ABELIAN NETWORKS III. THE CRITICAL GROUP

BENJAMIN BOND AND LIONEL LEVINE

ABSTRACT. The critical group of an abelian network is a finite abelian group that governs the behavior of the network on large inputs. It generalizes the sandpile group of a graph. We show that the critical group of an irreducible abelian network acts freely and transitively on recurrent states of the network. We exhibit the critical group as a quotient of a free abelian group by a subgroup containing the image of the Laplacian, with equality in the case that the network is rectangular. We generalize Dhar’s burning algorithm to abelian networks, and estimate the running time of an abelian network on an arbitrary input up to a constant additive error.

1. INTRODUCTION

Associated to a finite connected graph $G$ with marked vertex $s$ is a finite abelian group called its sandpile group. This group arose independently in three different fields: the Neron model of a curve in arithmetic geometry [Lor89, Lor91], the abelian sandpile model in statistical physics [Dha90], and discrete potential theory on graphs [Big99]. In this paper we are going to exhibit the sandpile group as an instance of a more general construction, the critical group $\text{Crit}_N$ of an abelian network $N$.

Deepak Dhar [Dha99] proposed abelian networks as models of self-organized criticality in physics, generalizing the abelian sandpile model of [BTW87, Dha90]. From the point of view of computer science, abelian networks are an interesting class of automata networks because they can compute asynchronously: an abelian network produces the same final output in the same number of steps regardless of the order of events at individual nodes of the network.

Many examples of abelian networks are surveyed in [BL13]. They include sandpile and rotor networks and their non-unary cousins, oil and water networks and abelian mobile agents. Besides the sandpile groups, critical groups of abelian networks have been studied in a few other particular cases: rotor networks [PDDK96] and height-arrow networks [DR04]. The group $\text{Crit}_\text{Rotor}(G,s)$ associated to a rotor network with a sink turns out to be isomorphic to the sandpile group, $\text{Crit}_\text{Sand}(G,s)$ [PDDK96, LL09]. We will see this isomorphism as a case of a
more general phenomenon: homotopic abelian networks have isomorphic critical groups (Corollary 3.12).

The sandpile group has several different constructions. Babai and Toumpakari [BT10] realized the sandpile group as the minimal ideal of a commutative monoid. Their approach, developed further in [C+13], is well-suited for generalizing to abelian networks. In §1.2 we review the small amount of monoid theory we will need. In §2 we recall the definition of an abelian network and relevant results from [BL13, BL14], and prove a few basic lemmas including a local-to-global principle for irreducibility, Lemma 2.6.

1.1. Main results. The setting for all of our main results is a finite irreducible abelian network $N$ that halts on all inputs. In §3.1 we define the critical group $\text{Crit} N$ and the set $\text{Rec} N$ of recurrent states, and show that the former acts freely and transitively on the latter. In §3.2 we consider a Markov chain defined by sending random input to the network; we relate the algebraic and probabilistic definitions of “recurrent,” and show that the stationary distribution of the chain is uniform on recurrent states. In §3.3 we find the expected time for $N$ to halt on a given input to a uniform random recurrent state. In §3.4 we give generators and relations for the critical group, and in §3.5 we compute its order.

In §4 we estimate the time for $N$ to halt on an arbitrary input, up to a constant additive error.

In §5.1 we give an efficient test for whether a given state of $N$ is recurrent, generalizing Dhar’s burning algorithm [Dha90] and Speer’s script algorithm [Spe93]. We define a burning element to be an input that returns the network to its initial state if and only if that state is recurrent. We show that any integer vector satisfying a certain set of linear inequalities is a burning element. In §5.2 we give an efficient way to find such a vector.

1.2. Review of monoid actions. Here we recall a few facts about actions of finite commutative monoids, which were proved in [BL14].

Let $M$ be a commutative monoid, $X$ a set and $\mu : M \times X \to X$ a monoid action. Commutativity of $M$ implies that the relation $\sim$ on $X$ defined by

$$x \sim x' : \exists m, m' \in M \text{ such that } mx = m'x'$$

is an equivalence relation. We say that $\mu$ is irreducible if $\sim$ has only one equivalence class.

For the rest of this section we assume that $M$ is finite and $\mu$ is irreducible. The minimal idempotent $e := \prod_{f \in M, f \neq e} f$ is the unique idempotent accessible from all of $M$ (that is, $ee = e$ and $e \in Mm$ for all $m \in M$).

