathrm{Re}_m(A_{t_0})) \\ &\geq \mathbb{P}_p(0 \notin \mathrm{Re}_m({\beta_S}^{\tau t + t_0}(A_0)) \\ &\geq \mathbb{P}_p(0 \notin {\beta_S}^{\tau t + t_0}(A_0) \end{aligned} \tag{2.4.8}$$

Then, looking at equations (1.0.4) and (1.0.5) for the definition of $\gamma(p)$ and π_c, we would have the following:

$$\begin{aligned} \gamma(p) &= \frac{\gamma}{\tau} \\ \pi_c &< p \end{aligned}$$

As we only had the restriction $p > 0$ this would actually show $\pi_c = 0$. So, we wish to prove that indeed there is a $\gamma > 0$ for which the reasoning in (2.4.8) holds. Note that both γ and τ may depend on m.

Now we define events I_x given $x \in \mathbb{Z}^2$, and the corresponding set $I \subseteq \mathbb{Z}^2$:

$$\begin{aligned} I_x &: P_m(x) \text{ is internally spanned by } A_0 \\ I &= \{x \in \mathbb{Z}^2 \mid I_x\} \end{aligned}$$

Then by definition of the starting time t_0 we have:

$$I \subseteq \mathrm{Re}_m(A_{t_0})$$

So now we want to show that convex bootstrap percolation starting from I percolates exponentially fast. We mean to show this using Theorem 2.2.13. Recall that this gave:

$$\pi_c \leq 1 - \frac{1}{\#S - 1}$$

Now, by definition of R_S we have:

$$P_p(I_x) = R_S(P_m(x), p)$$

Naively, we might then argue that by Lemma 2.3.3 we can simply pick m sufficiently large that:

$$R_S(P_m, p) > \frac{1}{\#S - 1} > \pi_c$$

and thus claim exponential growth. There are two obstacles to this line of reasoning. First of all, we do not know if we can write P_m in the form $\text{Poly}_S(mb)$ for some integer vector b. Therefore we cannot yet apply Lemma 2.3.3 to P_m. This will be solved by Lemma 2.4.5. The second obstacle is more insidious. The result regarding π_c does not apply because our polygons overlap and thus the events I_x are not independent.

Before we deal with the dependence of I_x we first want to prove that Lemma 2.3.3 actually applies to $P_m(x)$. To that end, we prove the following lemma:

Lemma 2.4.5. *If $S = (\text{Sq}_r \cap \mathbb{Z}^2) \setminus \{0\}$, then P_m can be written as $\text{Poly}_S(b)$ for some $b \in \mathbb{Z}^{\#U_S}$.*

Proof. We will prove that there exists some $b \in \mathbb{R}^n$ such that $P_m = \text{Poly}_S(b)$. It then follows that this b is restricted to the integers because the vertices of P_S all lie in $\mathbb{Z}^2$. This is a property of Poly_S. To show this b exists, we need to show all edges of P_m are normal to some vector in U_S.

Notably, scaling does not matter, so it suffices to only treat P_1. Recall this was defined as:

$$\begin{aligned} P_1 &= \text{Sq}_k \oplus P_S \\ &= \text{Sq}_k \oplus \sum \{\text{Conv}(C \cup \{0\}) \mid C \in \mathcal{C}_S\} \end{aligned}$$

Now, a Minkowski sum has an edge normal x, if and only if x is an edge normal of at least one of the summands. Thus, we need to show all summands only have edge normals in U_S. It is clear that the edge normals of Sq_k lie in U_S. The remaining summands are all of the form:

$$\bar{C} = \text{Conv}(C \cup \{0\}) \quad \text{where} \quad C \in \mathcal{C}_S$$

We need to show that these convex polygons have all edge-normals in U_S. To this end, consider the vertices of an arbitrary $\bar{C}$. Without loss of generality, presume that $C = S \cap H(u)$ for some $u \in Q^{++} \setminus V_S$.

It is guaranteed 0 is a vertex of $\bar{C}$. Moreover, given our restriction on u we can find 3 more guaranteed vertices. Firstly, there is the corner $c = (-r, -r)$. This corresponds to direction $-u$. This corner has two edges running along the edges of Sq_r. Their edge normals clearly lie in U_S. Moreover, these edges terminate at two vertices $d_1 = (a, -r)$ and $d_2 = (-r, b)$ with $a \geq 0$ and $b \geq 0$.

The existence of vertices $0, d_1, c, d_2$, combined with $\bar{C}$ being convex, restrict where the remaining vertices can occur. Consider such a remaining vertex v, we then know the following three things:

- $v \in \bar{C} \subseteq \text{Sq}_r$
- $v \notin Q^{++}$ because $v \in \bar{C} \subseteq H(u)$ and $Q^{++} \cap H(u) = \{0\}$.
- $v \notin \text{Conv}(0, d_1, c, d_2) \supseteq Q^{--}$ because it is a vertex of $\bar{C}$.

From this, we can conclude that:

$$v \in (\text{Sq}_r \cap Q^{-+}) \cup (\text{Sq}_r \cap Q^{+-})$$

Moreover, no vertices in Q^{-+} can have an edge to another vertex in Q^{+-}. For, such an edge would mean 0 is not a vertex. Thus, all remaining edges occur between points in a quadrant.

Now, suppose we have an edge (x, y) in $\mathrm{Sq}_r \cap Q^{+-}$. Then $x - y \in \mathrm{Sq}_r$ and thus, we have the parallel edge $(x - y, 0)$. This edge is normal to some vector in U_S, and thus the same holds for the edge (x, y).

The above line of reasoning also works for an edge in $\mathrm{Sq}_r \cap Q^{-+}$. As such, every edge in $\bar{C}$ is normal to some vector in U_S. ■

Now, we still have to deal with the dependence of the events I_x. Luckily, there is a limit to the dependence. Given $\|x - y\|_\infty$ large enough, we know that $P_m(x)$ and $P_M(y)$ do not overlap. Thus for these x and y the events I_x and I_y are independent. The following lemma allows us to use this limit in the dependence.

The lemma below is a simplified version of Theorem B26 from [Lig13, p. 27]. For a proof, we also refer to that book.

Lemma 2.4.6 (Stochastic domination)**.** *Fix $k \geq 1$ and set $\delta = \#\{y \in \mathbb{Z}^2 \mid \|y\|_\infty < k\} = (2k+1)^2$. Now, if $X_x \mid x \in \mathbb{Z}^2$ are random events, with X_x independent from X_y whenever $\|x - y\|_\infty > k$ and, for all $x \in \mathbb{Z}^2$ we have*

$$\mathbb{P}(X_x) \geq 1 - (1 - \sqrt{q})^\delta$$

Then, we set $X = \{x \in \mathbb{Z}^2 \mid X_x\}$ and let Y be an initial condition with density q.

Finally, take $\mathcal{A} \in 2^{\mathbb{Z}^2}$ to be any increasing family of sets. Then we have:

$$\mathbb{P}(X \in \mathcal{A}) \geq \mathbb{P}(Y \in \mathcal{A})$$

In order to make use of this lemma, we set m large enough that:

$$1 - (R_S(P_m, p) - 1)^{k'} = q > 1 - \frac{1}{\#S - 1} \tag{2.4.9}$$

Where k' depends on how much our $P_m(x)$s overlap. That is:

$$k' = 1 + \max\{\|x\|_\infty \mid P_m(x) \cap P_m \neq \emptyset\} < \infty$$

Next, we define the family of initial conditions that do not occupy the origin before some time t:

$$\mathcal{B}_t = \{B \subseteq \mathbb{Z}^2 \mid 0 \in {\beta_S}^t(B)\}$$

At any fixed t this is clearly an increasing family in the sense defined in the stochastic domination lemma. Then, we apply $I \subseteq \mathrm{Re}_m(A_{t_0})$, the stochastic domination lemma, Theorem 2.2.13, and (2.4.9) to conclude that:

$$\begin{aligned} \mathbb{P}_p\big(0 \in {\beta_S}^t(\mathrm{Re}_m(A_{t_0}))\big) &= \mathbb{P}_p(\mathrm{Re}_m(A_{t_0}) \in \mathcal{B}_t) \\ &\geq \mathbb{P}_p(I \in \mathcal{B}_t) \\ &\geq \mathbb{P}_q(A_0 \in \mathcal{B}_t) \\ &\geq 1 - Ce^{-\gamma(p)t} \end{aligned} \tag{2.4.10}$$

Where we take A_0 to be a random initial condition. The corresponding density is q, hence the switch from $\mathbb{P}_p$ to $\mathbb{P}_q$.

We now have everything we need to prove Theorem 2.4.1:

Proof of Theorem 2.4.1. Recall the statement we want to prove:
Given any generating set $S' \subseteq \mathbb{Z}^2$ we have $\pi_c = 0$ for $\beta_{S'}$.

Fix some $p > 0$ and let A_0 be an initial condition with density p. We then need to show we have exponential percolation when starting from this density.

First, we take a new square generating set S that encloses S'. Specifically, we take the smallest square Sq_r such that $S' \subseteq \mathrm{Sq}_r$.

$$S = (\mathrm{Sq}_r \cap \mathbb{Z}^2) \setminus \{0\} \supseteq S'$$

From $S' \subseteq S$ it follows that $\beta_{S'}$ dominates β_S. Hence it suffices to show that we have exponential percolation using S.

Next we pick a target density:

$$q > 1 - \frac{1}{\#S - 1}$$

Then, fix m according to Equation 2.4.9, and t_0 according to Equation 2.4.7. Now, by the renormalization Theorem (2.4.2) and definition of Re_m we get:

$$0 \in {\beta_S}^t(Re_m(A_0)) \ \rightarrow \ 0 \in \mathrm{Re}_m({\beta_S}^{\tau t + t_0}(A_0)) \ \rightarrow \ 0 \in {\beta_S}^{\tau t + t_0}(A_0)$$

We can then combine the above with equation 2.4.10 to get:

$$\mathbb{P}_p\big(T \leq \tau t + t_0\big) \geq \mathbb{P}_p\big(0 \in {\beta_S}^t(\mathrm{Re}_m(A_{t_0}))\big) \geq 1 - Ce^{-\gamma(q)t}$$

This can be rewritten to:

$$\mathbb{P}_p(T > t) \leq Ce^{-\gamma(q)\frac{t - t_0}{\tau}}$$

From this, we can conclude that:

$$\gamma(p) \geq \frac{\gamma(q)}{\tau} > 0$$

■

2.5 Conclusion for theoretical work

Finally, with the result of Theorem 2.4.1 we can produce the main result of this paper. The proof is mostly stringing together previous results.

Theorem 2.5.1. *Conjecture 1.1.3 holds for abelian groups of rank 2. That is, let H be a finitely generated abelian group of rank 2 and let $S \subseteq H$ be a finite symmetric set with $0 \notin S$. Moreover we set k_S as follows:*

$$k_S = \frac{\#\{s \in S \mid \forall n \in \mathbb{N}^+ : n \times s \neq 0\}}{2}$$

Then for β_k on $\Gamma(H, S)$ we have:

$$\pi_c = p_c = \begin{cases} 0 & \text{if} \quad k \leq k_S \\ 1 & \text{if} \quad k > k_S \end{cases}$$

Proof. The case $k > k_S$ is proven in Theorem 2.2.12.

So all that remains is the case $k \leq k_S$. Since H has rank 2, there exists a surjective group homomorphism:

$$\phi : H \rightarrow \mathbb{Z}^2$$

Now, we know that $\pi_c \geq p_c$. Thus it suffices to show that $\pi_c = 0$. To this end, we set $T \subseteq S$ to be all non-periodic elements of S. From Proposition 2.1.12 we then know that:

$$\mu_T(A) \subseteq \beta_k(A) \tag{2.5.1}$$

Next, we take $Q = \phi(T)$. By Theorem 2.4.1 we know that $\pi_c = 0$ for β_Q. By domination, the same must then hold for μ_Q. Then, by Theorem 2.1.13 it follows that $\pi_c = 0$ for μ_T. From this and the domination from (2.5.1) it finally follows that $\pi_c = 0$ for β_k on $\Gamma(H, S)$. ■

Behind this result lie 3 versions of bootstrap percolation that dominate each other. Specifically, we have:

$$\beta_k \geq \mu_S \geq \beta_S$$

Where β_k is k-threshold bootstrap percolation (Equation 1.0.1) taking $k = \frac{\#S}{2}$, μ_S is modified bootstrap percolation (Definition 2.1.11), and β_S is convex bootstrap percolation (Equation 2.2.1). Modified bootstrap percolation is only used in Section 2.1.2. We cannot use convex bootstrap percolation there because it doesn't make sense to talk about half planes in abelian groups with a toroidal part. Note that in the canonical case of $\mathbb{Z}^n$ with $S = \{e_1 \dots e_n\}$ modified and convex bootstrap percolation are the same.

Below we have a brief overview of the structure of the results supporting the main result for the case $k > k_S$. We list the most relevant lemmas and theorems, and show their dependence by indentation.

Theorem 2.1.13: For modified bootstrap percolation on any Cayley graph of a finitely generated abelian group H, we can discard the toroidal part of H.
  Theorem 2.1.8: The abelian structure theorem.
Theorem 2.4.1: $\mathbb{Z}^2$ has $\pi_c = 0$ for convex bootstrap percolation.
  Theorem 2.4.2: The renormalization theorem.
    Lemma 2.4.3: Sufficient condition for a set being internally spanned, and an upper bound on the number of steps require to get there.
  Lemma 2.4.6: Stochastic domination.
  Theorem 2.2.13: Peierls estimate gives upper bound on π_c.
  Lemma 2.3.3: $R_S(\text{Poly}_S(kb), p)$ goes to 1 as k goes to infinity.
    Lemma 2.2.8: Growth of a polytope through facets.
    Equation 2.3.9: Equation for the probability that all 'shells' around a polytope are internally spanned.

This structure closely follows the argument used by [Sch92], which got the result for any dimension, but using the canonical generators rather than an arbitrary set of generators. Our more general case added some significant difficulties. This was most noticeable with the renormalization theorem. Here, we covered the plane with overlapping polygons, whereas in [Sch92] it was possible to partition the space into hypercubes. This makes proving the actual theorem more difficult, and is also why we require the stochastic domination theorem.

Another difficulty of our case as compared with the case of canonical generators lies with Lemma 2.3.3. In the canonical case, our sets $\text{Poly}_S(b)$ are boxes. This means that when we grow a facet, that facet doesn't change. In our case, growing a facet changes that facet. This makes it much harder to use inductive arguments. In the 2D case, we saw that growing an edges shrinks the edge. In the general case, the changes to a facet can be more complicated.

2.5.1 Extending result to arbitrary abelian groups

The case $k > k_S$ in Conjecture 1.1.3 is proven for all finitely generated abelian groups in Theorem 2.2.12. The case $k \leq k_S$ remains unproven for abelian groups of rank above 2. To fully prove the conjecture it would suffice to show that convex bootstrap percolation on $\mathbb{Z}^d$ has $\pi_c = 0$ for arbitrary dimension d and generating set S.

