An Introduction to Chip-Firing

Robin Truax

May 2021

Contents

1 Introduction

The goal of these notes are to outline the formal math behind the images posted on my website here (and also at the end of these notes), which I think only adds to the aesthetic beauty of the images. However, following these notes is not at all necessary to appreciate the images, and if you do not feel up to diving into the formal math, I want to provide an option for you. If you don't want to work through these notes (or even if you do), I encourage you to think of the images in the following visual way.

Imagine a square table. On this square table we have placed thousands, perhaps millions, of tiny cubic grains of sand, aligned with each other to span the entire table. These grains of sand, despite their symmetric shape, are unstable due to their miniscule size; if there are more than 4 grains of sand stacked on top of each other, the tower will fall, spilling grains of sand equally in all four directions (any extra grains of sand, up to 3, it will retain). On the edges, sometimes towers spill over the edge, scattering grains of sand onto the ground.

You and I, out of curiosity at the dynamics of this system, have invented a method of dumping one sandpile onto another sandpile perfectly, with each tower stacking entirely on another. It's an engineering marvel and probably not physically possible, but it's certainly fun to watch. The identity elements – graphically represented by the images referenced above – are the unique "big enough" piles of sand such that, when we dump them on another sandpile and let the dust (or, rather, grains of sand) clear, we get the sandpile we started with. I can't create such a machine, so I can only imagine it – but in my mind's eye, watching the grains of sand roll off the table to reveal the exact configuration we started with looks absolutely magical.

1.1 Important Considerations

These notes are based on the book "The Mathematics of Chip-Firing" by Caroline J. Klivans. These notes draw from and summarize Chapters 1-4. Special thanks to Libby Taylor, who read this book alongside me and answered any questions I had as a part of Stanford's Math Directed Reading Program.

Roughly speaking, chip-firing can be understood as a discrete form of diffusion, where a certain number of chips are placed on the vertices of a graph and spread out from areas of high concentration. As such, the game of chip-firing is combinatorial in nature, but since it is usually analyzed with tools from algebra (in particular, recent research has emphasized the role of algebraic geometry and commutative algebra) it is classified as belonging to the field of *algebraic combinatorics*.

Assumed prerequisites: group theory (up through the structure theorem for finitely generated abelian groups), abstract linear algebra, and the very basics of graph theory. Any terms not often defined in these classes (ex. cokernel) will be defined and explained here.

Important assumption: All graphs discussed will be *connected*.

2 What is Chip-Firing?

Let $G = (V, E)$ be a finite graph with n vertices. Label the vertices of G by $v_1, \ldots, v_n$. Our first step will be to formalize the concept of placing "chips" on the vertices of a graph.

Definition 1 (Chip Configuration)**.** A *chip configuration* for G is a non-negative integer vector

$$\mathbf{c} = (c_1, c_2, \ldots, c_n) \in \mathbb{Z}^n_{\geq 0}$$

where each coordinate of the vector corresponds to the number of "chips" at a particular vertex. Each coordinate c_i represents the number of chips at the vertex v_i.

Definition 2 (Ready to Fire)**.** Given a chip configuration $\mathbf{c}$ for G, a vertex v is *ready to fire* if $c_v \geq \deg(v)$.

Definition 3 (Firing)**.** Given a chip configuration $\mathbf{c}$ for G, a vertex v *fires* by sending one chip to each of its neighbors. Formally, this process corresponds to creating a new vector $\mathbf{c}' = (c'_1, \ldots, c'_n) \in \mathbb{Z}^n$ where

1. $c'_v = c_v - \deg v$,
2. $c'_w = c_w + 1$ for each w neighboring v, and
3. $c'_w = c_w$ for each vertex w not neighboring w.

We say that this is a *legal fire* if $\mathbf{c}'$ is still a chip configuration; that is, if its entries are all still nonnegative.

Definition 4 (Stable)**.** A chip configuration $\mathbf{c}$ is *stable* if no vertex is ready to fire.

Definition 5 (Chip-Firing Process)**.** The *chip-firing process* on a finite graph G starts with an initial chip configuration $\mathbf{c}$, and at each step selects and fires a vertex which is ready to fire. If, at any step, a stable configuration is reached, then the process stops and we say the process has stabilized.

Note that the chip-firing process is not guaranteed to stabilize (we will see examples and theorems describing cases where it does not).
Now, *prima facie* it might appear that the terminal state of the chip-firing process might depend how we choose which ready-to-fire vertex we fire (if there is more than one ready vertex), but we will soon see that chip-firing satisfies a "confluence property" which avoids this issue. More precisely, we will see that if the chip-firing process starting with a chip configuration $\mathbf{c}$ for G terminates, there is a unique stable chip configuration $\mathbf{c}'$ which is reachable from $\mathbf{c}$ by legal fires.

2.1 The Adjacency and Laplacian Matrices

Definition 6 (Adjacency Matrix)**.** The adjacency matrix of G is the $n \times n$ matrix A given by

$$A_{ij} = \begin{cases} 1 & \text{if } \{v_i, v_j\} \in E \\ 0 & \text{otherwise.} \end{cases}$$

Definition 7 (Laplacian)**.** The *Laplacian* $\Delta(G)$ *of* G is the $n \times n$ matrix given by

$$\Delta_{ij} = \begin{cases} \deg(v_i) & \text{if } i = j \\ -1 & \text{if } \{v_i, v_j\} \in E \\ 0 & \text{otherwise.} \end{cases}$$

In other words, if D is the diagonal matrix with $D_{ij} = \delta_{ij} \deg(v_i)$, then $\Delta(G) = D - A$. It is a basic result in graph theory that the rank of $\Delta(G)$ is equal to the number of vertices of G minus the number of connected components of G. Since we assume G is connected, $\Delta(G)$ has rank $n-1$. In particular, the kernel of $\Delta(G) : \mathbb{Z}^n \to \mathbb{Z}^n$ is spanned by the vector $\mathbf{1} = (1, \ldots, 1)$.