**Lemma 1.1.** $eM$ is an abelian group with identity element $e$.

**Lemma 1.2.** (Recurrent Elements Of An Irreducible Monoid Action) The following are equivalent for $x \in X$:

1. $x \in My$ for all $y \in X$
2. $x \in M(mx)$ for all $m \in M$
3. $x \in mX$ for all $m \in M$
An element $x \in X$ is called \textit{recurrent} if it satisfies the equivalent conditions of Lemma 1.2. To explain this terminology, suppose we are given a probability distribution $\alpha$ on $M$ such that $\{ m \in M : \alpha(m) > 0 \}$ generates $M$ as a monoid, and consider the Markov chain on $X$ that transitions from $x$ to $mx$ with probability $\alpha(m)$. The states $x$ satisfying the equivalent conditions of Lemma 1.2 are precisely those visited infinitely often by the Markov chain (see §3.2).

We say that $m \in M$ \textit{acts invertibly} on a subset $Y$ of $X$ if the map $y \mapsto my$ is a permutation of $Y$.

\textbf{Lemma 1.3.} Every $m \in M$ acts invertibly on $eX$.

We say that a monoid action $\mu : M \times X \to X$ is \textit{faithful} if there do not exist distinct elements $m, m' \in M$ such that $mx = m'x$ for all $x \in X$.

Let $G$ be a group with identity element $e$. Recall that a group action $G \times Y \to Y$ is called \textit{transitive} if $Gy = Y$ for all $y \in Y$, and is called \textit{free} if for all $g \neq e$ there does not exist $y \in Y$ such that $gy = y$. If the action is both transitive and free, then for any two elements $y, y' \in Y$ there is a unique $g \in G$ such that $gy = y'$; in particular, $\#G = \#Y$.

\textbf{Theorem 1.4.} (Group Actions Arising From Monoid Actions) The restriction of $\mu$ to $eM \times eX$ is a transitive group action

$$e\mu : eM \times eX \to eX.$$ 

In addition, if $\mu$ is faithful, then $e\mu$ is free.

## 2. Abelian networks

We now recall the definition and basic properties of abelian networks, refering the reader to [BL13, BL14] for details.

### 2.1. Abelian processors

Let $Q$ be a set and $\text{End}(Q)$ the monoid of all set maps $Q \to Q$ with the operation of composition. An \textit{abelian processor} with input alphabet $I$ and state space $Q$ is a collection of maps $(t_a)_{a \in I}$ where each $t_a \in \text{End}(Q)$, such that $t_a t_b = t_b t_a$ for all $a, b \in I$.

### 2.2. Abelian processors with message passing

An \textit{abelian processor with output alphabets} $(A_u)_{a \in U}$ has in addition to the state transition maps $t_a$ a message passing function $\alpha^u_a : Q \to A^*_u$ for each $a \in I$ and $u \in U$. Here $A^*$ denotes the free monoid of all finite words in an alphabet $A$. These functions are required to satisfy a commutativity condition: namely, if two input words $w, w' \in I^*$ are equal up to permutation, then for each $u \in U$ the resulting output words in $A^*_u$ are equal up to permutation.
2.3. Abelian networks. An abelian network on a directed graph \( G = (V, E) \) with alphabet \( A = \bigcup_{v \in V} A_v \) and state space \( Q = \prod_{v \in V} Q_v \) is a collection \( (P_v)_{v \in V} \), where each \( P_v \) is an abelian processor with input alphabet \( A_v \), state space \( Q_v \) and output alphabets \( A_u \) for each edge \((v, u) \in E\). When \( P_v \) in state \( q \in Q_v \) processes letter \( a \in A_v \), it transitions to state \( t_a(q) \) and sends the message \( o^u_a(q) \) to each neighboring processor \( P_u \). Note that \( o^u_a(q) \) might be the empty word, which signifies that no message is sent.

The total state of an abelian network \( N = (P_v)_{v \in V} \) is described by an element \( q \in Q \) indicating the internal states of all processors, together with a vector \( x \in \mathbb{Z}^A \) indicating how many letters of each type are waiting to be processed. We use the notation \( x.q \) for this pair. Note that \( A \) is a disjoint union, so each letter belongs to the input alphabet of a unique processor.

2.4. Sandpiles, rotor networks, toppling networks. These will be our running examples of abelian networks. Let \( G = (V, E) \) be a finite directed graph and \( s \in V \) a vertex such that from every other vertex \( v \in V \) there is a directed path from \( v \) to \( s \). Each processor \( P_v \) has alphabet \( A_v = \{v\} \) and state space \( Q_v = \mathbb{Z}/r_v\mathbb{Z} \) for a positive integer \( r_v \) called the threshold of \( v \). The state transition is \( t_v(q) = q + 1 \mod r_v \).