It should be noted that we do have some weaker results that apply to the case $k \leq k_S$ in arbitrary dimension. They are listed below.

- From [Sch92] we know that $\pi_c = 0$ on $\Gamma(\mathbb{Z}^d, \{e_1 \dots e_d\}$ for convex, modified, and k_S-threshold bootstrap percolation. Here we take $e_1 \dots e_d$ to be the canonical basis vectors.
- By domination, the above result means that $\pi_c = 0$ on $\Gamma(\mathbb{Z}^d, S)$ for d-threshold bootstrap percolation. This presumes that S is a generating set as per Definition 2.2.2. Moreover, it presumes that S actually generates $\mathbb{Z}^d$, so that $\Gamma(\mathbb{Z}^d, S)$ is connected.
- From Theorem 2.2.13 we know that:

  $$\pi_c \leq \frac{1}{\#S - 1} < 1$$

  for convex bootstrap percolation on $\Gamma(\mathbb{Z}^d, S)$ for any generating set S. By domination this extends to modified and k_S-threshold bootstrap percolation. It is interesting to compare this to the result of Lemma 2.1.4. This lemma means that:

  $$p_c \leq 1 - q$$

  where q is the critical threshold for site percolation. However, the best bound on q we could find is actually the same: $q > \frac{1}{\#S-1}$ based on the exact same argument as Theorem 2.2.13.

- From Theorem 2.1.9 we know that all k-forts are infinite.

These results combined really support the conjecture.

Furthermore, we have an idea for how one might extend our argument for $\mathbb{Z}^2$ to $\mathbb{Z}^r$. Most of our intermediary results already are formulated and proven for arbitrary dimension. However, a few key results are specific to $\mathbb{Z}^2$. Notably, there are two big hangups: the renormalization theorem, and the seed argument.

Of these, the renormalization theorem is the hardest to extend. Currently, the supporting Lemma 2.4.4 is formulated in 2D, but the proof generalizes to any dimension. One only needs to generalize quadrants to their higher dimensional equivalent of *Orthants* and define higher dimensional bounding boxes analogously.

The biggest obstacle with renormalization lies in the proof of Theorem 2.4.2 itself. An idea for extending this proof to arbitrary dimension is to show that:

$$\mathrm{Nm}_m(C \cup \{0\}) = \mathrm{Conv}(mC) \oplus P_m$$

for any minimal convex half $C \in \mathcal{C}_S$. This seems to be true or at least achievable by tweaking the definition of P_m based on our cursory examination. We would then need to combine this with Lemma 2.4.3 to show that $\mathrm{Nm}_m(C)$ grows to $\mathrm{Nm}_m(C \cup \{0\})$. We suspect this approach would also greatly simplify the proof of Theorem 2.4.2.

In the seed argument, at some point we want to describe the following scenario. We have a polytope $P = \mathrm{Poly}_S(kb)$ and wish to grow it to $P' = \mathrm{Poly}_S((k+1)b)$. As per Lemma 2.2.8 this can be done by growing a finite sequence of facets. This growth occurs when these facets are internally spanned. Since the facets have a lower dimension, the probability of them being internally spanned is easier to describe. However, we need some way to describe the intermediate polytopes. Or rather, we need some way to describe these intermediate facets we need to be internally spanned. In 2D, these facets are edges, and we were able to describe them sufficiently with Equation 2.3.7. Combining that equation with the know probability of a line being internally spanned yields Equation 2.3.8. This is of the form:

$$\mathbb{P}_p(P' \text{ internally spanned} \mid P \text{ internally spanned}) \geq \left(1 - e^{f(P)}\right)^J \tag{2.5.2}$$

where $f(\mathrm{Poly}_S(kb)) = O(k)$.

The remainder of the seed argument is based on the above equation. Presuming (2.5.2) holds in dimension $d-1$, one can generalize the proof of Theorem 2.3.2 to yield $p_c = 0$ in dimension d. Similarly, the proof of Lemma 2.3.3 generalizes to yield:

$$\lim_{k \to \infty} R_S(\mathrm{Poly}_S(kb), p) = 1$$

in dimension d presuming (2.5.2) holds in dimension $d-1$. Thus, if we can get something of the form Equation 2.5.2 for higher dimensions, that would extend our results.

To do this in dimension d, it would be very useful to have a bound of the following form for polytopes of dimension $d-1$:

$$R_S(\mathrm{Poly}_S(kb), p) \geq 1 - e^{f(k)} \tag{2.5.3}$$

where $f(k) = O(k)$. This would take the role of Equation 2.3.2 which gives such a bound for 1-dimensional polytopes. Note that it is possible to adapt Proposition 3.2 in [Sch92] to conclude a bound like above from $\pi_c = 0$. This is done much like our Lemma 2.3.3 is adapted from Lemma 3.2 in [Sch92]. Since we have $\pi_c = 0$ for 2D, this would yield the desired bound for 3D.

Next, we sketch an approach for proving Equation 2.5.2 in dimension d assuming we have Equation 2.5.3 in dimension $d-1$. The challenge here is to describe the intermediate facets $F_i'(P)$ (defined in Equation 2.3.6), given only P. So, let F_1 be the facet normal to u_i in polytope $P = \mathrm{Poly}_S(kb)$ and let F_2 be the facet $F_i'(P)$. Our idea is to rewrite F_2 as follows:

$$F_2 = (F_1 \ominus A) \oplus B$$

for some masks A and B. Using this expression we can get a crude bound:

$$R_{S\cap N(u)}(F_2, p) \geq R_{S\cap N(u)}(F_1 \ominus A)^{\#B} \tag{2.5.4}$$

Because, if for every $b \in B$ the set $b + F_1 \ominus A$ is internally spanned then F_2 is internally spanned. We then simply ignore the dependence of these events. This is not an issue because these are all increasing events.

The idea is then to pick a mask A that can be used universally for all intermediary facets $F_i'(P)$, $1 \leq i \leq J$. We use that to get a lower bound on:

$$R_{S\cap N(u)}(F_i'(P) \ominus A)$$

We then consider all Bs needed given our fixed A. This gives a finite upper bound on $\#B$. Then, we get a lower bound on Equation 2.5.4 that holds for all $F_i'(P)$. Notably, we need these bounds on A and B to be independent of the scaling factor k. We believe this, combined with Equation 2.5.3 for the facets would then lead to Equation 2.5.2.

Our approach for the 2D case was very similar to what is outlined above. However, there we did not need the power $\#B$. Instead, for F_1 and F_2 line segments, we had the much simpler bound:

$$R_{S\cap N(u)}(F_2, p) \geq R_{S\cap N(u)}(F_1 \ominus A)$$

Now suppose that the above outline holds, *and* the renormalization argument holds in arbitrary dimensions, *and* we can conclude Equation 2.5.3 from $\pi_c = 0$. In that case, we could make an inductive argument on the dimension d to conclude $\pi_c = 0$ for $\mathbb{Z}^d$. The base case would be $d = 1$ which holds trivially. Then, we use the above outline to conclude that for $d + 1$ we have $R_S(\mathrm{Poly}_S(kb), p) \to 1$. We then use the renormalization theorem, together with stochastic domination and Theorem 2.2.13 to conclude that $\pi_c = 0$ for $d + 1$. Finally, we can conclude Equation 2.5.3 from $\pi_c = 0$. This would finish the inductive argument. Since Theorem 2.1.13 holds in arbitrary dimension, we would then be able to confirm Conjecture 1.1.3!

Chapter 3

Experimental Work

In the previous sections, we have progressed towards exploring Conjecture 1.1.2 by mathematical methods. Here, we take a more empirical approach. This comes at the cost of rigour, but allows us to explore much more difficult problems.

Recall that the conjecture states that for undirected Cayley graphs of amenable groups with degree $2k$ we have $p_c = 0$ for k-threshold bootstrap percolation. To test this, we will compute bootstrap percolation on an increasing sequence of finite subgraphs of these Cayley graphs. It should be noted that by working with finite graphs, we lose a lot of our previous theoretical properties. For example, the 0-1 law no longer holds. Moreover, for our original definition of the 'critical point' p_c we always have $p_c = 1$. For any initial density $p < 1$ the probability that the entire graph (of size n) is completely unoccupied is $(1-p)^n > 0$ so $\mathbb{P}_p(T = \infty) < 1$.

We hope to see some form of critical behavior in these finite subgraphs. That is, given a finite subgraph G, we hope that there is some critical density q_G such that initial densities slightly above q_G lead to occupying almost the entire graph, and initial densities slightly below q_G show very little growth. The range around q_G where we transition from one form of behavior to the other form is called the 'phase transition'. Then, if the conjecture holds, one would expect that this critical point q_G converges to 0 when we let the subgraph G increase in size.

It is of course possible this convergence is so slow it is unnoticeable. Conversely, convergence to a sufficiently small positive value is indistinguishable from convergence to 0. Moreover, it is not inconceivable that $p_c = 0$ whilst q_G does not converge to 0. Finally, there are many different increasing sequences of finite subgraphs; the choice of this sequence could affect the convergence of q_G. The choice could also affect the speed of convergence, the value q_G converges to, or whether we have convergence at all. As such, the aim of this experiment is not to give certainty. Instead, it is meant to give guidance for further research.

An issue of approximating by finite subgraphs is the introduction of a 'border'. The border of a subgraph is the set of edges with exactly one vertex inside the subgraph. Formally, given a graph (V, E) and a subgraph $F \subseteq V$ we define the border $\delta(F)$ as:

$$\delta(F) = E \cap (F \times (V \setminus F))$$

A vertex $v \in F$ lies on the border of F if it is a vertex of any edge in $\delta(F)$. These vertices are essentially missing one of their neighbors. This is a simple issue with vertices on the border. Having fewer neighbors makes it harder to activate using bootstrap percolation. Another issue is that vertices on the border are locally different from the Cayley graph. That is, around these points the sub-graph 'looks' different from the Cayley graph. Since we are looking to approximate the Cayley graph, this is undesirable.

A potential solution is to not take subgraphs of the Cayley graphs, but instead take Cayley graphs of finite groups that are locally similar. One can take a sequence of such graphs that 'converges' to the infinite Cayley graph. However, it is less evident that behavior of such locally similar graphs informs us about behavior of the full graph.

3.1 Computing bootstrap percolation

To do our experiments, we need to actually compute bootstrap percolation. Moreover, as we want to simulate graphs of increasing size, we aimed for an efficient algorithm. Throughout this section, we take (V, E) to be a finite graph. We also set $n = \#V$ and take A to be the adjacency matrix of our graph. From this adjacency matrix we have an implicit enumeration of V. Finally, k is the threshold we use for bootstrap percolation.

Recall that k-threshold bootstrap percolation on the graph (V, E) is defined as iterated steps of the following function:

$$\beta_k : 2^V \to 2^V$$
$$\beta_k(A) = A \cup \{x \in V \mid \#E \cap (\{x\} \times A) \geq k\}$$

Where we call A our set of active points.

Now, as our graph is finite, the adjacency matrix A has implicitly enumerated the nodes V. Thus we can encode any subset $X \subseteq V$ as a vector of 0s and 1s. That is, we identify such a subset X as an element of $\{0,1\}^n$. We can then get an equivalent formulation based on the threshold function $t_k : \mathbb{N}^n \to \{0,1\}^n$ that does pointwise greater-than-or-equal comparison with k. That is, consider $\geq$ a Boolean operator $\mathbb{N} \times \mathbb{N} \to \{0,1\}$, then we have:

$$t_k((x_1 \dots x_n)) = (x_1 \geq k \dots x_n \geq k)$$

Similarly, we let the logical or operator $\vee$ operate pointwise on $\{0,1\}^n$. We then get:

$$\beta_k : \{0,1\}^n \to \{0,1\}^n$$
$$\beta_k(X) = X \vee t_k(AX)$$

Where AX denotes matrix multiplication. This multiplication yields a vector storing for each vertex how many of its neighbors are in X. Thus, we see 3 basic operations: a matrix-multiplication, a thresholding, and a logical or.

We can thus compute bootstrap percolation by the following naive recurrent sequence $[X_i] \in [\{0,1\}^n]$

$$X_{i+1} = X_i \vee t_k(AX_i)$$

Which we iterate until we reach a fixed point. We are guaranteed to reach a fixed point because X_i is increasing and bounded.

If we were to implement an algorithm based on this, it is important to use the sparsity of A. For A contains $n \times n$ entries, of which at most $d \times n$ are nonzero. (taking d to be the maximal degree of any node in V). Notably, d is constant whilst we take $n \to \infty$. However, we are better off storing the vector X_i as a full vector. For if we have percolation, X_i will eventually contain very few 0s.

We can improve on the naive algorithm by memoizing the calculation of AX_i. This is based on the increasingness of $[X_i]$. We create a new sequence of the memoized matrix multiplications $[Y_i] \in [\mathbb{N}^n]$ with $Y_i \subseteq \mathbb{N}^n$. This vector Y_i stores how many neighbors of a vertex are active. We also define a sequence $[\Delta_i] \in [\{0,1\}^n]$ storing the points added to X_i:

$$Y_i = AX_i$$
$$\Delta_0 = X_0$$
$$\Delta_{i+1} = X_{i+1} - X_i$$

We can then derive the following expression:

$$Y_{i+1} = AX_{i+1} = AX_i + A(X_{i+1} - X_i) = Y_i + A\Delta_{i+1}$$

This allows us to derive the following recurrent expressions for calculating X_i:

$$\begin{aligned} X_{i+1} &= X_i \vee t_k(Y_i) \\ \Delta_{i+1} &= X_{i+1} - X_i \\ Y_{i+1} &= Y_i + A\Delta_{i+1} \end{aligned} \tag{3.1.1}$$

Note that here, we can store Δ_i as a sparse vector. Because, unlike X_i, Δ_i does not have an increasing number of nonzero entries. Then, computing Y_i becomes a sparse-vector sparse-matrix multiplication, which is a lot faster than the full-vector sparse-matrix multiplication we had with the previous approach. An easy way to analyze the resulting speed up is to consider how often we need to query the neighbors of a point (i.e. a column of matrix A). In the previous approach, we need the neighbors of a point at every step after the point was activated. But in the new formulation we only need the neighbors of an active point once, at the step after it was activated.

This is a step in the right direction, but there is more to be gained. It turns out there is a lot of duplicated information between X_i and Y_i. We have the following lemma:
Lemma 3.1.1. *With X_i and Y_i as defined above we have:*

$$X_i = X_0 \vee t_k(Y_i)$$

Proof. The case $i = 0$ follows from the first line of Equation 3.1.1. We then proceed by induction. So suppose the lemma holds for i. Then for X_{i+1} we have the following, by again using Equation 3.1.1.

$$X_{i+1} = X_i \vee t_k(Y_i) = X_0 \vee t_k(Y_{i-1}) \vee t_k(Y_i)$$

Now, note that the sequence $[Y_i]$ is increasing and t_k is an increasing function. Thus we have:

$$t_k(Y_{i-1}) \vee t_k(Y_i) = t_k(Y_i)$$ ■

Paraphrasing the lemma: a node is active if and only if it had more than k active neighbors in the previous step, or it was active in the initial condition.