Note: By abuse of notation, we will denote $\Delta(G)$ by Δ when the graph in question is clear.

The Laplacian is relevant because firing at the vertex v_i corresponds to subtracting the ith row of $\Delta(G)$ from the current chip configuration. Precisely,

Proposition 1. *Given a chip configuration* $\mathbf{c}$*, the chip configuration* $\mathbf{c}'$ *given by firing at* v_i *is*

$$\mathbf{c}' = \mathbf{c} - \Delta e_i$$

where $e_i = \begin{bmatrix} \delta_{i1} & \ldots & \delta_{in} \end{bmatrix}^\top$ *is the* i*th standard basis vector in* $\mathbb{R}^n$*.*

Definition 8 (Characteristic Vector)**.** Let $S \subseteq V$ be a subset of vertices of a graph G. Then the *characteristic vector* of $S \subseteq V$ is defined to be

$$\chi_S = \sum_{i=1}^{n} x_i e_i,$$

where e_i is the ith standard basis vector in $\mathbb{R}^n$, and $x_i = 1$ if $v_i \in S$ and 0 otherwise. Therefore, χ_S is the vector whose ith entry is 1 if $v_i \in S$ and 0 otherwise.

Definition 9. Let $S \subseteq V$ be a subset of vertices of G. Then the *cluster-fire* of S from configuration $\mathbf{c}$ is

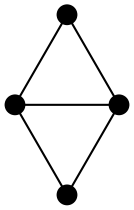
$$\mathbf{c}' = \mathbf{c} - \Delta(G)\chi_S.$$

The cluster-fire is called *legal* if $\mathbf{c}'$ is still a chip configuration. We imagine this as firing simultaneously at all vertices in S, starting from the chip configuration $\mathbf{c}$ on G.

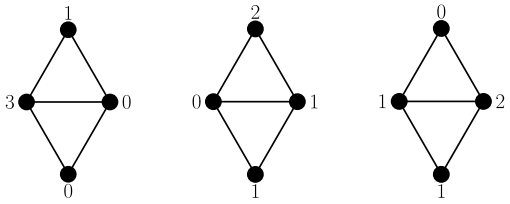
Notice that it is possible for a cluster-fire of S from configuration $\mathbf{c}$ to be legal while individually firing at some of the vertices of S is illegal. For example, the cluster-fire at the entire vertex set V is always legal (it leaves the chip configuration unchanged) even while various vertices are not ready to fire.

2.2 Examples of Chip-Firing

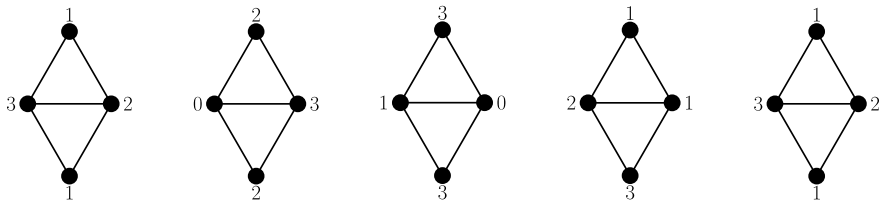
Here, as a way to build intuition, we consider two examples of the chip-firing process – one that stabilizes, and one that does not. We'll use the following graph with just four vertices as G.



First, let's look at a chip-firing process that stabilizes.



Next, let's look at a chip-firing process which doesn't stabilize.



Notice that the first and last configurations are the same, implying an infinite loop.

2.3 Local and Global Confluence

Theorem 2 (Local Confluence)**.** *Suppose* $\mathbf{c}_1$ *and* $\mathbf{c}_2$ *are two configurations on a graph* G *which are both reachable from a configuration* $\mathbf{c}$ *after one firing. Then there exists a common configuration* $\mathbf{d}$ *reachable from both* $\mathbf{c}_1$ *and* $\mathbf{c}_2$ *after a single firing.*

Proof. Suppose $\mathbf{c}_1$ and $\mathbf{c}_2$ are reachable from $\mathbf{c}$ by firing v_1 and v_2. After firing v_1, then we are certainly still able to fire v_2 (since the number of chips at v_2 has either stayed the same or increased). Similarly, after firing v_2, then we are certainly still able to fire v_1. Therefore, firing in either order are both equal to the cluster fire at $\{v_1, v_2\}$. □

Corollary 2.1 (Global Confluence)**.** *If a stable configuration* $\mathbf{c}_s$ *is reachable from a configuration* $\mathbf{c}$ *after a finite number of legal fires, then* $\mathbf{c}_s$ *is the unique stable configuration reachable from* $\mathbf{c}$*. Also, the number of times each site fires (and hence the length) of any stabilizing sequence is the same.*

Proof. This simply results from inductively applying local confluence. □

Definition 10 (Stabilization of a Configuration)**.** If $\mathbf{c}$ is a chip configuration, then let $\text{stab}(\mathbf{c})$ denote the unique stable configuration resulting from a sequence of legal fires from $\mathbf{c}$ (if one exists).