To make this even simpler, we define a new sequence $[Z_i] \in [\mathbb{N}^n]$ that is just $[Y_i]$ plus a constant vector.

$$Z_i = Y_i + kX_0$$

where kX_0 just denotes scalar multiplication by k. By Lemma 3.1.1 and definition of t_k and $\vee$ we then get:

$$X_i = t_k(Z_i) \tag{3.1.2}$$

This is the basis of our final recurrent form:

$$\begin{aligned} Z_0 &= kX_0 \\ \Delta_0 &= X_0 \\ Z_{i+1} &= Z_i + A\Delta_i \\ \Delta_{i+1} &= t_k(Z_{i+1}) - t_k(Z_i) \end{aligned} \tag{3.1.3}$$

If we ever need X_i we can simply use Equation 3.1.2 to derive it from Z_i.

Note that, since we only ever need a column of A once, it does not make sense to actually pre-compute the matrix if we can just generate the columns on the fly. In practice we constructed all our adjacency matrices one column at a time. This makes it rather easy to simply compute these columns on the fly. A downside is that this fixes our enumeration of V to one in which it is easy to compute the neighbors on the fly. This might not be the enumeration that is best for performance. Because the enumeration chosen affects the memory-access pattern of the algorithm. Due to modern CPU caching, accessing memory 'close' to previously accessed memory is much faster. Thus, it might help for the enumeration to give nodes that are close in the graph indices that do not differ too much. When implementing an algorithm, care could then be taken to try and keep subsequent access be to nodes that are 'close'.

We started with an implementation based on Equation 3.1.1. This actually stored Δ_i as a full vector for the benefit of sequential memory access and stored the matrix A as a sparse CSC matrix. The final implementation, based on Equation 3.1.3, was a massive improvement in running time, and a smaller but still very significant improvement in memory use.

Now based on our final recurrent form (3.1.3) we get the following algorithm in pseudo-code. This algorithm is based around lists, which comes with some notation. By A^* we mean lists with elements

from A. To construct a list, we have a function *list* that takes 0 or 1 arguments and returns a list containing its arguments. We denote list concatenation using .. as an infix operator. That is, given two lists $X, Y \in A^*$ we define the $X..Y$ to be Y concatenated to X. Finally, the function `length`: $A^* \to \mathbb{N}$ returns the length of a list.

Algorithm 1: BootstrapPercolation

```
define BootstrapPrecolation(X_0, k, neighbs)
    input:  X_0 ∈ {0,1}^n
            # vector of point states in initial condition
        k ∈ ℕ
            # the threshold
        neighbs: ℕ → ℕ*
            # given the index of a node, this returns a list of
            # indices of the neighboring nodes.
output: numActive  # Final number of nodes active
        stepsTaken # The number of steps taken

    declare counts∈ ℕ^n
    declare Δ ⊆ ℕ
    declare newCounts∈ ℕ*
    declare stepCount ∈ ℕ

    Δ ← ∅
    stepCount ← 0
    counts ← kX_0 # scalar vector multiplication is pointwise
    forall vertexIdx ∈ 0 ... (n − 1)
        if X_0[vertexId] = 1
            Δ ← Δ ∪ {vertexIdx}

    declare occupied ∈ ℕ
    declare lastOccupied ∈ ℕ
    occupied ← length(Δ)
    lastOccupied ← 0

    while occupied ≠ lastOccupied

        newCounts ← list()
        forall newVertex ∈ Δ
            newCounts ← newCounts..neighbs(newVertex)

        Δ ← ∅
        forall newNeighbor ∈ newCounts
            counts[newNeighbor] ← counts[newNeighbor] + 1
            if counts[newNeighbor] = k
                Δ ← Δ ∪ {newNeighbor}

        lastOccupied ← occupied
        occupied ← occupied +#Δ

        stepCount ← stepCount + 1

    return (numActive : occupied, stepsTaken : stepCount)
```

Here, Δ and `counts` from the algorithm correspond to Δ_i and Z_i from Equation 3.1.3. The variable `newCounts` is an intermediate value storing $A\Delta_i$. Note that we are storing `newCounts` as a multi-set using a list. This list contains all indices of non-zero elements. The multiplicity of the index encodes the value at that index. So the vector $(1, 0, 0, 4)$ could be encoded as $[0, 3, 3, 3, 3]$ or $[3, 0, 3, 3, 3]$. This means that incrementing an entry in `newCounts` is done by simply appending the appropriate index to the list. The vector `counts` is stored as a full vector because in the case of percolation, this vector contains increasingly many nonzero elements.

This algorithm is easy to parallelize in a shared memory model. The parallelized version of the algorithm is below. Notationally, we introduce new keywords prefixed by `||:` that deal with the parallelization. Of these, there are three. `||:parallel` is a keyword that modifies the subsequent for loop to run in parallel. This keyword takes certain variables as arguments. These variables are shared among the threads, so one thread's modifications are visible to the other threads. For all other variables, each thread gets a local copy. Then we have `||:Barrier`. This operation waits until all other threads have hit this barrier, and only then continues. This is used for synchronizing the threads. Finally, we have the `||:atomic` modifier to a declaration. This modifier signifies that increments to these variables are done atomically. For an array, this applies to each element individually. Thus, one can concurrently increment two distinct elements of the array. We then get the following algorithm.

Besides adding the parallel features, the only difference between this algorithm and Algorithm 3.1 is termination of the outer loop. Due to synchronization requirements, we change to a `while true` with a conditional `break` statement.

Algorithm 2: BootstrapPercolationParallel

```
1  define BootstrapPrecolationParallel(X_0, k, neighbs)
2  input:  X_0 ∈ {0,1}^n
3              # vector of point states in initial condition
4          k ∈ ℕ
5              # the threshold
6          neighbs: ℕ → ℕ*
7              # given the index of a node, this returns a list of
8              # indices of the neighboring nodes.
9  output: numActive  # Final number of nodes active
10         stepsTaken # The number of steps taken
11
12     declare counts ∈ ℕ^n ||:atomic
13     declare Δ ⊆ ℕ
14     declare newCounts∈ ℕ*
15     declare stepCount ∈ ℕ
16
17     Δ ← ∅
18     stepCount ← 0
19     counts ← kX_0 # scalar vector multiplication is pointwise
20     forall vertexIdx ∈ 0 ... (n − 1)
21         if X_0[vertexId] = 1
22             Δ ← Δ ∪ {vertexIdx}
23
24     declare occupied ∈ ℕ ||:atomic
25     declare lastOccupied ∈ ℕ
26     occupied ← length(Δ)
27     lastOccupied ← 0
28
29     ||:parallel(counts, occupied)
30     while true
31         ||:barrier
32         if occupied = lastOccupied
33             break
34         lastOccupied ← occupied
35         stepCount ← stepCount + 1
36         ||:barrier
37
38         newCounts ← list()
39         forall newVertex ∈ Δ
40             newCounts ← newCounts..neighbs(newVertex)
41
42         Δ ← ∅
43         forall newNeighbor ∈ newCounts
44             counts[newNeighbor] ← counts[newNeighbor] + 1
45             if counts[newNeighbor] = k
```

```
46                  Δ ← Δ ∪ {newNeighbor}
47
48          occupied ← occupied + #Δ
49
50      return (numActive : occupied, stepsTaken : stepCount)
```

We see that each thread has shared access to the array `counts` and the value `occupied`. Not coincidentally, these are the two variables that are made atomic. Beyond that, we have added two `||:Barrier`s. To see that this algorithm remains correct, consider the lists `newCounts` and Δ in the non-parallel case at any step. Next, consider the concatenation of all local versions of those lists at the same step. The concatenated lists contain the same elements as the lists from the non-parallel case. Thus `counts` is the same at each step, and so is `occupied`.

Due to the first barrier, all threads are always running the same step of bootstrap percolation. Moreover, the two barriers combined ensure that we check `occupied` against `lastOccupied` only after all threads have properly updated `occupied`. Finally, the barriers also ensure that no other thread modifies `occupied` before we save it in `lastOccupied`.

In the parallel algorithm we get some indeterminism when reading from a shared variable. This is relevant when we read from `counts`. For `occupied` this is not an issue because during the reads we have either atomicity, or synchronization via `barrier` to prevent indeterminism. As explained, this indeterminism does not affect the correctness of the algorithm. However, it does have an effect on performance, for it might lead to unbalanced work amongst the threads. It might happen that one thread gets a really big Δ (and thus `newCounts`) whereas other threads have very little. Due to the synchronization of the while loop, this leads to the threads with less work waiting on the other threads.

This could be solved by rebalancing the vector Δ. That is, by moving a part of a large local version Δ to another, small, local version of Δ. Many different versions of rebalancing is possible. One might rebalance every step, every few steps, or only when the difference are very big (either absolutely or relatively). The downside to rebalancing is constant overhead and the rather complex interaction between threads that is needed. In practice, it would require extensive benchmarking to determine which method, if any, is better. Due to time limitations we did not do this benchmarking.

One might note that in our algorithm the list `newCounts` is redundant. We could merge the first and second for loops in the main loop, essentially dealing with each new neighbor as it comes up, rather than first storing it. We chose not to do this for reasons of extensibility. For example, the above mentioned rebalancing requires the list `newCounts`. It is also very useful if one wants to create a distributed version of the algorithm, as will be explained in Section 3.1.2.

3.1.1 Implementation

We implemented Algorithm 3.2 in C++. The code can be found in Appendix A.

We used a `std::vector` to store Δ and `newCounts`. The type `std::vector` is a dynamically resized array. This is quite fast for appending values, which is constant time unless the array needs to be resized. It is also optimal for iterating over all values, because that is just sequential memory access. The multi-threading and corresponding synchronization and variable sharing was done using OpenMP. For atomic access, we made use of the atomic intrinsic in C++. This compiles to code using assembly-level lock instructions. This is a lot faster than doing the required synchronization at a higher level. Specifically, the array `counts` was an array of atomic `char`s, and the variable `occupied` was an atomic 64-bit integer.

Now, let us consider the memory usage of our implementation. Here, we take N to be the number of nodes in our graph, K to be the maximal degree of our graph and D to be the initial density.

For the array `counts` we simply need N bytes. Next, we have the list Δ. This is a list of 64-bit integers. We cannot use 32-bit integers because we want to handle cases where $N > 2^{32}$. Due to the dynamic resizing, we can have at most a factor 2 between the amount of elements in the list, and the amount of memory reserved. In this analysis, we ignore that factor. Now, we know the list contains DN elements in the first iteration. For subsequent iterations, the exact value is hard to qualify. In practice, we sometimes

see this list grow significantly, though it never seems to exceed $2DN$. Finally, we have the list `newCounts`. Presuming that the border is small, we can approximate every node in Δ to have degree K. Thus for every element in Δ we have approximately K elements in `newCounts`. This yields a lower bound of:

$$N + 8\left(DN + KDN\right) = \left(1 + 8 \times (K+1) \times D\right) N \tag{3.1.4}$$

and an upper bound based on our observations on the size of Δ and the potential factor 2 overhead of our dynamic arrays of:

$$2\left(1 + 8 \times (K+1) \times 2D\right) N$$

Notably, we see that the initial density matters quite a bit. For graphs with degree 4, the density has a multiplier of at least 40 and at most 160. So at a density of only 0.025 we would already use more memory to store the 'sparse' vectors than we use to store the full vector `counts`. However, it should be noted that this is about peak memory usage. For computations with a very long tail of steps with minimal growth, the sparsity becomes very useful.

3.1.2 Potential distributed algorithm

Our parallelized Algorithm 3.2 depends on shared memory. This means the algorithm is limited by the amount of memory available to any single machine. In practice, we had access to machines with up to 2TB of RAM. This gives an upper bound of around 2^{39} nodes in our graphs (based on Equation 3.1.4 with $K = 4$ and $D \approx 0.1$). In practice, this was the limiting factor to our simulations.

To get past this limit, we would need to go to a distributed algorithm. That is, one that is run on multiple different computers (as opposed to multiple different threads). A Distributed global address space is the obvious way to go about this. In essence, this transparently joins the memory space of multiple computers into a single address space, and allows it to be accessed as if it were contiguous local memory. However, this comes with the disadvantage of much more memory latency. For random access, the difference is multiple orders of magnitude. This is unacceptable for our use case.

As such, we can no longer store the `counts` array in shared memory. We suggest the following alternative method for storing and modifying `counts`:

- Each server is responsible for a specific subset of vertices of the graph. For these vertices, it stores `counts` and Δ.
- Each server computes `newCounts` based on Δ as in the normal algorithm.
- When `newCounts` contains vertices for which the current server is not responsible, the current server communicates this to the correct server.
- Servers calculate new Δ based on their local `newCounts` *and* based on received communications of other servers' `newCounts`.

The big question is how the servers communicate changes in `newCounts`. This could be done synchronously, at the end of every step. However, that comes with significant overhead. Moreover, we incur the full cost of latency every step, and every step takes as long as the slowest server.

Luckily, we do not actually need to synchronize the loops. By giving up synchronization we can no longer get an accurate count of how many steps of β_S were needed to reach the final state, but that is not very important. However, there is another downside. It becomes very hard to determine when we have actually reached the final state. For example, it is possible that each server has no nodes in either Δ or `newCounts`, but there are still messages about new nodes in `newCounts` in flight.

Beyond the stopping problem, just stating 'we send messages asynchronously' is not specific. Instead, we need a way to batch messages to keep down overhead. However, batching too much might leave servers idling because they are waiting for a batch update. There is no obvious right way to batch, and we expect getting an acceptable level of performance to require much trial and error. For these reasons, we did not design a proper distributed algorithm. We did write and run a synchronized distributed algorithm using the MPI framework, but performance was abysmal.

3.2 Tested graphs

We did our experiments on Cayley graphs of two classes of amenable groups: Heisenberg groups and lamplighter groups. As a benchmark, we did the same experiments on abelian graphs. For our experiments, the way we choose the finite subgraphs matters. It is easy to find 'adversarial' examples of ever growing subgraphs with very low probability of percolation. In general, since the border of our subgraph is where we differ locally from the full graph, we wish to minimize this border. Recall that given a graph (V, E) and a subgraph $F \subseteq V$ we defined the border of F as:

$$\delta(F) = E \cap (F \times (V \setminus F))$$

The size of the border is the cardinality of the above set.

As our graphs are amenable, they admit Følner sequences (see Definition 1.1.1). These are guaranteed to have a vanishingly small border. That is by definition of a Følner sequence, if $[F_i]$ is a Følner sequence we have:

$$\lim_{i\to\infty} \frac{\#\delta(F_i)}{\#F_i} = 1$$

As such, we will let these sequences guide our choice of subgraph.

Below, we treat each class of groups separately. For both, we first define the group and give some exposition as to why we chose it. Then, we move on to its Følner sequences, and the specific subgraphs we chose in the end. Finally, we discuss locally similar finite groups that converge to the full group.