2.4 Basic Results on Stabilization

Lemma 3. *Suppose the initial configuration* $\mathbf{c}$ *on a connected graph* G *never stabilizes. Then every vertex fires infinitely often in the chip-firing process starting at* $\mathbf{c}$*.*

Proof. By the Pigeonhole Principle, some vertex must fire infinitely many times. But then, it distributes infinitely many chips to its vertices, so each of its neighbors will fire infinitely many times, each of *their* neighbors will fire infinitely many times, and so on. Since the graph is connected, this inductive process will necessarily reach every vertex. □

Corollary 3.1. *The chip-firing process starting from an initial configuration* $\mathbf{c}$ *is finite if and only if there exists a vertex which fires finitely many times during the stabilization of* $\mathbf{c}$*.*

Lemma 4. *Given a configuration* $\mathbf{c}$ *on* G*, if every vertex of* G *can fire at least once, then the chip-firing process from* $\mathbf{c}$ *does not stabilize.*

Proof. Suppose all vertices fire and the process stabilizes. Then, if v is the vertex which stops firing first, then all the other vertices fire at least once. But then v has certainly gained at least $\deg v$ vertices, so it can fire again! Thus the process is not stable, a contradiction. □

Corollary 4.1. *The chip-firing process starting from an initial configuration* $\mathbf{c}$ *is finite if and only if there exists some vertex which never fires during the stabilization of* $\mathbf{c}$*.*

Now, we'll give some rough bounds that will help us discover when chip configurations stabilize.

Theorem 5. *Let* G *be a finite connected graph with* n *vertices and* m *edges. Suppose* $\mathbf{c}$ *is a chip configuration on* G *with* N *chips (that is,* $c_1 + \cdots + c_n = N$*). Then,*

1. *If* $N > 2m - n$*, the chip-firing process starting at* $\mathbf{c}$ *is infinite.*
2. *If* $m \leq N \leq 2m - n$ *then the chip-firing process starting at* $\mathbf{c}$ *could be finite or infinite.*
3. *If* $N < m$*, then the chip-firing process starting at* $\mathbf{c}$ *is finite.*

Proof.
(1) For the first part of the question, suppose $N > 2m - n$. Now, $\sum_{v \in V} \deg(v) = 2m$. Thus, by the Pigeonhole Principle, there's always a vertex with at least $\deg(v)$ chips, and hence always a vertex ready to fire.

(2) For this part of the question, we need to find a configuration whose chip-firing process is finite and a configuration whose chip-firing process is infinite for each N satisfying $m \leq N \leq 2n - m$. For the former, take the configuration with $\deg(v) - 1$ chips at each vertex. This has $2n - m$ chips and is stable, so its chip-firing process is trivially finite. We can also remove some of these chips to get a stable configuration with anywhere between m and $2m - n$ chips.

However, producing configurations satisfying the chip-requirement of (2) whose chip-firing process is infinite is more difficult, and requires a few definitions and propositions to do with directed graphs.

Definition 11 (Acyclic Orientation)**.** Given an (undirected) graph G, an *orientation* of G is an assignment of directions to each edge of G, making G into a directed graph. An *acyclic orientation* of G is an orientation of G such that there are no directed cycles.

Proposition 6. *Every finite graph* G *has an acyclic orientation.*

Proof. Label the vertices of G by $v_1, \ldots, v_n$ arbitrarily. Then, if there exists an edge between v_i and v_j, direct the edge from v_i to v_j if $i < j$ and from v_j to v_i if $j < i$. There are no cycles in this orientation because of the transitive property of the usual ordering on the natural numbers. □

Definition 12 (Indegree and Outdegree)**.** Suppose v is a vertex in a directed graph G. Then the number of edges pointing into v is the *indegree* of v, and the number of edges pointing out of v is the *outdegree* of v. A *parent* of v is a vertex with an edge pointing to v; the number of parents of v is equal to the indegree of v.

Definition 13 (Source Vertex)**.** A source vertex v_s is a vertex of a directed graph with indegree 0.

Proposition 7. *Every finite graph* G *with an acyclic orientation* $\mathcal{O}$ *has a source vertex* v_s*.*

Proof. Suppose, for the sake of contradiction, that every vertex of G has indegree 1 or higher. Pick any vertex of G; since it has positive indegree it has a parent. Visit the parent, and since this parent has positive indegree we can visit *its* parent. Repeating this process, we notice that by the Pigeonhole principle we must visit a vertex we've already visited. But this induces a cycle, a contradiction with the fact that $\mathcal{O}$ is an acyclic orientation. Hence our assumption was incorrect, and a vertex of G has indegree 0, as desired. □

Hence we can take an acyclic orientation of G. Place $\text{outdeg}(v)$ chips at each vertex v; this uses a total of m chips since each edge adds one to the sum of all vertices' outdegrees. Let v_s be the source vertex of this orientation, and fire at v_s. Then, reverse the orientation of all edges incident to v_s. The resulting orientation is acyclic since v_s was a source vertex, and the number of chips at each vertex is still equal to the outdegree. Hence we can repeat. Furthermore, if we add more chips, the chip-firing process is still infinite, so by adding chips we can get a configuration with infinite chip-firing process and N chips for any $m \leq N \leq 2m-n$.

(3) For the final part of the question, suppose $N < m$. Now, for each edge e in the graph, consider the first chip that fires across e. Associate this chip with e so that in every following firing, this chip either moves

across e or doesn't move at all. The initial configuration has less chips than edges, so there is some edge with no associated chip. But that means that the two endpoints of this edge do not fire, so by our earlier work the configuration must stabilize. □

3 Chip-Firing with Sinks

Definition 14 (Graph with a Sink)**.** Let $G = (V, E)$ be a finite graph and $q \in V$ be a distinguished vertex (called the *sink vertex*). Then (G, q) is called a *graph with a sink*. By abuse of notation, we often denote (G, q) by G when the sink vertex q is well-understood.

Definition 15 (Configuration with a Sink)**.** Let $G = (V, E)$ be a finite graph with $n + 1$ vertices and sink vertex q. Then a chip configuration on G is an integer vector $\mathbf{c} = (c_1, \ldots, c_n, c_q)$ such that $c_i \geq 0$ for all $i \neq q$. Notice that c_q is allowed to be negative! In fact, we often normalize the configuration on a graph with a sink by defining $c_q = -\sum_{i=1}^{n} c_i$.

Definition 16 (Stable Configuration)**.** A configuration $\mathbf{c}$ on a graph with a sink is stable if no non-sink vertex is ready to fire; that is, $c_v < \deg(v)$ for all non-sink vertices $v \neq q$.

3.1 All Graphs with Sinks Stabilize

Definition 17 (Chip-Firing on a Graph with a Sink)**.** The *chip-firing process on a graph with a sink* on a finite graph G with sink q starts with an initial chip configuration $\mathbf{c}$, and at each step selects and fires a **non-sink** vertex which is ready to fire. If a stable configuration is reached, then the process stops.

Proposition 8. *Let G be a finite graph with a sink q. Then every initial configuration $\mathbf{c}$ on G eventually reaches a stable configuration.*

Proof. If the chip-firing process doesn't stabilize, then every vertex fires infinitely many times (as discussed earlier). In particular, every neighbor of the sink fires infinitely many times. But each time this happens, one chip moves into the sink, decreasing the number of chips on firing vertices. Since there are only finitely many chips on G, this is a contradiction; eventually there would not be enough chips to fire. □

3.2 Critical Configurations

Definition 18 (Criticality)**.** A configuration $\mathbf{c}$ is *critical* if it is stable, and reachable from a *sufficiently large* initial configuration $\mathbf{b}$, where a configuration is sufficiently large if every non-sink vertex is ready to fire: $b_v \geq \deg(v)$ for all $v \neq q$.

The critical configurations can be roughly thought of as the "interesting" stable configurations.

3.3 Firing Equivalence

First, let's consider the idea of firing equivalence on graphs without sinks:

Definition 19 (Firing Equivalence)**.** Let G be a graph on n vertices with graph Laplacian Δ. Two integer sequences $\mathbf{c}$ and $\mathbf{d}$ in $\mathbb{Z}^n$ are *firing-equivalent* on G, denoted $\mathbf{c} \sim \mathbf{d}$, if $\mathbf{d} - \mathbf{c} = \Delta\mathbf{z}$ for some $\mathbf{z} \in \mathbb{Z}^n$. The equivalence class of $\mathbf{c}$ is denoted $[c]$ and called the *firing-class of* $\mathbf{c}$.

Hence the firing classes on G are the elements of the cokernel of the map $\Delta : \mathbb{Z}^n \to \mathbb{Z}^n$.

Now, to define firing equivalence on graphs with sinks, we need a way to consider a Laplacian which somehow encodes the idea that the sink is not allowed to fire. Luckily, this is simple:

Definition 20 (Reduced Laplacian)**.** Let Δ be the Laplacian of a graph G with sink q. Then, the *reduced graph Laplacian of G with respect to q*, denoted Δ_q, is the matrix obtained from Δ by deleting the row and column corresponding to q. Similarly, a configuration $\mathbf{c}$ is said to "have its values at the sink suppressed" if the row corresponding to q is deleted.

Recall that since we usually say that a graph with sink has $n+1$ vertices, Δ_q will still be an $n \times n$ matrix.

Also, recall that Δ is an $(n+1) \times (n+1)$ matrix of rank n (since we assume that the graph G is connected), with kernel spanned by $\mathbf{1} = (1, \ldots, 1)$. Therefore, deleting a row and column to get Δ_q will eliminate the kernel; hence Δ_q has rank n. In particular, $\Delta_q : \mathbb{Z}^n \to \mathbb{Z}^n$ is injective.

Definition 21 (Firing Equivalence on Sinks)**.** Let G be a graph with $n+1$ vertices and sink q. Then, if $\mathbf{c}$ and $\mathbf{d}$ are configurations with their values at the sink suppressed, then $\mathbf{c}$ and $\mathbf{d}$ are *firing-equivalent*, denoted $\mathbf{c} \sim \mathbf{d}$, if $\mathbf{d} - \mathbf{c} = \Delta_q \mathbf{z}$ for some $\mathbf{z} \in \mathbb{Z}^n$. The equivalence class of $\mathbf{c}$ is denoted $[c]$ and called the *firing-class of* $\mathbf{c}$. Again, the firing classes on G are the elements of the cokernel of the map $\Delta_q : \mathbb{Z}^n \to \mathbb{Z}^n$.

Since $\Delta_q : \mathbb{Z}^n \to \mathbb{Z}^n$ is injective, it has finite cokernel by the structure theorem for finite Abelian groups; therefore there are finitely many firing-classes on a graph with sink.

Theorem 9. *For a graph G with sink q and reduced Laplacian Δ_q, there exists a unique critical configuration per firing class (equiv. per element of the cokernel of Δ_q).*

Proof. Assume G has $n+1$ vertices, so that $\Delta_q : \mathbb{Z}^n \to \mathbb{Z}^n$ is injective. We prove the following lemma:

Lemma 10. *There exists a vector $\mathbf{z} \in \mathbb{Z}^n$ such that $\Delta_q \mathbf{z} > \mathbf{0}$; that is, each coordinate of $\Delta_q \mathbf{z}$ is positive.*

Proof. First, notice that since $\Delta_q : \mathbb{Z}^n \to \mathbb{Z}^n$ is injective, it extends naturally to an invertible map $\Delta_q : \mathbb{Q}^n \to \mathbb{Q}^n$ with the same matrix representation. Therefore, there exists a rational matrix Δ_q^{-1}. Now let $\mathbf{1}$ be the all ones vector, and define $\mathbf{g} = \Delta_q^{-1}\mathbf{1}$. Since Δ_q^{-1} is rational, $\mathbf{g}$ may not be an integer vector. Let $\lambda \in \mathbb{Z}^+$ be sufficient to clear denominators so that $\mathbf{z} = \lambda \mathbf{g}$ is an integer vector. But then

$$\Delta_q \mathbf{z} = \Delta_q \lambda \mathbf{g} = \lambda \Delta_q \mathbf{g} = \lambda \Delta_q \Delta_q^{-1} \mathbf{1} = \lambda \mathbf{1} \geq \mathbf{1} > \mathbf{0}.$$