3.2.1 The Heisenberg group

The Heisenberg group is the group of upper triangular 3×3 matrices under multiplication over some ring. In our case, we looked at all upper triangular matrices over $\mathbb{Z}$. That is, all matrices of the form:

$$x, y, z \in \mathbb{Z}$$
$$\begin{pmatrix} 1 & x & z \\ 0 & 1 & y \\ 0 & 0 & 1 \end{pmatrix}$$

We will often denote an element of this group by the tuple (x, y, z). In this notation, the group operation is as follows:

$$(x, y, z) \circ (x', y', z') = (x + x', y + y', z + z' + xy')$$

We see that this is almost an operation group except for the third coordinate. This is part of the reason why we chose this group, because it is close to an abelian group, and we have theoretical results about abelian groups. Specifically, we know from [Sch92] that abelian groups with canonical generators have $p_c = 0$. Moreover, we know from [Bal+12] that cubes of size L in the abelian group $\mathbb{Z}^n$ have a critical point at $O\left(\frac{1}{\log^n L}\right)$ (see Equation 1.1.1). Another reason for choosing the Heisenberg group is because it is one of the simplest finitely generated infinite amenable groups.

The group can be generated by $(1, 0, 0)$ and $(0, 1, 0)$. These are the generators we use for our Cayley graphs. This means the four neighbors of a point (x, y, z) are:

$$\begin{aligned} &(x + 1, y, z)\ , \quad (x, y + 1, z + x)\ , \\ &(x - 1, y, z)\ , \quad (x, y - 1, z - x) \end{aligned} \tag{3.2.1}$$

Easy to describe subsets of this group are the equivalents of boxes in $\mathbb{Z}^3$. By a box with dimensions (a, b, c) we mean all elements corresponding to tuples of the form:

$$(x, y, z) \text{ such that } |x| \leq \frac{a}{2} \wedge |y| \leq \frac{b}{2} \wedge |z| \leq \frac{c}{2} \tag{3.2.2}$$

We then have an easy Følner sequence indexed by i in boxes with dimensions (i, i, i^2). Note that simple cubes, or boxes with fixed proportions, will not work. To see this, suppose we take boxes of the form (i, ai, bi) for $a, b \in \mathbb{R}$ (ignoring rounding). Then the element $(0, b, 0)$ is a counter-example to this being a Følner sequence. We really need the z dimension to grow asymptotically faster than the other dimensions.

In choosing our subgraph, we do not just want any Følner sequence, we want one that minimizes the border of our box for our specific generators. Now given a box with dimensions (x, y, z) it is easy to calculate the size of the border. Our box has 6 faces, we consider 1 face in each direction and see which edges cross these faces.

For the xz face and the yz face, if we look at Equation 3.2.1 we see neighbors only ever differ by 1 unit in the y or z direction respectively. Thus, we have $x \times z$ elements for the xz face and similarly $y \times z$ for the yz face.

The xy face is a bit trickier. A point (x', y', z') moves x' units in the z direction. As such, we cross the xy face when $z' + x' > z$. This occurs for $\frac{y}{4}x^2$ points. Note that we have now counted some edges twice. Some edges cross both the xy face and the xz face. We can exclude these duplicates from our count which yields $\frac{y-2}{4}x^2$ points. The error caused by including the duplicates is small, and including the duplicates gives an easier expression. Thus we get a total border size of:

$$xz + yz + \frac{y-2}{4}x^2 \approx xz + yz + \frac{y}{4}x^2$$

If we fix the total number of nodes $x \times y \times z = N$, we find a minimum of the approximate border at a box with dimensions:

$$\left(\sqrt[4]{N},\, 2\sqrt[4]{N},\, \tfrac{1}{2}\sqrt{N}\right) \tag{3.2.3}$$

These are not guaranteed to be integers, we just round and consider the errors introduced by this rounding small enough to ignore.

If we want to eliminate the border by taking locally similar finite groups, we can simply take the matrix over the ring $\mathbb{Z}/n\mathbb{Z}$. This is actually isomorphic to a quotient group of our full Heisenberg group. This follows from the simple mapping $(x, y, z) \mapsto (x \mod n, y \mod n, z \mod n)$ being a group-homomorphism. The corresponding kernel is generated by $(n, 0, 0)$ and $(0, n, 0)$.

3.2.2 Lamplighter groups

The standard lamplighter group (also known as the binary lamplighter group) can be seen as all possible states of a Turing machine tape, including the head position. That is, the elements can be encoded as (n, T) where $n \in \mathbb{Z}$ and $T : \mathbb{Z} \to \mathbb{Z}/2\mathbb{Z}$. Here n encodes the position of the head over the tape, and T encodes the tape itself. The identity element is $(0, x \mapsto 0)$, that is, the empty tape with the head at position 0. Finally, the group operation is:

$$(n, T) \circ (n', T') = (n + n', x \mapsto T(x) + T'(x + n + n'))$$

note that because the addition $T(x) + T(y)$ above happens in the codomain of T here, that is $\mathbb{Z}/2\mathbb{Z}$. Later, we will see cases where T has a different codomain. The group is generated by the following two elements:

$$(1, x \mapsto 0)\,, \quad (0, \mathbb{1}_{\{0\}})$$

where $\mathbb{1}_A$ is the indicator function of the set A. These two elements can be seen as 'move the head right' and 'toggle the bit under the head' respectively. The group operation can then be interpreted as follows. An element (n, T) of the group represents a combination of head movements and bit flips that leads to tape state (n, T) from an empty tape. Applying (n', T') to (n, T) is then applying the head movements and bit flips for (n', T') only from the starting tape (n, T) instead of from an empty tape.

One can generalize this from a binary tape to a tape with N values per position. Essentially all that changes is that elements on the tape are then of the form (n, T) with $n \in \mathbb{Z}$ and $T : \mathbb{Z} \mapsto \mathbb{Z}/N\mathbb{Z}$. We have the same expression for our group operation and the same generators. Only now the addition $T(x) + T(y)$ happens in $\mathbb{Z}/N\mathbb{Z}$.

For the lamplighter group, a Følner sequence indexed by i is:

$$\{(n, T) \mid |n| \leq i \wedge \forall m \in \mathbb{Z} : T(m) \neq 0 \rightarrow |m| \leq i\}$$

That is, instead of taking a bi-infinite tape we take a tape that has positions from i to $-i$. It is somewhat surprising that the lamplighter group admits a Følner sequence, because it has an exponential growth rate. That is, if we take the sequence of sets indexed by i:

$$\{x \mid \text{path length to } x \text{ from the origin} \leq i\}$$

where we count path length based on a Cayley graph. Then the cardinality of those sets grows exponentially in i. Most amenable groups have a polynomial growth rate while all non-amenable groups have an exponential growth rate. For example, the Heisenberg group has growth rate of order $O(i^4)$. This makes the lamplighter group an edge case as it is 'almost non-amenable'.

For our Cayley graphs of the lamplighter group, we do not want periodic generators. Thus, we do not take the generators given above, but we take:

$$(1, x \mapsto 0), \quad (1, \mathbb{1}_{\{0\}})$$

That is, we add a head-shift to the generator that flips the bit under the head. For our subgraphs, we take something very close to the Følner sequences above, but also add tapes that have positions from $i - 1$ to $-i$. This allows us to have tapes of odd length, doubling the number of subgraphs we can examine. This is important because a tape with length L has $L2^L$ nodes (presuming a binary tape). This very quick growth means there are not a lot of tape-lengths we can actually run computations for due to memory limitations. For such a subset with tape-length L the border has size 4×2^L (again, presuming a binary tape). To see this, consider that the only way to leave the sub-graph via a generator is by moving the head position. If the head is at one end, the two generators that increment the head position exit the subgraph. At the other end, the inverses of those generators exit the subgraph.

If we want to remove the border by taking locally similar finite groups, we can take a 'wrapped tape'. For a tape-length L an element of our group is then: (n, T) with $n \in \mathbb{Z}/L\mathbb{Z}$ and $T : \mathbb{Z}/L\mathbb{Z} \rightarrow \mathbb{Z}/2\mathbb{Z}$. The group operation remains the same, noting that the additions involving n now happen modulo L. This is a quotient group of the full lamplighter group. To see this, consider the homomorphism:

$$(n, T) \mapsto \left(n \bmod L, x \mapsto \sum \{T(y) \mid y \bmod L = x\}\right)$$

3.3 Results

As described in the previous section, we now have interesting sequences of increasing finite graphs, and know how to compute bootstrap percolation on these graphs. Given such a finite subgraph G, we are interested in the relation between the initial density p and the expected final density, denoted by B_G. Formally, we define the function $B_G : [0, 1] \rightarrow [0, 1]$ as:

$$B_G(p) = \mathbb{E}_p \left(\frac{\#\left(\beta_k{}^{\infty}(A_0)\right)}{\#G} \right)$$

Where A_0 is an initial condition with density p. That is, for any vertex $v \in G$ we have $\mathbb{P}(v \in A_0) = p$ and this is independent of any other vertices in G. Our goal is to approximate the function B_G for various subgraphs G. We can easily derive the following properties of B_G regardless of G.

$$\begin{gathered} B_G(0) = 0 \\ B_G(1) = 1 \\ B_G(p) \geq p \\ B_G \text{ is an increasing function} \end{gathered} \tag{3.3.1}$$

Besides the final density, we are also interested in the *density growth*. Given an initial density p and a final density f, this is defined as:

$$\frac{f-p}{1-p}$$

The advantage of the density growth is that no growth occurring shows a density growth of 0, whereas the final density is p. We divide by $1-p$ so that occupying all nodes gives a density growth of 1. The density growth gives the fraction of inactive nodes that were activated. We also define the expected final density:

$$B'_G(p) = \frac{B_G(p)-p}{1-p}$$

Note that this is also an increasing function. To see this, let A be an initial condition with density p. Then, take $B = {\beta_k}^\infty(A)$. Now, we mean to create an initial condition A' with density $p+q$ by extending A. To do this, we take $\Delta \subseteq V \setminus A$ with density $\frac{q}{1-p}$. We then set:

$$A' = A \cup \Delta$$
$$B' = {\beta_k}^\infty(A') \supseteq B \cup \Delta$$

That last inclusion follows from β_k being an increasing and extensive function. Then, since Δ was chosen uniformly, we have the following:

$$\mathbb{E}\left(\frac{\#B' \setminus A'}{\#V \setminus A'}\right) \geq \mathbb{E}\left(\frac{\#B \setminus (A \cup \Delta)}{\#V \setminus (A \cup \Delta)}\right) = \mathbb{E}\left(\frac{\#B \setminus A}{\#V \setminus A}\right)$$

What we expect to see is B'_G showing critical behavior. That is, there is some density q_G called the *critical density*. For $p << q_G$ we would have $B'_G(p)$ take a low value. Similarly, for $p >> q_G$ we would have $B'_G(p) \approx 1$. Finally, for $p \approx q_G$ we expect B'_G to rise rapidly. For these $p \approx q_G$ we say G undergoes a phase-transition. Note that this critical point q_G is a much looser concept than the tightly defined p_c (in Equation 1.0.3. We are then interested how q_G behaves when we take G through our well-chosen increasing sequences of subgraphs.

To approximate $B'_G(p)$ for a given p and G we simply generate an initial condition A_0 (pseudo-)randomly multiple times, then run the bootstrap percolation algorithm and record the final density. The mean of these final densities then approximates $B'_G(p)$. Given a subgraph G, we repeat this procedure for various p until we believe we have a good view of $B'_G(p)$ based on Equation 3.3.1. Then, using our approximate B'_G we derive the critical point q_G. We repeat this for ever increasing subgraphs G until the computations start taking too much resources.

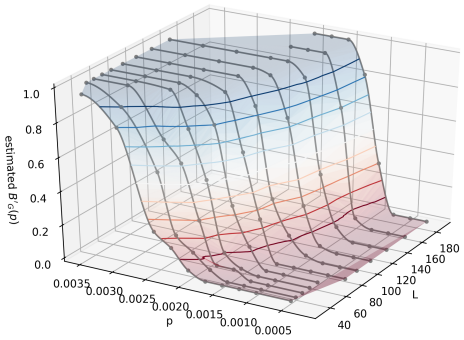
When interpolating our estimated points of B'_G we use a piecewise monotonic cubic spline. We want smoother interpolation that linear, thus we pick a spline. However, a normal spline can 'overshoot'. This might yield interpolated values of B'_G outside $[0,1]$. By using piecewise monotonic splines, this is avoided. The monotonicity is also expected because B_G is an increasing function. So it is very unlikely that B'_G is not monotonic. Moreover, the possible local-decreasingness of B'_G is very limited. As such, if our approximations for B'_G are locally decreasing we take that to be a statistical aberration. Where computationally feasible, when we see such an aberration, we smooth it over by increasing the total number of runs for subgraph G at initial density p where the aberration occurs.

3.3.1 Abelian 4D

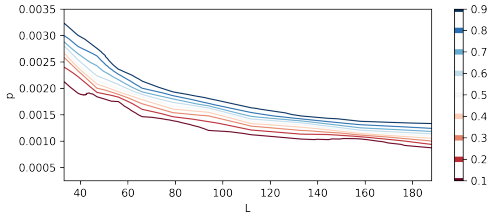
As a benchmark, we first apply our method to the group $\mathbb{Z}^4$ using addition. From theory, we already know that $q_G = O((\ln\ln\ln L)^{-1})$. Our subgraphs were simple cubes of side length L. For the sake of resource conservation, we did not push L as far as possible here. This limited L to 188, with a corresponding maximal memory consumption of only about 3.2GB.

As theory predicts, we see critical behavior, with the critical point slowly decreasing. In Figure 3.2 we plot an estimate of q_G by interpolating B'_G and finding the p where $B'_G(p) \approx 0.5$. Here we can see that

$$q_G \approx \frac{0.000606}{\ln\ln\ln L}$$



(3.1-a) 3D view of B'_G. The gray lines represent interpolated estimates of $B'_G(p)$. The dots on these lines represent the points through which we interpolate.



(3.1-b) Contour lines of the graph to the left, plotted from above.

Figure 3.1: Estimated $B'_G(p)$ for cubes of length L in the group $\mathbb{Z}^4$. Each point is estimated by at least 20 runs.

which is congruent with what theory predicts.

When looking at individual runs, we see an interesting dichotomy. A given initial condition either percolates or has no growth at all. This is evident in Figure 3.3. This is a clear form of critical behavior.