□

The above lemma will be used to prove existence. For uniqueness, we will prove the following:

Lemma 11. *Suppose $\mathbf{c}$ and $\mathbf{d}$ are two firing-equivalent chip configurations. Then there exists a chip configuration $\mathbf{b}$ that legally fires to $\mathbf{c}$ and $\mathbf{d}$.*

Proof. By assumption, there exists $\mathbf{z}$ such that $\mathbf{d} - \mathbf{c} = \Delta_q \mathbf{z}$. Then define $I = \{i \mid z_i \geq 0\}$ and $J = \{j \mid z_j < 0\}$. One can imagine I to be the "positive part" of $\mathbf{z}$ and J is the "negative part" of $\mathbf{z}$. Note that I and J are subsets of $\{1, \ldots, n\}$. Now, clearly

$$\mathbf{d} - \mathbf{c} = \sum_{i \in I} z_i \Delta_q e_i + \sum_{j \in J} z_j \Delta_q e_j.$$

Hence by reorganizing

$$\mathbf{d} + \sum_{i \in I} z_i \Delta_q e_i = \mathbf{c} - \sum_{j \in J} z_j \Delta_q e_j$$

and if we define $\mathbf{b}$ to be equal to the above vector, then $\mathbf{b}$ legally fires to both $\mathbf{c}$ and $\mathbf{d}$. □

Now we are prepared to show the desired result.

Existence of a critical configuration in each equivalence class: Let $\mathbf{z} \in \mathbb{Z}^n$ be such that $\Delta_q \mathbf{z} > 0$ (via the lemma above). Then, clearly $\mathbf{c}_t = \mathbf{c} + t\Delta_q \mathbf{z}$ is firing-equivalent to $\mathbf{c}$ for any t (since $\mathbf{c}_t - \mathbf{c} = \Delta_q t\mathbf{z}$), and is sufficiently large for sufficiently large t. Then $\text{stab}(\mathbf{c}_t)$ is critical and an element of $[\mathbf{c}]$.

Uniqueness of a critical configuration in each equivalence class: Suppose $\mathbf{c}$ and $\mathbf{d}$ are firing-equivalent critical configurations. Then, by the above lemma, there exists a configuration $\mathbf{b}$ which legally fires to both $\mathbf{c}$ and $\mathbf{d}$. Hence, since both $\mathbf{c}$ and $\mathbf{d}$ are stable, by global confluence they must be equal. □

Corollary 11.1. *On any graph G with sink q, there are only finitely many critical configurations.*

3.4 Dhar's Burning Algorithm

Definition 22 (Superstable)**.** A configuration is *superstable* if it has no nontrivial legal cluster-fires.

Suppose G has n vertices. For determining if a chip configuration $\mathbf{c}$ is stable we need only check if n different firings are legal. However, determining if a chip configuration is *superstable* is much harder, as there are 2^n different cluster-firings. The Burning Algorithm provides a less naive method for checking superstability, using the following argument.

Definition 23 (Burning Algorithm)**.** Let G be a finite graph with n vertices and sink q. Let $\mathbf{c}$ be a chip configuration on $G \setminus q$.

Envision the chips as firefighters protecting their location. Imagine a fire burning through the graph – if a vertex v is on fire then the fire spreads along all incident edges towards the neighbors of v. Each firefighter can turn and stop the fire along exactly one edge. Thus, a site is protected as long as there are at least as many firefighters (chips) as there are incident burning edges. If a site has more burning incident edges than firefighters, then the site catches on fire and the fire spreads through towards all of its neighbors.

Theorem 12. *Let G be a finite graph with sink q. Start a fire at q. A configuration is superstable if and only if every vertex is eventually on fire.*

Now, the only way for the system to change is for a new vertex to catch fire, and this can happen at most n times. Hence we only need to make n checks, and each check only needs to consider at most n vertices (to see if any of them have caught fire). Therefore, our method requires on the order of n^2 steps, making it substantially more efficient than the 2^n steps that the naive method uses.

4 Sandpile Groups

First, consider the following observation about stabilization:

Proposition 13. *For any chip configurations $\mathbf{c}$ and $\mathbf{d}$, we have*

$$\text{stab}(\mathbf{c} + \mathbf{d}) = \text{stab}(\text{stab}(\mathbf{c}) + \mathbf{d}).$$

Proof. This follows directly from global confluence. □

Proposition 14. *Let G be a graph with sink q and let $\mathbf{c}$ and $\mathbf{d}$ be two critical configurations of G. Then* $\text{stab}(\mathbf{c} + \mathbf{d})$ *is also a critical configuration of G.*

Proof. Clearly $\text{stab}(\mathbf{c} + \mathbf{d})$ is a stable configuration. Furthermore, since $\mathbf{c}$ is critical, then there exists $\mathbf{c}'$ sufficiently large, such that $\text{stab}(\mathbf{c}') = \mathbf{c}$. But then

$$\text{stab}(\mathbf{c} + \mathbf{d}) = \text{stab}(\text{stab}(\mathbf{c}') + \mathbf{d}) = \text{stab}(\mathbf{c}' + \mathbf{d})$$

and $\mathbf{c}' + \mathbf{d}$ is clearly sufficiently large, so $\text{stab}(\mathbf{c} + \mathbf{d})$ can be reached from a sufficiently large configuration. Since $\text{stab}(\mathbf{c}+\mathbf{d})$ is stable and reachable from a sufficiently large initial criterion, by definition it is critical. □

Definition 24 (Sandpile Sum)**.** The sandpile sum of two critical configurations $\mathbf{c}$ and $\mathbf{d}$ is defined as $\mathbf{c} \oplus \mathbf{d} = \text{stab}(\mathbf{c} + \mathbf{d})$. As proven above, this is another critical configuration.