3.3.2 Heisenberg

For the Heisenberg group, we looked at subgraphs based on boxes as described in Equation 3.2.2. We picked boxes with the following dimensions:

$$\left(L, 2L, \frac{L^2}{2}\right) \tag{3.3.2}$$

Here we call L the side length of a box. This choice was based on Equation 3.2.3, which minimizes the border of our subgraph. We look at a sequence of increasing values of L. Every step, we increased L

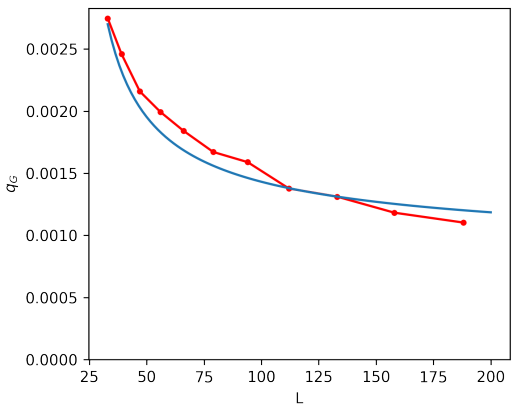


Figure 3.2: The estimation of q_G as a function of L the side length of G (in red). And the function: $L \mapsto \frac{0.000606}{\ln \ln \ln L}$ (in blue)

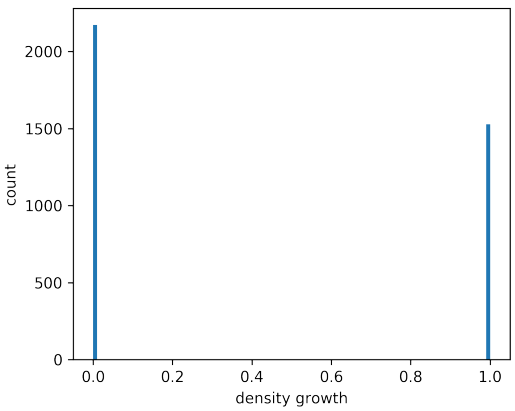
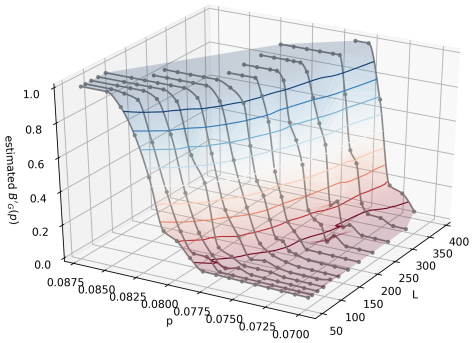
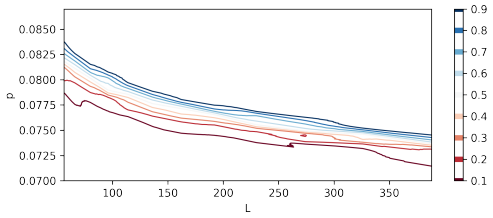


Figure 3.3: Distribution of the final density for all individual runs on $\mathbb{Z}^4$. Note that there is not a single run with a final density in the range 0.02 to 0.98.



(3.4-a) 3D view of B'_G. The gray lines represent interpolated estimates of $B'_G(p)$. The dots on these lines represent the points through which we interpolate.



(3.4-b) Contour lines of the graph to the left, plotted from above.

Figure 3.4: Estimated $B'_G(p)$ for various subgraphs G of the Heisenberg group indexed by the side length L. At a given side length L the subgraph G is chosen as per (3.3.2). Each point is estimated by at least 20 runs, except for $L = 388$ which has 15 runs.

such that the number of vertices in the graph (and thus the amount of memory required) doubled every step.

In Figure 3.4 we plot the resultant approximation of $B'_G(p)$. We see the 'critical behavior' we expected. That is, there seems to be a rather sharp transition in B'_G. Moreover, the point at which this transition occurs decreases as the size of our subgraph increases. Comparing this to the results for $\mathbb{Z}^4$ we note that the critical point q_G seems to be an order of magnitude higher for the Heisenberg group.

We do see some points where the estimation for B'_G is decreasing. As explained earlier, we treat these as aberrations and tried to correct them. We were not able to correct all of these aberrations by doing more runs due to the computational costs. As it stands, some points were estimated by more than 140 runs to remove aberrations.

Next, we need to derive the critical points q_G based on our approximations for B'_G. As the shape of $B'_G(p)$ seems to be rather symmetrical during the transition, we define the critical point by $B'_G(q_G) = 0.5$. In Figure 3.5 we plot these critical points as a function of the side length of G. It then seems we have:

$$q_G \approx \frac{0.17}{\log_2 \log_2 \log_2 \frac{L^4}{16}}$$

Note that a side length of L corresponds to a subgraph with L^4 vertices, this could explain the term L^4 in the above function. The 3 logs coincide with the results for $\mathbb{Z}^4$. Notably, the Heisenberg group and $\mathbb{Z}^4$ share a quartic growth rate, this might explain the 3 logs. We consider the quality of the fit combined with the above observations weak evidence that indeed we have an approximation of the critical point. This translates to weak evidence that indeed the Heisenberg graph has $p_c = 0$. What remains unexplained is the constant 0.17 and the base 2 for the logarithms. It should be noted that our approximation is still $O((\ln \ln \ln)^{-1})$. It should also be noted that the convergence we claim to have is very slow. Without the correspondence to the abelian case, one would probably conclude this seems to converge to a positive value.

If we look at the outcome of individual runs, we see very clear evidence of critical behavior for individual runs. In Figure 3.6 we plot the distribution of all final densities of individual runs. We see that any given run either finished with almost no new points added, or with nearly the entire graph occupied. It would be very interesting to know why it is so unlikely to add only an intermediary fraction of our subgraph. As per Lemma 2.1.2 this informs us about the kind of k-forts we see in our finite subgraphs.

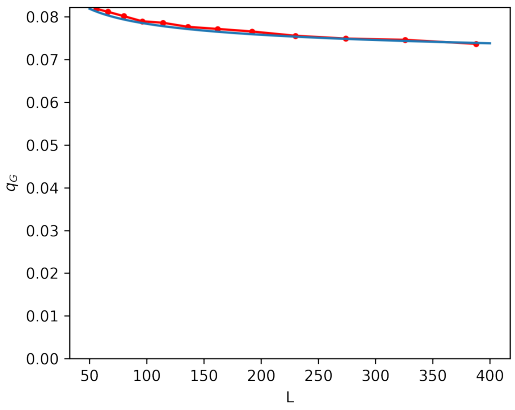


Figure 3.5: The estimation of q_G as a function of L the side length of G (in red), and the function $L \mapsto \frac{0.17}{\log_2 \log_2 \log_2 \frac{L^4}{16}}$ (in blue).

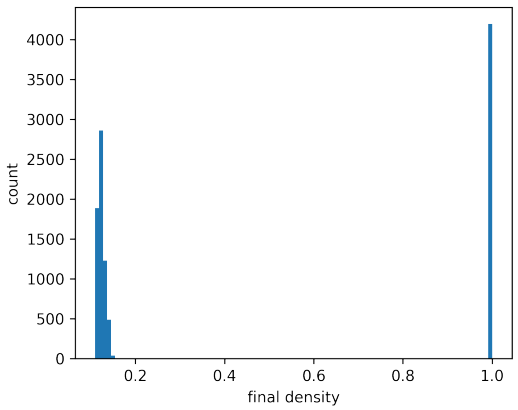


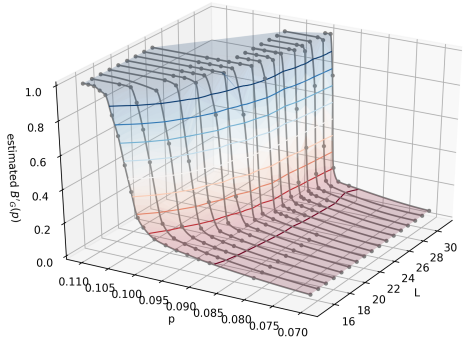
Figure 3.6: Distribution of the final density for all individual runs on Heisenberg graphs. Note that there is not a single run with a final density in the range 0.1 to 0.95.

3.3.3 Lamplighter

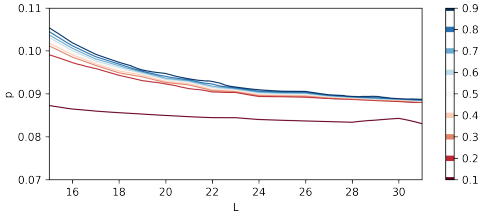
For the lamplighter group, we pick our subgraphs on the basis of a tape length L as described in Section 3.2.2. We started at a tape length of 15, and incremented the tape length as RAM permitted. In Figure 3.7 we plotted the resulting approximations of B'_G.

Again we see 'critical behavior'. However, for p below the critical point we do not have $B'_G(p) \approx 0$. Instead, it $B'_G(p)$ rises slowly as p increases, until we get near the critical point. There $B'_G(p)$ ramps up to very quick growth until it levels off sharply at $B'_G(p) = 1$. This slow rise before the critical point is nicely illustrated by the contour lines in Figure 3.7-b especially when looking at the distance between the line $B'_G(p) = 0.1$ and the line $B'_G(p) = 0.2$. Notably, we see a nice smooth and monotonic function. This stands in contrast with the same plots for the Heisenberg group.

Next, we want to derive the critical densities q_G from our approximated function B'_G. Here, we see that B'_G does not behave symmetrically around the critical point. The slow growth and gradual ramp-up



(3.7-a) 3D view of B'_G. The gray lines represent interpolated estimates of $B'_G(p)$. The dots on these lines represent the points through which we interpolate.



(3.7-b) Contour lines of the graph to the left, plotted from above.

Figure 3.7: Estimated $B'_G(p)$ for various subgraphs of the lamplighter group G indexed by their tape-length L. Almost every point was estimated by 20 runs. At $p = 0.7$ at $L = 31$ we only had 5 runs and at $L = 15$ and $L = 16$ some points are estimated by 40 runs due to unintentional duplicate runs.

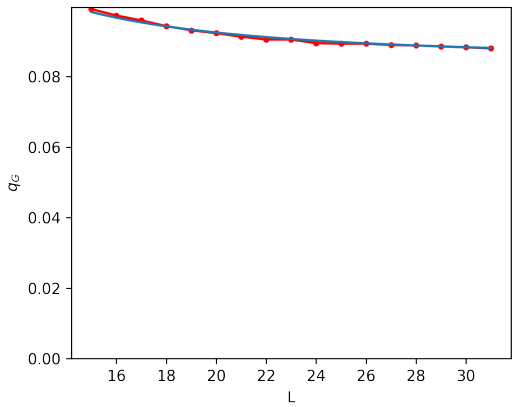


Figure 3.8: The critical density q_G as a function of L, the tape length of G for the normal lamplighter group (in red), and the function $L \mapsto 0.085 + 3L^{-2}$ (in blue).

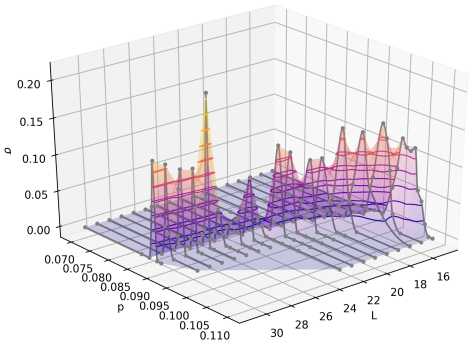
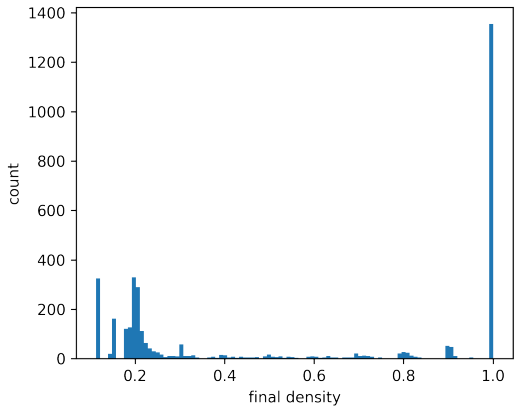


Figure 3.9: The standard deviation of the density growth over all runs for any given combination of p and L for the normal lamplighter group.

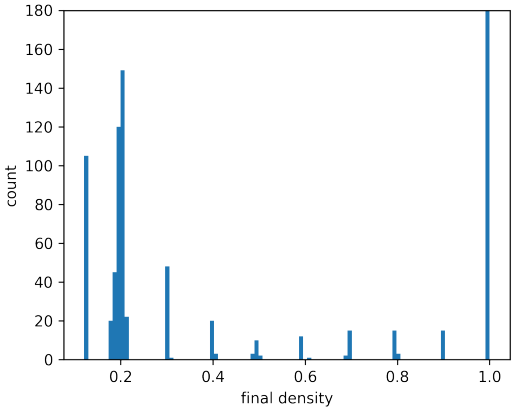
before the critical point are very different from the sharp leveling-off after the critical point. Moreover, the point where we level-off varies much more with the tape length L than the point where growth ramps up. This complicates the decision of how to derive the point q_G from the approximated B'_G. This is a bigger problem for the smaller graphs, because the range in which the phase transition occurs shrinks as the subgraphs become larger. In the end, we decided to say the point of critically lies at $B'_G(p) = 0.2$. This emphasizes the part of B'_G where growth ramps up. In Figure 3.8 we plot the estimated q_G against the tape-length L.

Looking at this graph, we cannot argue for convergence to 0. We could find no 'sensible' function that converges to 0 that fits this graph. That is, we could not fit functions that can be defended on the basis of known results. Instead it seems we have an asymptote at around 0.085. The fitted line in the figure is $L \mapsto 0.085 + 3L^{-2}$ where L is the tape-length. We do not consider this a meaningful fit; we only included it to show the graph seems to have an asymptote above at 0.085.

This does not necessarily mean that $p_c \approx 0.085$ for the lamplighter graph. It might be that convergence is indeed very slow. Moreover, it remains notable that the apparent asymptote is quite low. If the apparent asymptote were to lie at $q_G = 0.5$ that would strongly suggest that $p_C > 0$. However, with

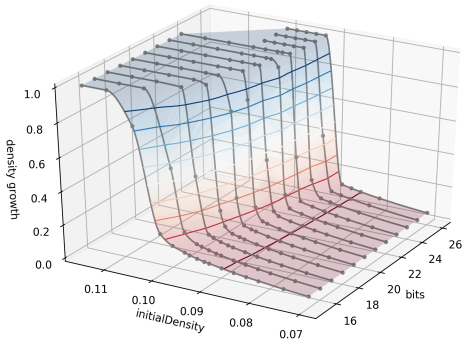


(3.10-a) Distribution of the final density for all runs.

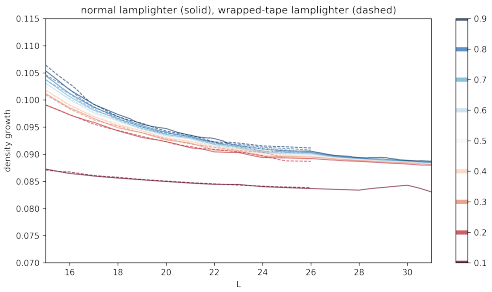


(3.10-b) Distribution of the final density for all runs with $L > 25$. For a density growth of 1.0 the count is clipped, it reaches approximately 400.

Figure 3.10: Distribution of the final density for individual runs on lamplighter graphs



(3.11-a) 3D view of estimated B'_G. The gray lines represent interpolated estimates of $B'_G(p)$. The dots on these lines represent the points through which we interpolate.



(3.11-b) The contours of the estimated B'_G for the wrapped-tape lamplighter group as plotted to the left. Compared to the contours of estimated B'_G for the normal subgraphs of the lamplighter group of the same tape length as plotted in Figure 3.7.

Figure 3.11: Estimated $B'_G(p)$ for Cayley graphs of wrapped-tape lamplighter groups of tape length L.

our current apparent asymptote it seems much more likely that we misestimated the asymptote to be positive.

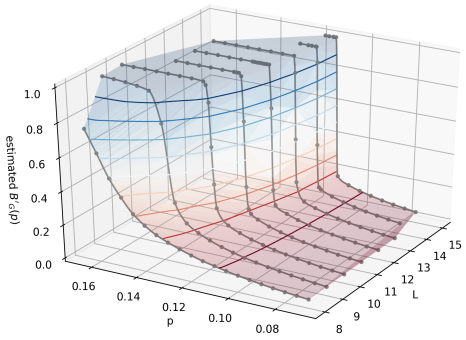
Also note that, the reasoning that gave us $\log \log \log$ convergence for the Heisenberg group was based on the growth rate of the Heisenberg group corresponding to the growth rate of $\mathbb{Z}^4$. If we extrapolate Equation 1.1.1 for a growth rate of $O(x^n)$ we would then expect $O\left(\frac{1}{\log^{n-1} x}\right)$ convergence of q_G. However, the lamplighter group has an exponential growth rate. Note that by $\log^n$ we mean the logarithm applied n times. Thus, the above reasoning would yield $O(\log^{\infty-1} x)$ which is nonsense.

Next, we investigate the behaviour of individual runs. In Figure 3.10-a we plot the distribution of density growth for individual runs. Here we see an interesting difference with the Heisenberg group and $\mathbb{Z}^4$. We have individual runs with final densities that are neither close to 0 or 1. In fact, it seems like the final densities cluster around tenths i.e. around the points $\frac{1}{10}, \frac{2}{10} \ldots \frac{10}{10}$. The relative heights of the peaks here are meaningless; They are strongly affected by the distribution of tape lengths L and initial densities p in our runs. For our case, this distribution is very far from uniform.

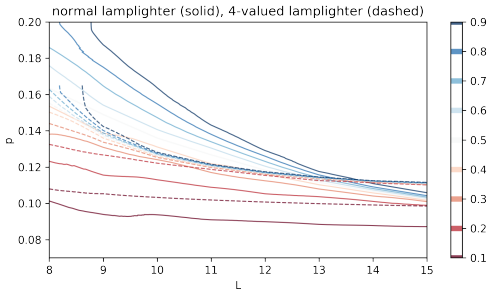
Beyond looking at the distribution of all final densities for all runs, we can also consider the distribution of final densities for a given subgraph G and initial density p. Figure 3.7 plots the mean of these distributions. Now, in Figure 3.9 we plot the standard deviation of these distributions. The peaks in this plot occur during the phase-transition, as is to be expected. Outside that range $B'_G(p)$ is roughly 0 or 1. This means there cannot be much variation. Notably, the peaks of the plot are not very high. This means intermediate values of $B'_G(p)$ occur because most runs yield a final density close to that intermediate value. This explains why the approximation of B'_G here is smoother than it was for the Heisenberg group. However, per Figure 3.10-a we know these intermediate densities are not normally distributed around $B'_G(p)$; instead, they cluster around tenths. This remains mysterious and definitely warrants further investigation.

Next, to study the effects of removing the border, and in the hope of seeing convergence of q_G to zero, we ran simulations on the wrapped-tape lamplighter groups. For the sake of efficiency, we only ran simulations up to a tape-length of $L = 26$. These simulations gave essentially identical results as the normal lamplighter group. In Figure 3.11-b we plot the contours of $B'_G(p)$ for both types of graphs over each other. When we study the distribution of final densities, we again see peaks around tenths for the distribution of final densities for individual runs. As such, we concluded the behavior for the wrapped tape lamplighter groups is essentially the same as for the subgraphs with the same tape length. This concluded our investigation into the wrapped tape lamplighter group.

We also ran simulations on the 4-valued lamplighter group. This essentially allows any position on the tape to have 4 values as opposed to just 2. For a given tape length L, this tape has quadratically more



(3.12-a) 3D view of estimated B'_G. The gray lines represent interpolated estimates of $B'_G(p)$. The dots on these lines represent the points through which we interpolate.

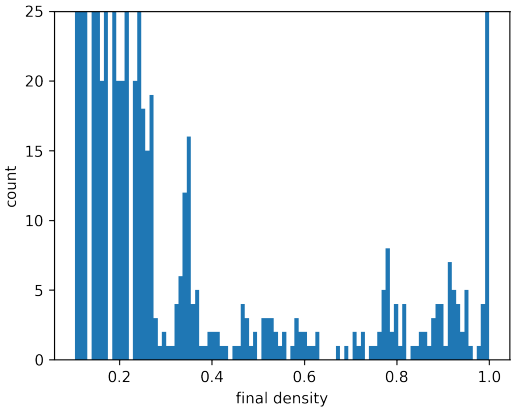


(3.12-b) The contours of the estimated B'_G for the 4-valued lamplighter group as plotted to the left. Compared to the contours of estimated B'_G for the normal subgraphs of the lamplighter group of the same tape length.

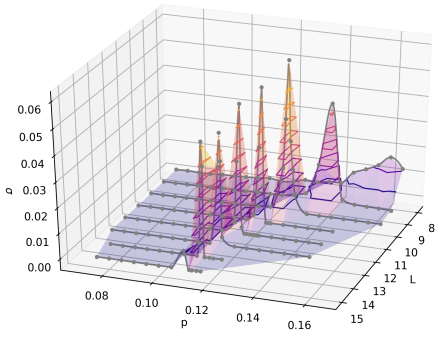
Figure 3.12: Estimated $B'_G(p)$ for subgraphs of the 4-valued lamplighter group indexed by their tape length L.

nodes. ($L \times 4^L$ as opposed to $L \times 2^L$) As such, we could only run simulations for much shorter tapes. The results can be found in Figure 3.12-a. For the sake of comparison we also ran simulations of the normal lamplighter group for the same tape-lengths. In Figure 3.12-b we compare the results for the normal lamplighter group to the 4-valued lamplighter group at the same tape lengths. Here, we see the 4-valued lamplighter group has a quicker phase-transition, with a higher critical point q_G when compared to the normal lamplighter group with the same tape-length. As such, we conclude this graph is less likely to have $q_G \to \infty$ than the normal lamplighter group.

We also looked at the distribution of final densities for the 4-valued lamplighter group. Part of distribution is shown in Figure 3.13-a. Here we no longer see the clustering around tenths. One could make an argument for clustering around fractions of 21, but that is a strained argument at best. Next, in Figure 3.13-b we plot the standard deviation of final densities for any given tape length L and initial density p. We see this standard deviation is really low, with a maximum around 0.06. Thus, intermediate values for $B'_G(p)$ result from essentially all runs given the same final density.



(3.13-a) Distribution of all individual final densities for runs with a tape length $L > 10$, clipped at a count of 25.



(3.13-b) The standard deviation of the density growth for any given combination of p and L for the 4-valued lamplighter group.

Figure 3.13: Distribution of final densities for runs on a 4-valued lamplighter group

Based on our data for the 4-valued lamplighter group, we cannot see whether there are intermediate final densities, and if so whether they cluster around tenths. This is because we have very few simulations that yield these intermediate densities. Due to the very sharp phase-transition, there is a very small range of initial densities that would produce these intermediate densities.

We did not study any other n-valued lamplighter groups. It is technically rather difficult to study these when n is not a power of two, which is why we did not try $n = 3$. The next lamplighter group we could study are the 8-valued lamplighter groups. These get very big very quickly. We could study tape lengths of at most 8. We consider this to be too short to be worth the effort.

3.4 Performance analysis

Recall that in theory, when we have percolation our algorithm runs in $O(nd)$ where n is the size of our graph and d is the degree of our nodes. This is based on the inner loops of the algorithm. A caveat here lies with the balance of work between the threads. The $O(nd)$ bound is the number of operations needed. Running time is proportional to the amount of operations need only if the work is balanced the same between all threads. Moreover, we are now ignoring the overhead in a loop, only considering the time spent on the inner loops. Thus, there amount of steps taken also affects the total run time.

In the following tables we analyze the wall time of various runs on various graphs. The wall time is how much time elapsed in the 'real world' that is, as if one measured the time by looking at a clock hanging on the wall. This counts time the CPU spent idling, and doesn't count time two cores were running double.

In the first table, we have some runs on small lamplighter graphs, based on a large sample size. These were run with 6 cores of an Intel Xeon E5 2680v3 CPU.

Tape length L	Density	Seconds per Run	Seconds per Run per Node
18	0.12	0.0967	$2.05 \cdot 10^{-8}$
18	0.105	0.1067	$2.26 \cdot 10^{-8}$
20	0.12	0.4667	$2.23 \cdot 10^{-8}$
20	0.105	0.4467	$2.13 \cdot 10^{-8}$
22	0.12	1.7	$1.84 \cdot 10^{-8}$
22	0.1	1.7	$1.84 \cdot 10^{-8}$
22	0.09	0.5	$5.42 \cdot 10^{-9}$
24	0.12	30.6	$7.60 \cdot 10^{-8}$
24	0.1	33.5	$8.32 \cdot 10^{-8}$
24	0.09	11.9	$2.96 \cdot 10^{-8}$

Next, we have data on the biggest lamplighter graph we tested. This is for a tape length of $L = 31$ running on 36 cores spread over 3 Intel Xeon E7 4860v2 CPUs. These 3 CPUs are in the same machine, and have access to the same memory.

Graph	Density	Seconds per Run	Seconds per Run per Node
Lamp 31	0.09	1140.2	$1.71 \cdot 10^{-8}$
Lamp 31	0.089	1702.75	$2.56 \cdot 10^{-8}$
Lamp 31	0.0885	3842.5	$5.79 \cdot 10^{-8}$
Lamp 31	0.088	1011.05	$1.52 \cdot 10^{-8}$

First of all, if compare seconds per run per node for the same graphs at different initial densities, we see an increase that is sometimes followed by a sharp decrease. The decrease are cases where we inadvertently chose an initial density where percolation is not guaranteed, and can thus be ignored. This means that as we approach the critical point the running time increases. This approach to the critical point coincides with needing more steps to percolate. As such, this can be explained by the overhead of a single step. This effect is most pronounced for the example where $L = 31$. This is simply because the initial densities chosen there are the result of trying to find the critical density. For the other test, the initial densities were chosen less carefully.

If next we compare the seconds per run per node for different graph sizes, we see again a marked increase. When comparing the table for $L = 31$ to the other table, take into account that for $L = 31$ we had 6 times as many cores. Thus, naively we would expect a factor 6 improvement in performance. Taking this into account we see the seconds per run per node increase as we take larger graphs. Again, this can be explained by needing more runs, and thus suffering more overhead. Moreover, when we have more nodes, we require more memory. Thus, the probability of incidental cache-hits is higher for the smaller graphs. This could also contribute to computations on smaller graphs being faster.

3.4.1 Profiling results

Next, we used a sampling profiler (specifically `perf record`). This allows us to see how much time is spent in a function and how much time is spent executing any given machine instruction. The goal is to find out which parts are the bottleneck for our algorithm. We did this for the normal Heisenberg group and the normal lamplighter group. We picked the size of the graph so computation was quick (about 10s) and we picked the initial density high enough to guarantee percolation.

In the table below we summarize the pertinent results of this profiling. Three values stood out as pertinent. The first value is the percentage of time spent generating the initial condition. The second value is the percentage of time spent on reading and modifying the variable `counts`. The final value is the percentage of time spent calculating neighbors of a given vertex.

graph type	initial density	initial condition	access `counts`	calculating neighbors
Heisenberg $L = 50$	p=0.1	7%	$50 \pm 2\%$	22%
lamplighter $L = 12$	p-0.15	9%	$47 \pm 2\%$	-

We did not have the percentage of time spent calculating neighbors in the lamplighter graph because this function was inlined by the compiler. These results were based on an Intel core i5-6300U CPU using two threads. The profiled binary was compiled with `icc -O3 -qopenmp -xHOST`.

We see that a very big bottleneck is the array `counts`. This is the array that stores, for each point, how many of its neighbors are active. This all happens in Line 44 and the line below it in Algorithm 3.2. In the actual source code this is line 83 in `bootstrap/bootstrap.h` which can be found in Appendix A. One might think this is due to the atomic increment of that value. Specifically, this might be the result of lock contention. However, the same percentages persist if we remove the atomicity. Instead, we expect this is the result of cache misses. If we look at what happens in these lines, it is essentially the only non-sequential memory access in the main loop. Not only is it non-sequential, it is essentially random. Random access into more than a Gigabyte of memory is rather unlikely to be a cache miss.

The performance penalty of cache misses is why we focus on sequential access in our algorithm and implementation. However, sequential access is not viable for this specific operation. In a given step, we are inherently touching only a very small fraction of `counts` and the locations are fully random. This is inherent to bootstrap percolation, and thus unavoidable. Beyond incurring cache misses, this is also why we did not write an algorithm for a GPU. GPUs incur an even worse penalty for random access into very large arrays.

3.4.2 Comparison with an earlier algorithm

Before we settled on the current implementation, we used another implementation. This older implementation was based on Equation 3.1.1, storing A as a sparse CSC matrix, and storing Δ_i as a full matrix for the benefit of sequential access. Here, we compare the performance of both implementations again based on wall time. These comparison tests were run on an Intel core i5-6300U CPU using two threads. We compared the two algorithms on the Heisenberg group at various side-lengths at two initial densities, $p = 0.1$ and $p = 0.09$. We chose those densities because percolation is almost guaranteed here. Both algorithms performed 20 runs. Note that the old algorithm needs the adjacency matrix. This is calculated once and then used for all 20 runs. The results are in the table below:

L	$p = 0.1$ old	$p = 0.1$ new	old/new $p = 0.1$	$p = 0.09$ old	$p = 0.09$ new	old/new $p = 0.09$
20	12.74s	2.58s	4.94	20.10s	2.69s	7.47
25	40.80s	6.65s	6.14	56.90s	6.94s	8.20
30	89.04s	14.47s	6.15	112.03s	16.26s	6.89
35	176.68s	28.02s	6.31	251.04s	27.73s	9.05
40	316.32s	49.11s	6.44	410.51s	49.74s	8.25
45	579.65s	86.71s	6.68	710.18s	86.58s	8.20

We see that the new algorithm is faster by at least a factor 6. The speed up seems higher for $p = 0.09$ than $p = 0.1$. We expect this is due to lower initial densities needing more steps. The old implementation, due to storing the vector Δ_i as a full vector, needs to iterate over that entire vector at each step. Thus, when there are more steps, each of which add fewer points, the old implementation becomes slower when handling Δ_i.

Another advantage becomes clear when looking at memory usage. This is mostly due to no longer needing to store the adjacency matrix. The CSC format used required $1 + d$ 64-bit integers per node where d is the degree of nodes in our graph. This gives 40 bytes per node of memory for the old algorithm, just for the CSC matrix. There are another 2 bytes per node to store Δ_i and `counts` (Y_i in Equation 3.1.1.