4.1 The Sandpile (Semi)group

Definition 25 (Semigroup)**.** A *semigroup* $(G, \cdot)$ (denoted by G using abuse of notation) is a set G equipped with an associative binary operation $\cdot$, that is, a binary operation such that $a \cdot (b \cdot c) = (a \cdot b) \cdot c$ for any $a, b, c \in G$. Essentially, a semigroup is a group without a required identity or inverse.

Definition 26 (Sandpile Group)**.** Given a graph G with sink q, the *sandpile group* $\mathcal{S}(G)$ on G is the semigroup of critical configurations operated on by the sandpile sum. It is clear that this operation is associative by the associativity of addition and Proposition 13.

The reason why we define $\mathcal{S}(G)$ to be a semigroup instead of a group (despite the fact that we call it the sandpile *group*) is that it's not intuitively clear that there is a critical configuration which acts as the identity, or that every critical configuration has an inverse critical configuration.

Notice two key facts about the sandpile (semi)group:

1. The sandpile (semi)group is finite, because there are only finitely many critical configurations on a graph (see Corollary 11.1).
2. The sandpile (semi)group is commutative, because addition of integer vectors is commutative.

4.2 Defining the Sandpile Group using Cokernels

As we've seen, our definition of the sandpile group has some flaws: namely, it's not clear that the sandpile group is even a group. Therefore, we'll define the sandpile group using another method, one which makes it clear that the sandpile group is indeed a group.

Definition 27 (Sandpile Group 2)**.** Let G be a graph with $n+1$ vertices and sink q. Let $\Delta_q : \mathbb{Z}^n \to \mathbb{Z}^n$ be the linear transformation given by the action of the reduced Laplacian Δ_q on the canonical basis of $\mathbb{Z}^n$. Then the sandpile group $\mathcal{S}(G)$ is defined to be $\operatorname{coker} \Delta_q = \mathbb{Z}^n / \operatorname{im} \Delta_q$.

Clearly, this is an abelian group; it is the quotient of two finite groups. Yet this definition is identical to our definition from the earlier section; to see why, recall that there is a unique critical configuration per firing class. Therefore, the map $\mathcal{S}(G) \to \operatorname{coker} \Delta_q$ given by sending a critical configuration $\mathbf{c}$ to its firing class $[\mathbf{c}]$ is an isomorphism. Hence $\operatorname{coker} \Delta_q \simeq \mathcal{S}(G)$, implying that $\mathcal{S}(G)$ is also a group whose identity is the unique critical configuration which is firing-equivalent to $\mathbf{0}$. Similarly, the inverse of a critical configuration $\mathbf{c}$ is the unique critical configuration firing-equivalent to $-\mathbf{c}$.

4.3 Computing the Sandpile Group: the Smith Normal Form

Definition 28. For a non-singular integer matrix M, the *Smith normal form* of M is a diagonal matrix A suh that

$$A = \operatorname{diag}(d_1, d_1 d_2, \ldots, d_1 \ldots d_n) = PMQ$$

for invertible matrices $P, Q \in \mathrm{GL}(n, \mathbb{R})$ and integers $d_i \in \mathbb{Z}$.

A standard result from linear algebra is that by adding integer multiples of rows and columns or inverting the signs of rows or columns, we can turn any non-singular matrix M into its Smith normal form. Notice that none of these operations change the image of M, and therefore do not affect the cokernel of M (either as a rational matrix associated to a map $\mathbb{Q}^n \to \mathbb{Q}^n$ or an integer matrix associated to a map $\mathbb{Z}^n \to \mathbb{Z}^n$).

Proposition 15. *If M is a non-singular $n \times n$ integer matrix and the Smith normal form of M is diag$(e_1, \ldots, e_n)$, then the cokernel of diag$(e_1, \ldots, e_n)$ is isomorphic to $(\mathbb{Z}/e_1\mathbb{Z}) \oplus \cdot \oplus (\mathbb{Z}/e_n\mathbb{Z})$, implying*

$$\operatorname{coker}_{\mathbb{Z}}(M) \simeq (\mathbb{Z}/e_1\mathbb{Z}) \oplus \cdot \oplus (\mathbb{Z}/e_n\mathbb{Z}).$$

In particular we can apply this theorem to $M = \Delta_q$ to compute the sandpile group.

4.4 Computing the Identity of the Sandpile Group

Proposition 16. *The identity element $\mathbf{c}_{\mathrm{id}}$ of the sandpile group of critical elements is given by*

$$\mathbf{c}_{\mathrm{id}} = \operatorname{stab}(2\mathbf{c}_{\max} - \operatorname{stab}(2\mathbf{c}_{\max})).$$

Proof. The configuration $\mathbf{c}_{\mathrm{id}}$ is (1) critical and (2) clearly firing-equivalent to the all zeroes configuration, since $\mathbf{c}_{\mathrm{id}} = \operatorname{stab}(2\mathbf{c}_{\max} - \operatorname{stab}(2\mathbf{c}_{\max}))$ is firing-equivalent to $2\mathbf{c}_{\max} - \operatorname{stab}(2\mathbf{c}_{\max})$, which is firing-equivalent to $2\mathbf{c}_{\max} - 2\mathbf{c}_{\max} = \mathbf{0}$. Since critical configurations are unique per firing equivalence class and $\mathbf{0}$ belongs to the identity class of $\operatorname{coker}_{\mathbb{Z}}(\Delta_q)$, this is must be the identity configuration. □

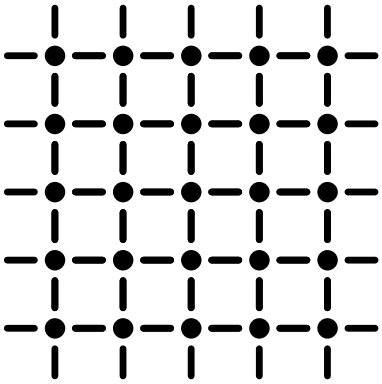
4.5 Example: Identities of Sandpile Groups as Raster Images

Finally we are prepared to discuss the images presented on my website here.