Taking into account how the memory usage of the new implementation scales with the initial density, the new implementation still takes a lot less memory. In practice, looking at an initial density of $p = 0.1$ for the Heisenberg group, we see the new algorithm is a factor 5 improvement to memory usage.

3.5 Conclusion for experimental work

We found evidence suggesting that $p_c = 0$ for the Heisenberg group and found no such evidence for the lamplighter groups. It should be noted that our results for the lamplighter group do not suggest that $p_c \neq 0$ for the lamplighter group. The results were simply inconclusive for the lamplighter group. One interesting thing did pop-up with the lamplighter group: the clustering of final densities around tenths.

Specifically, for the Heisenberg group we found:

$$q_G \approx \frac{0.17}{\log_2 \log_2 \log_2 \frac{L^4}{16}}$$

based on Figure 3.5. This approximation is partly based on a comparison to known results for abelian groups. (specifically, Equation 1.1.1) Moreover, we also saw that an individual run on the Heisenberg group will either almost completely percolate or show almost no growth. This is a clear indicator of critical behavior.

For the lamplighter group, the data suggested an asymptote for q_G around 0.085. This could still be a very slow convergence to 0, but the data simply do not show this. We see some other interesting contrasts between the lamplighter group as compared to both the Heisenberg group and abelian groups. One such difference is the amount of growth before the critical point. With the lamplighter group, there seems to be approximately 10% of growth before the critical point which also increased with the initial density in this regime. For the other groups, there was much less growth for initial densities below the critical point. Moreover, this amount of growth was almost constant, regardless of the initial density.

Next, we get to the most interesting behavior we see with the lamplighter group. Near the critical point q_G, where B'_G is neither close to 0 nor close to 1, we see final densities between 0 and 1. This is already different to what we see for the other groups. Moreover, specifically for the binary lamplighter groups (either with wrapped-tape or the normal group) we see these final densities cluster around tenths. This can be seen in Figure 3.10-b. We do not have an explanation for this, but the phenomena is undeniable. When changing from the binary lamplighter group to the 4-valued lamplighter group, we no longer see this behavior. It seems odd that this behavior is unique to the 2-valued lamplighter groups. Thus we suspect these values also cluster, but perhaps the number of clusters is so high we cannot see them based on our limited number of runs.

3.5.1 Further work

The phenomenon of final densities in the lamplighter group clustering around tenths definitely deserves more attention. Most interesting would be to analyze the subgraphs of active points that correspond to these final densities. One could for example try to visualize these final states. One might expect that the graph can be divided into 10 equal sub-parts, each of which either do or do not become active. However, no such partitioning is obvious here. Especially because the tape-lengths studied aren't multiples of 10.

Beyond studying the tenths phenomenon, there are other interesting possibilities for further work. Most obviously would be trying more amenable groups. Similarly obvious would be, getting a distributed algorithm to look at even bigger subgraphs. Beyond that, we would suggest studying what happens when we pick a value of k that is not half the degree of nodes in our graph. For example, one could add a generator to the Heisenberg group (element $(0, 0, 1)$ and see what happens at $k \in \{2, 3, 4\}$. Based on similarity to the abelian groups, one would expect very quick growth for $k = 2$ and almost no growth for $k = 4$. Similar variations could be tried for the lamplighter group.

It might also be possible to find a totally different approach to calculating bootstrap percolation than our algorithm. For this, one might try to apply methods from the Hashlife algorithm for Conway's Game of Life as presented in [Gos84].

Chapter 4

Conclusion

4.1 Conclusion

Our goal was to make progress towards Conjecture 1.1.2. For this we took a theoretical and an experimental approach. The theoretical approach culminated in Theorem 2.5.1 and Theorem 2.2.12. The former states the conjecture holds for the specific case of abelian groups of rank 2. The latter states the conjecture holds in the case $k > k_S$ for all finitely generated abelian groups. Moreover, we have an approach that might prove the conjecture in the case $k \leq k_S$ for all finitely generated abelian groups based. This is outlined in Section 2.5.1. It is an idea for extending the proof of Theorem 2.5.1. It would be really interesting to see if this outline can be refined into a full proof or whether there is some hidden obstacle blocking the outlined approach.

When considering whether Conjecture 1.1.3 holds in the case $k \leq k_S$ for all abelian graphs, consider Theorem 2.1.9. That theorem states that for these graphs, all k_S-forts are infinite. This contrasts with the case $k > k_S$ where we have proven the existence of finite k-forts. As such, we know that k_S is the 'critical threshold' for the existence of finite k-forts. It then makes a certain amount of sense that this would also be the 'critical threshold' for $p_c = 1$. Moreover, Theorem 2.2.13 means that, for $k \leq k_S$ we have $p_c \geq \pi_c > 0$ on any Cayley graph of a finitely generated abelian group. Furthermore, we know from [Sch92] that $p_c = \pi_c = 0$ when $S = \{e_1 \dots e_d\}$. That is, when S consists of the canonical basis vectors. We feel this strongly supports Conjecture 1.1.3.

The experimental approach had varied results. The experiments seem to confirm that indeed $p_c = 0$ for $k = 2$ on Cayley graphs of the Heisenberg group. However, for Cayley graphs of lamplighter groups we were not able to get such results. That said, our results to not show that $p_c \neq 0$ either. The results were simply inconclusive.

This does suggest an alternative conjecture. Instead of making the conjecture depend on amenability, we might make it depend on whether the growth rate is exponential. The reason being that the lamplighter group is amenable but has an exponential growth rate. We still prefer the original conjecture, because the proofs from [BPP06] depend explicitly on amenability rather than the growth rate.

The experiments also showed a very intriguing phenomenon for binary lamplighter groups. The final densities seems to cluster around tenths. This definitely warrants further investigation. One might hope such investigation gives better insight into bootstrap percolation on the binary lamplighter group. Such insight might even form a basis for theoretical work on the lamplighter group.

It should be noted that Theorem 2.1.13 actually resulted from insights gained by experiments. Specifically, we ran experiments on Cayley-graphs of the group $\mathbb{Z} \times \mathbb{Z}/n\mathbb{Z}$. There we saw that using generators $(1, 0 \mod n\mathbb{Z})$ yielded very little growth at $k = 2$, whilst generators $(1, 1 \mod n\mathbb{Z})$ yielded growth that was very similar to growth on $\mathbb{Z}$. This illustrates that our theoretical and experimental work weren't separate.

Acknowledgments

First and foremost, I owe a debt of gratitude to my supervisors. I am grateful for their guidance when I was stuck, their help pruning options when I saw many ways forward, their gentleness and constructive criticism when I made mistakes, and their patience when my estimated timelines proved too optimistic. Thank you for your help.

Next I wish to thank Dr. Arnold Meijster for his advice on how to parellelize my code. Without his advice, I might still be waiting for my computations to finish. Similarly, I would like to thank the Center for Information Technology of the University of Groningen for their support and for providing access to the Peregrine high performance computing cluster.

Moreover, I want to thank Martijn Kluitenberg for proofreading. He caught many mistakes in texts I had read and re-read many times. Finally I want to thank my friends who supported me in the face of slipping deadlines.

Bibliography

[Sch92] Roberto H Schonmann. "On the behavior of some cellular automata related to bootstrap percolation". In: *The Annals of Probability* (1992), pp. 174–193.

[CLR79] John Chalupa, Paul L Leath, and Gary R Reich. "Bootstrap percolation on a Bethe lattice". In: *Journal of Physics C: Solid State Physics* 12.1 (1979), p. L31.

[Ent87] Aernout CD van Enter. "Proof of Straley's argument for bootstrap percolation". In: *Journal of Statistical Physics* 48.3 (1987), pp. 943–945.

[BPP06] József Balogh, Yuval Peres, and Gábor Pete. "Bootstrap percolation on infinite trees and non-amenable groups". In: *Combinatorics, Probability and Computing* 15.05 (2006), pp. 715–730.

[AL88] Michael Aizenman and Joel L Lebowitz. "Metastability effects in bootstrap percolation". In: *Journal of Physics A: Mathematical and General* 21.19 (1988), p. 3801.

[Hol03] Alexander E Holroyd. "Sharp metastability threshold for two-dimensional bootstrap percolation". In: *Probability Theory and Related Fields* 125.2 (2003), pp. 195–224.

[Bal+12] József Balogh et al. "The sharp threshold for bootstrap percolation in all dimensions". In: *Transactions of the American Mathematical Society* 364.5 (2012), pp. 2667–2701.

[LP16] Russell Lyons and Yuval Peres. *Probability on Trees and Networks*. Vol. 42. Cambridge Series in Statistical and Probabilistic Mathematics. Available at `http://pages.iu.edu/~rdlyons/`. Cambridge University Press, New York, 2016, pp. xv+699. ISBN: 978-1-107-16015-6. DOI: `10.1017/9781316672815`. URL: `http://dx.doi.org/10.1017/9781316672815`.

[GG93] Janko Gravner and David Griffeath. "Threshold growth dynamics". In: *Transactions of the American Mathematical society* (1993), pp. 837–870.

[Lig13] Thomas M Liggett. *Stochastic interacting systems: contact, voter and exclusion processes*. Vol. 324. springer science & Business Media, 2013.

[Gos84] R.Wm. Gosper. "Exploiting regularities in large cellular spaces". In: *Physica D: Nonlinear Phenomena* 10.1 (1984), pp. 75–80. ISSN: 0167-2789. DOI: `https://doi.org/10.1016/0167-2789(84)90251-3`. URL: `http://www.sciencedirect.com/science/article/pii/0167278984902513`.

Appendix A

C++ code

../code/C_bootstrap/bootstrap/bootstrap.h

```
#pragma once

#if defined(_OPENMP)
#include <omp.h>
#else
typedef int omp_int_t;
inline omp_int_t omp_get_thread_num() { return 0;}
inline omp_int_t omp_get_num_threads() { return 1;}
#endif

#include <iostream>
#include <vector>
#include <atomic>
#include <array>
#include <random>

#include "../pcg_random/include/pcg_random.hpp"

std::vector<long int> initializeSparse(const float density, const long int
    startId, const long int endId);

struct bootstrapResult{
    long int initialOccupied, finalOccupied;
    long int nodeCount;
    long int stepsTaken;
    long int stepsOrigin;
};

template<typename GraphType>
bootstrapResult bootstrap(
        const GraphType graph,
        const float density,
        const int threshold
        ){
    using namespace std;

    long int nodeCount = graph.numNodes();
    long int origin    = graph.origin();
    bootstrapResult result;
    result.stepsOrigin = -1;
    atomic<long int> occupied(0);
    occupied = 0;
```

```
42      atomic<char> *neighborCounts = new atomic<char>[nodeCount]();
43
44      result.nodeCount = nodeCount;
45      long int finalStepCount = 0;
46      #pragma omp parallel shared(occupied, neighborCounts)
47      {
48          long int threadId = omp_get_thread_num();
49          long int nThreads = omp_get_num_threads();
50
51          long int lastOccupied = 0;
52          vector<long int> newNeighbors;
53
54          long int nodesPerThread = (nodeCount + nThreads - 1) / nThreads;
55          vector<long int> newNodes = initializeSparse(
56                  density,
57                  nodesPerThread * threadId,
58                  min( nodesPerThread * (threadId + 1), nodeCount )
59          );
60          for (long int node : newNodes){
61              neighborCounts[node] = threshold;
62          }
63
64          occupied += newNodes.size();
65          #pragma omp barrier
66          #pragma omp single
67          {
68              result.initialOccupied = occupied;
69          }
70
71          long int stepCount;
72          for(stepCount = 0; true; stepCount++) {
73              newNeighbors.clear();
74              for (long int idx : newNodes){
75                  auto row = graph.getNeighbors(idx);
76                  for (auto elem : row){
77                      if (elem >= 0) newNeighbors.push_back (elem);
78                  }
79              }
80
81              newNodes.clear();
82              for (auto neighb : newNeighbors){
83                  if (++neighborCounts[neighb] == threshold){
84                      newNodes.push_back(neighb);
85                  }
86              }
87              occupied += newNodes.size();
88
89              // Race condition does not matter,
90              // if they write, they write at the same time and write the smae
    thing
91              if (result.stepsOrigin == -1 and neighborCounts[origin] >= threshold)
    {
92                  result.stepsOrigin = stepCount;
93              }
94          #pragma omp barrier
95              if (occupied == lastOccupied) {
96                  finalStepCount = stepCount;
97                  break;
98              }
99              lastOccupied = occupied;
100         #pragma omp barrier
```

```
101         }
102     }
103     result.stepsTaken = finalStepCount;
104     result.finalOccupied = occupied;
105 
106     delete [] neighborCounts;
107 
108     return result;
109 }
```

../code/C_bootstrap/bootstrap/initializeState.cpp

```
1  #include "bootstrap.h"
2  
3  using namespace std;
4  
5  vector<long int> initializeSparse(const float p, const long int startId, const
       long int endId){
6      vector<long int> state;
7      pcg32_fast rng(pcg_extras::seed_seq_from<std::random_device>{});
8      bernoulli_distribution bernoulli(p);
9      for(long int idx = startId; idx != endId; idx++){
10         if (bernoulli(rng)){
11             state.push_back(idx);
12         }
13     }
14     return state;
15 }
```

../code/C_bootstrap/graphGen/squareHeis.h

```
1  #pragma once
2  
3  #pragma GCC diagnostic ignored "-Wsubobject-linkage"
4  
5  #include <array>
6  #include "libdivide.h"
7  
8  struct Point{
9      int x, y, z;
10 } typedef Point;
11 
12 
13 struct Heisenberg
14 {
15     const int xLength, yLength, zLength;
16     constexpr static int numGens = 4;
17     const long int rectangleXYlength;
18 
19     const libdivide::divider<long int> rectangleXYdivider, xLengthDivider;
20 
21     Heisenberg(int xLength, int yLength, int zLength);
22 
23     long int numNodes() const ;
24     long int origin() const;
25     std::array<long int, 4> getNeighbors(const long int inputId) const ;
26     void generateCsc(int64_t *rowIdxs, int64_t *colPtrs) const ;
27 
28     Point id2point(const long int id) const ;
29     long int point2id(const Point point) const ;
30     bool validPoint(const Point point) const ;
```

```
31 private:
32     long int l2s(const long int x) const ;
33
34 };
```