First, let's define a "grid graph" which naturally makes a raster image into a graph.

Definition 29 (Grid Graph)**.** The $n \times n$ *grid graph* is defined to be the graph G with $n^2 + 1$ vertices. n^2 of these vertices are arranged in a grid, with vertices that are horizontally or vertically adjacent connected by an edge. The final vertex is the sink vertex q of G, and every boundary edge is connected to the sink either once (for non-corner vertices) or twice (for corner vertices), so that every non-sink vertex of G has degree 4.

For example, following is the 5×5 grid graph (the sink vertex is not pictured, but can be envisioned as the endpoint of all edges with just one endpoint):



An $n \times n$ raster image can be interpreted as an $n \times n$ grid graph by letting each pixel be the corresponding vertex and letting the color of the pixel correspond to how many chips are on the vertex. Then, using the formula from Proposition 16, we can compute the identity element of the grid graph using the code here.

On the next page are the identity elements of grid graphs of various sizes and colored with various color schemes (first the standard scheme from Klivan's textbook, a scheme which I chose, and then a scheme selected by my friend). The following legend describes these color schemes:

- *Color Scheme 1:* Blue (0 grains of sand), Light Blue (1 grain of sand), Yellow (2 grains of sand), Maroon (3 grains of sand).
- *Color Scheme 2:* Neon Blue (0 grains of sand), Medium Blue (1 grain of sand), Dark Blue (2 grains of sand), Midnight Blue (3 grains of sand).
- *Color Scheme 3:* Neon Pink (0 grains of sand), Light Purple (1 grain of sand), Dark Purple (2 grains of sand), Midnight Blue (3 grains of sand).

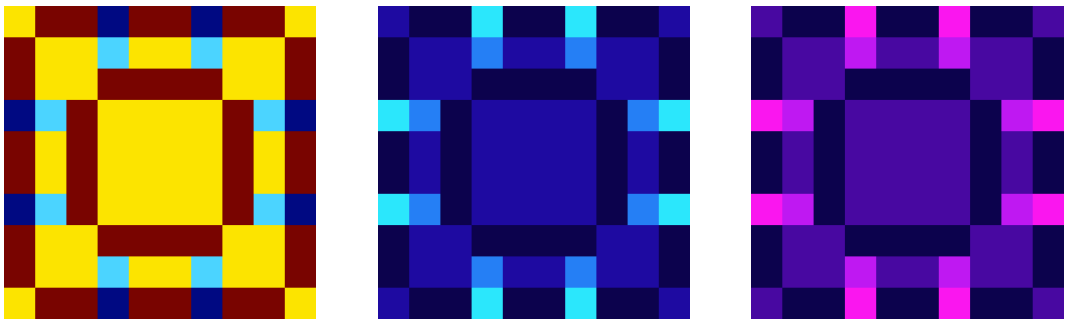


Figure 1: 10×10 Identity Elements

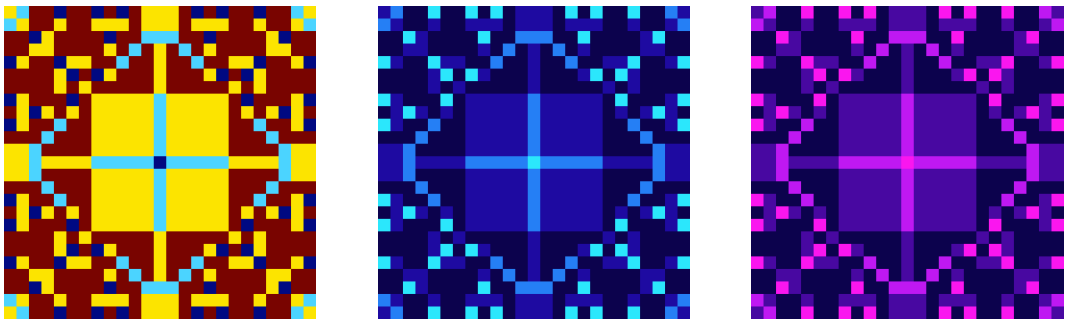


Figure 2: 25×25 Identity Elements

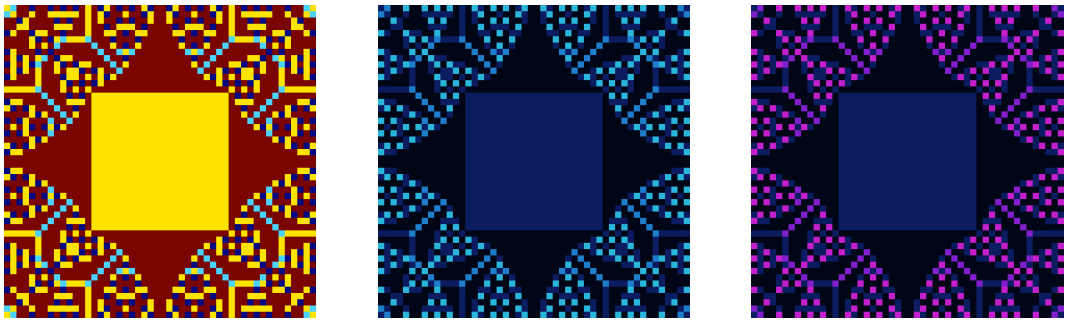


Figure 3: 50×50 Identity Elements

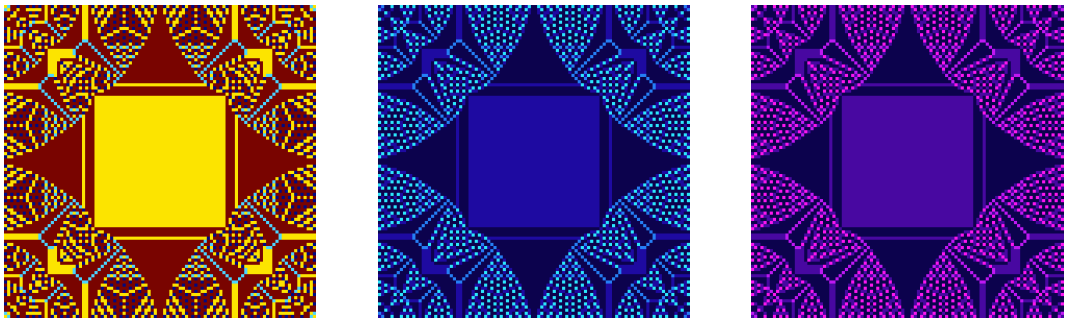


Figure 4: 100×100 Identity Elements

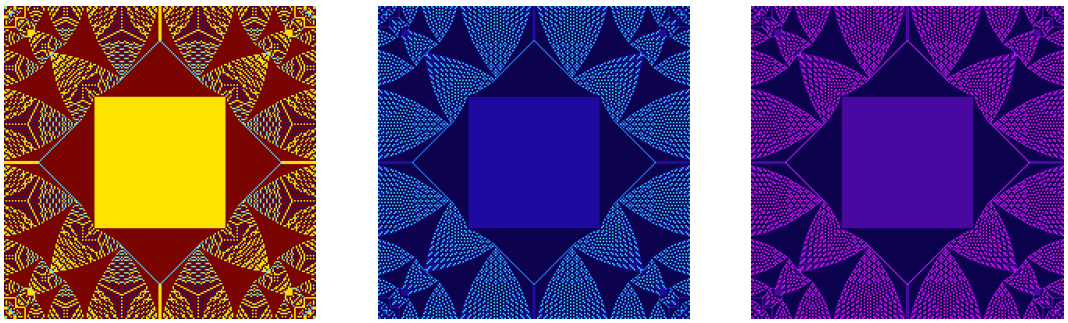


Figure 5: 200×200 Identity Elements

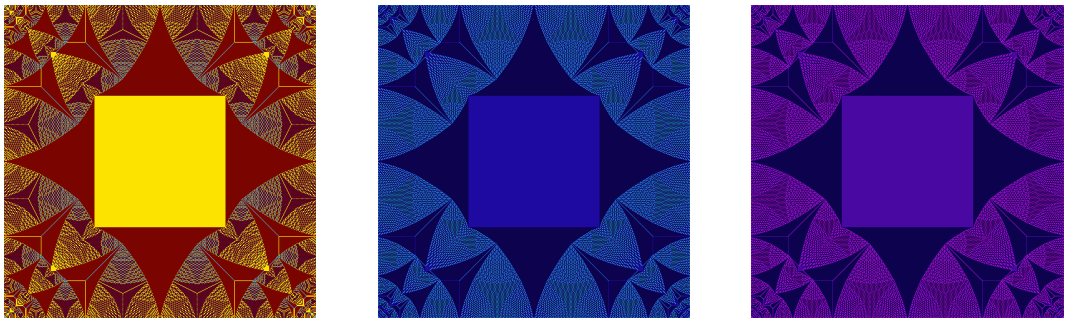


Figure 6: 500×500 Identity Elements

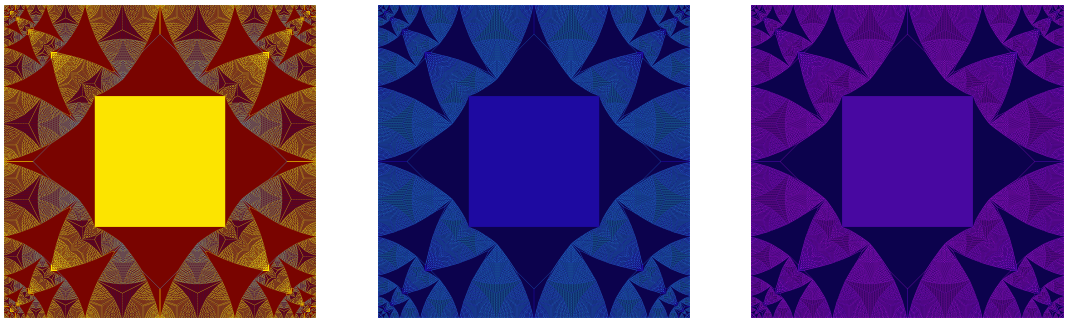


Figure 7: 1000×1000 Identity Elements

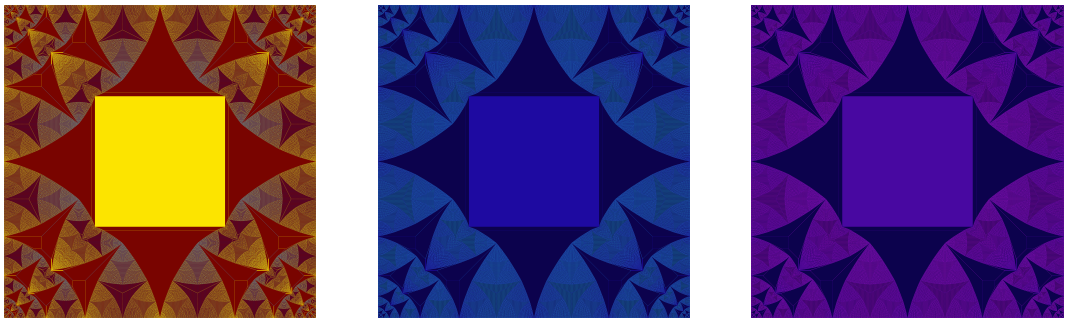


Figure 8: 2000×2000 Identity Elements