../code/C_bootstrap/graphGen/squareHeis.cpp

```
1  #include "squareHeis.h"
2
3  Heisenberg::Heisenberg(int xLength, int yLength, int zLength)
4          : xLength(xLength)
5          , yLength(yLength)
6          , zLength(zLength)
7          , rectangleXYlength(l2s(xLength) * l2s(yLength))
8          , rectangleXYdivider(l2s(xLength) * l2s(yLength))
9          , xLengthDivider(l2s(xLength))
10 {}
11
12 long int Heisenberg::numNodes() const{
13     return (long int) l2s(xLength) * l2s(yLength) * l2s(zLength);
14 }
15
16 long int Heisenberg::origin() const
17 {
18     return point2id(Point{0, 0, 0});
19 }
20
21 long int Heisenberg::l2s(const long int x) const{
22     return (2L * x + 1L);
23 }
24
25 // hard-coded generators
26 std::array<long int, 4> Heisenberg::getNeighbors(const long int inputId) const {
27     Point inputPoint = id2point(inputId);
28     int x = inputPoint.x;
29     int y = inputPoint.y;
30     int z = inputPoint.z;
31     std::array<long int, 4> output;
32
33     /* Computing the adjacent elements according to the generators. We use gen x
       point in the heisenberg group
34      * presuming the point is:
35      * 1 x z
36      * 0 1 y
37      * 0 0 1
38      *
39      * Hard coded the following generators in order
40      * 1 1 0    1 0 0    1 -1 0    1 0  0
41      * 0 1 0    0 1 1    0  1 0    0 1 -1
42      * 0 0 1    0 0 1    0  0 1    0 0  1
43      * Note that all have the z element equal to 0
44      * This makes the acutal point computation easy
45      */
46     output[0] = point2id(Point{x + 1, y,     z    });
47     output[1] = point2id(Point{x    , y + 1, z + x});
48     output[2] = point2id(Point{x - 1, y    , z    });
49     output[3] = point2id(Point{x    , y - 1, z - x});
50     return output;
51 }
52
53 Point Heisenberg::id2point(const long int id) const {
```

```
54      int z = id / rectangleXYdivider;
55      // calculate remainder given divizor.
56      long int remainder = id - (z * rectangleXYlength);
57
58      int y = remainder / xLengthDivider;
59      int x = remainder - (y * (long int) l2s(xLength));
60
61      return Point{x - xLength, y - yLength, z - zLength};
62  }
63
64  long int Heisenberg::point2id(const Point point) const {
65      return validPoint(point) ? ((long int)point.x + xLength) +
66                                 (point.y + yLength) * l2s(xLength) +
67                                 (point.z + zLength) * rectangleXYlength
68                             : -1L;
69  }
70
71  bool Heisenberg::validPoint(const Point point) const {
72      return abs(point.x) <= xLength &&
73             abs(point.y) <= yLength &&
74             abs(point.z) <= zLength;
75  }
76
77  // Generate the Cscmatrix of this cayleygraph by essentially building an
        adjacency list
78  void Heisenberg::generateCsc(long int *rowIdxs, long int *colPtrs) const {
79
80      // keep track of rowIdxs length for colPtrs
81      long int totalConnections = 0;
82      std::array<long int, 4> neighbours;
83      long int numNodesInt = numNodes();
84      for (long int nodeId = 0; nodeId < numNodesInt; nodeId++){
85          colPtrs[nodeId] = totalConnections;
86
87          neighbours = getNeighbors(nodeId);
88          for (int neighbIdx = 0; neighbIdx < 4; neighbIdx++){
89              if (neighbours[neighbIdx] >= 0){ // neighbours that fall outside the
        domain get id -1
90                  rowIdxs[totalConnections++] = neighbours[neighbIdx];
91              }
92          }
93      }
94      // finally, 'close' colPtrs
95      colPtrs[numNodesInt] = totalConnections;
96  }
```

../code/C_bootstrap/graphGen/lampLighter.h

```
1   #pragma once
2
3   #include <array>
4
5   struct LampLighter {
6       static constexpr int numGens = 4;
7
8       const long int positions;
9
10      LampLighter(const long int positions);
11
12      long int numNodes() const;
13      long int origin() const;
```

```
14      std::array<long int, numGens> getNeighbors(const long int elem) const;
15      void printTape(long int node, std::ostream &out);
16  };
```

../code/C_bootstrap/graphGen/lampLighter.cpp

```
1   #include "lampLighter.h"
2   #include <iostream>
3   using namespace std;
4
5   LampLighter::LampLighter(const long int positions) : positions(positions) {}
6
7   array<long int, LampLighter::numGens> LampLighter::getNeighbors(const long int
        elem) const{
8       array<long int, numGens> result;
9       long int headPos = elem >> positions;
10
11      if (headPos < positions - 1){
12          // move right
13          result[0] = elem + (1L << positions);
14          // toggle, then move right
15          result[1] = (elem ^ (1L << headPos)) + (1L << positions);
16      } else {
17          result[0] = result[1] = -1;
18      }
19      if (headPos > 0){
20          // move left
21          result[2] = elem - (1L << positions);
22          // move left, then toggle
23          // Note order reversal as opposed to result[1].
24          // Otherwise, this is not an inverse
25          headPos = headPos - 1;
26          result[3] = (elem ^ (1L << headPos)) - (1L << positions);
27      } else {
28          result[2] = result[3] = -1;
29      }
30
31      return result;
32  }
33
34  long int LampLighter::numNodes() const{
35      return positions << positions;
36  }
37
38  long int LampLighter::origin() const{
39      return (positions / 2) << positions;
40  }
41
42  void LampLighter::printTape(long int node, ostream &out){
43      long int headPos = node >> positions;
44      for(long int idx = positions; idx-- != 0; ){
45          out << (idx == headPos ? "_ " : "  ");
46      }
47      out << '\n';
48
49      for(long int idx = positions; idx-- != 0; ){
50          out << ((node >> idx ) & 1) << ' ';
51      }
52      out << '\n';
53  }
```

../code/C_bootstrap/graphGen/wrapLighter.h

```h
#pragma once

#include <array>

struct LampLighter {
    static constexpr int numGens = 4;

    const long int positions;

    LampLighter(const int positions);

    long int numNodes() const;
    long int origin() const;
    std::array<long int, numGens> getNeighbors(const long int elem) const;
    void printTape(long int node, std::ostream &out);
};
```

../code/C_bootstrap/graphGen/wrapLighter.cpp

```cpp
#include "lampLighter.h"
#include <iostream>
using namespace std;

LampLighter::LampLighter(const long int positions) : positions(positions) {}

array<long int, LampLighter::numGens> LampLighter::getNeighbors(const long int
    elem) const{
    array<long int, numGens> result;
    long int headPos = elem >> positions;

    if (headPos < positions - 1){
        // move right
        result[0] = elem + (1L << positions);
        // toggle, then move right
        result[1] = (elem ^ (1L << headPos)) + (1L << positions);
    } else {
        // wrap around
        result[0] = elem & ((1L << positions) - 1);
        // toggle, wrap
        result[1] = result[0] ^ (1L << headPos);
    }
    if (headPos > 0){
        // move left
        result[2] = elem - (1L << positions);
        // move left, then toggle
        // Note order reversal as opposed to result[1].
        // Otherwise, this is not an inverse
        headPos = headPos - 1;
        result[3] = (elem ^ (1L << headPos)) - (1L << positions);
    } else {
        // wrap
        result[2] = elem | ((positions - 1) << positions);
        // move left, then toggle
        // Note order reversal as opposed to result[1].
        // Otherwise, this is not an inverse
        headPos = positions - 1;
        result[3] = (elem ^ (1L << headPos)) | ((positions - 1) << positions);
    }

    return result;
```

```
41 }
42
43 long int LampLighter::numNodes() const{
44     return (long int) positions << positions;
45 }
46
47 long int LampLighter::origin() const{
48     return (long int) positions / 2 << positions;
49 }
50
51 void LampLighter::printTape(long int node, ostream &out){
52     long int headPos = node >> positions;
53     for(long int idx = positions; idx-- != 0; ){
54         out << (idx == headPos ? "_ " : "  ");
55     }
56     out << '\n';
57
58     for(long int idx = positions; idx-- != 0; ){
59         out << ((node >> idx ) & 1) << ' ';
60     }
61     out << '\n';
62 }
```

../code/C_bootstrap/graphGen/lampExtend.h

```
1  #include <array>
2  #include <iostream>
3  #include <stdexcept>
4
5  template<int baseLog2>
6  struct LampExtend {
7      static constexpr int numGens = 4;
8      static constexpr int base = 1 << baseLog2;
9
10     const long int positions;
11     const long int maxOffset;
12
13     LampExtend(const long int positions);
14
15     long int numNodes() const;
16     long int origin() const;
17     std::array<long int, 4> getNeighbors(const long int elem) const;
18     void printTape(long int node, std::ostream &out);
19 };
20
21 /* A node in this graph is a head position and a tape of length
22  * $positions$. Each position on the tape taking $base = 2^baseLog2$ values
23  * with modular arithmetic.
24  *
25  * This is encoded in a long int, storing position $i$ on the tape in bits
       $baseLog2 i$
26  * until $baseLog2 (i + 1)$. Those are appended by the head position in bits
27  * $baselog2 * positions$ untill the end.
28  *
29  * That is, we encode the tape in the lower bits, and the head position
       afterwards.
30  */
31
32 template<int baseLog2>
33 LampExtend<baseLog2>::LampExtend(const long int positions)
34     : positions(positions)
```

```
35      , maxOffset(positions * baseLog2)
36  {
37      // positions * baseLog2 for tape data
38      // 5 = log_2(64) for head location data
39      // finally another baseLog2 + 1 to prevent overflow after adding minusOne
40      // (the +1 is because long int is signed)
41      if (positions * baseLog2 + 5 + baseLog2 + 1 >= sizeof(long int) * 8){
42          throw std::invalid_argument("To many positions, might not fit in long int
        ");
43      }
44  }
45
46  /* Calculate neighbors based on 4 generators:
47   * - move head right
48   * - increment value under current head and move head right
49   * - move head left
50   * - decrement value under current head and move head left
51   * (left is decreasing position, right is increasing position)
52   * When head is at maximal left position, the final two generators do not apply
53   * Similarly at maximal right position the other two generators do not apply
54   * (our tape does not wrap arround)
55   */
56  template<int baseLog2>
57  std::array<long int, 4> LampExtend<baseLog2>::getNeighbors(const long int elem)
        const
58  {
59      std::array<long int, 4> result;
60      const long int headPos    = (elem >> maxOffset);
61      const long int headOffset = baseLog2 * headPos;
62      // 1s except at the bits corresponding to head
63      const long int mask = ~( ((1L << baseLog2) - 1L) << headOffset);
64
65      if (headPos < positions - 1){
66      // move head right
67          result[0] = elem + (1L << maxOffset);
68
69      // increment then move head right
70          long int incremented = elem + (1L << headOffset);
71          // mask out potential overflow
72          incremented = incremented ^ ((elem ^ incremented) & mask);
73          result[1] = incremented + (1L << maxOffset);
74      } else {
75          result[0] = result[1] = -1;
76      }
77      if (headPos > 0){
78      // move head left
79          result[2] = elem - (1L << maxOffset);
80
81      // move head Left, then decrement
82          // Create -1 mod base by taking base - 1
83          // This prevents underflow in the case of 0 at a position
84          constexpr long int minusOne = (1L << (baseLog2)) - 1;
85
86          // Decrement element left of head
87          long int decremented = elem + (minusOne << (headOffset - baseLog2));
88          // mask out overflow
89          decremented = decremented ^ ((elem ^ decremented) & (mask >> baseLog2));
90          result[3] = decremented - (1L << maxOffset);
91      } else {
92          result[2] = result[3] = -1;
93      }
```

```
94      return result;
95  }
96
97  template<int baseLog2>
98  long int LampExtend<baseLog2>::origin() const{
99      // set head in the middle, tape remains at 0
100     return (positions / 2) << positions * baseLog2;
101 }
102
103 template<int baseLog2>
104 long int LampExtend<baseLog2>::numNodes() const{
105     return  positions << positions * baseLog2;
106 }
107
108 template<int baseLog2>
109 void LampExtend<baseLog2>::printTape(long int node, std::ostream &out){
110     long int headPos = node >> maxOffset;
111     if (baseLog2 > 3)
112         out << headPos << ' ';
113     else {
114         for(long int idx = positions; idx-- != 0; ){
115             out << (idx == headPos ? "_ " : "  ");
116         }
117         out << '\n';
118     }
119     constexpr long int mask = (1L << (baseLog2)) - 1;
120     for(long int idx = positions; idx-- != 0; ){
121         out << ((node >> (baseLog2 * idx)) & mask) << ' ';
122     }
123     out << '\n';
124 }
```

../code/C_bootstrap/graphGen/abelian.h

```
1   #pragma once
2
3   #include <cstdint>
4   #include <cmath>
5   #include <cstdlib>
6   #include <array>
7
8   using namespace std;
9
10  template<int dimension> struct Abelian
11  {
12      const long int sideLength;
13      static const int numGens = 2 * dimension;
14
15      Abelian(int64_t sideLength);
16
17      long int numNodes() const;
18      long int origin() const;
19      std::array<long int, 2 * dimension> getNeighbors(const long int inputId)
        const ;
20      void generateCsc(int64_t *rowIdxs, int64_t *colPtrs) const ;
21
22      std::array<long int, dimension> id2point(long int id) const ;
23      long int point2id(const std::array<long int, dimension>  point) const ;
24      bool validPoint(const std::array<long int, dimension>  point) const ;
25  };
26
```

```
27 template<int dimension>
28 Abelian<dimension>::Abelian(long int sideLength) :
29         sideLength(sideLength)
30         {}
31
32 template<int dimension>
33 long int Abelian<dimension>::numNodes() const{
34     return  std::pow(sideLength , dimension);
35 }
36
37 template<int dimension>
38 long int Abelian<dimension>::origin() const{
39     return numNodes() / 2;
40 }
41
42 template<int dimension>
43 std::array<long int, 2 * dimension> Abelian<dimension>::getNeighbors(const long
       int inputId) const{
44     std::array<long int, dimension> inputPoint = id2point(inputId);
45     std::array<long int, 2 * dimension> output;
46
47     std::array<long int, dimension>  modifiedPoint;
48     for (int coorIdx = 0; coorIdx < dimension; coorIdx++){
49         // Even indexed points in output get incremented at coorIdx
50         modifiedPoint = inputPoint;
51         modifiedPoint[coorIdx] += 1;
52         output[2 * coorIdx    ] = point2id(modifiedPoint);
53         // Odd  indexed points in output get decremented at coorIdx
54         modifiedPoint = inputPoint;
55         modifiedPoint[coorIdx] -= 1;
56         output[2 * coorIdx + 1] = point2id(modifiedPoint);
57     }
58     return output;
59 }
60
61 template<int dimension>
62 std::array<long int, dimension>  Abelian<dimension>::id2point(long int id) const{
63     std::array<long int, dimension> point;
64
65     for(int idx = dimension; idx--; ){
66         point[idx] = id % sideLength;
67         id /= sideLength;
68     }
69     return point;
70 }
71
72 template<int dimension>
73 long int Abelian<dimension>::point2id(const std::array<long int, dimension> point
       ) const{
74     int64_t id = 0;
75     if (! validPoint(point)){
76         id = -1;
77         return id;
78     }
79     for(int idx = 0; idx < dimension; idx++){
80         id *= sideLength;
81         id += point[idx];
82     }
83     return id;
84 }
85
```

```
86 template<int dimension>
87 inline bool Abelian<dimension>::validPoint(const std::array<long int, dimension>
       point) const{
88     for(int idx = 0; idx < dimension; idx++){
89         if (point[idx] >= sideLength || point[idx] < 0)
90             return false;
91     }
92     return true;
93 }
```