It remains to describe the message passing. In the networks Sand\((G, s)\) and Rotor\((G, s)\), processor \( P_s \) is a sink (which means it has just one state and never passes any messages) and we take \( r_v = d_v \), the outdegree of \( v \), for all \( v \neq s \). In the sandpile network Sand\((G, s)\), whenever processor \( P_v \) transitions from state \( d_v - 1 \) to state 0 it passes \( d_v \) letters: one letter \( u \) along each outgoing edge \((v, u)\). The message passing for the rotor network Rotor\((G, s)\) is specified by fixing an ordering \( e_1, \ldots, e_{d_v} \) of the outgoing edges from \( v \). Whenever it transitions from state \( q \) to \( q + 1 \mod d_v \), processor \( P_v \) passes exactly one letter, along the edge \( e_{q+1} \).

Toppling networks are a generalization of sandpiles, where we allow \( r_v \neq d_v \); however, when processor \( P_v \) in a toppling network transitions from state \( r_v - 1 \) to state 0, it passes \( d_v \) letters just as in a sandpile network: one letter \( u \) along each outgoing edge \((v, u)\). The importance of the sink \( s \) in Sand\((G, s)\) and Rotor\((G, s)\) is to ensure the network halts on all inputs. Depending on the thresholds \( r_v \), a toppling network may halt on all inputs even if no vertex is a sink.

Rotor and toppling networks are unary, meaning that each alphabet \( A_v \) is a singleton. See [BL13] for two examples of non-unary abelian networks, oil and water and abelian mobile agents.

2.5. Executions. An execution is a finite word \( w \in A^\ast \). It prescribes an order in which letters are to be processed. We write \( \pi_w(x.q) \) for the result of executing \( w \) starting from \( x.q \); this is another pair \( x'.q' \) that can be computed using the state transitions and message passing functions. It is important to note that some coordinates of \( x' \) may be negative! For example, if \( w \) consists of a single letter \( a \) and \( x_a = 0 \), and processing \( a \) does not cause any letters \( a \) to be passed, then \( x'_a = -1 \). The interpretation is that a processor was instructed to process letter \( a \) even though no letter \( a \) was present; the processor follows the instruction and
keeps track of the “debt” that it is owed one letter \(a\). Note however that messages passed from one processor to another are always nonnegative.

The axioms of an abelian network imply that \(\pi_w(x,q)\) depends only on the vector \(|w| \in \mathbb{N}^A\) where \(|w|_a\) is the number of letters \(a\) in \(w\). We write \(\pi_w\) and \(\pi_{|w|}\) interchangeably. Writing \(w = a_1 \cdots a_r\) and \(\pi_{a_1 \cdots a_i}(x,q) = x^i.q^i\), we say that \(w\) is legal for \(x,q\) if \(x^i_{a_i^{-1}} \geq 1\) for all \(i = 1, \ldots, r\). We say that \(w\) is complete for \(x,q\) if \(x^r = 0\) (inequalities between vectors are coordinatewise). In words, a legal execution is one that incurs no “debts,” and a complete execution is one that removes all letters from the network. Note that if \(w\) is both legal and complete, then \(x^i \geq 0\) for \(i = 1, \ldots, r-1\) and \(x^r = 0\).

The least action principle [BL13, Lemma 4.3] says that if \(w\) is any legal execution for \(x,q\) and \(w'\) is any complete execution for \(x,q\), then \(|w| \leq |w'|\). It follows that \(|w|\) is the same for all complete legal executions of \(x,q\). If there exists a complete legal execution for \(x,q\) then we say that \(N\ halts\) on input \(x,q\). In this case the odometer \([x,q]\) is defined as
\[
[x,q] := |w|
\]
where \(w\) is any complete legal execution for \(x,q\). The odometer is a vector in \(\mathbb{N}^A\) whose \(a\)th coordinate is the total number of letters \(a\) processed.

The final state \(q^f_v\) of each processor \(P_v\) can be determined from its initial state \(q_v\) and the odometer coordinates \([x,q]_a\) for \(a \in A_v\), namely
\[
q^f_v = \left(\prod_{a \in A_v} t^{[x,q]}_a\right) q_v.
\]

where the product denotes composition of the commuting maps \(t_a\). To make its dependence on \(x\) explicit we will use the notation \(x \gg q\) for the final state \(q^f\).

Like the odometer \([x,q]\), the state \(x \gg q\) depends only on \(x\) and \(q\) and not on the choice of complete legal execution.

2.6. The global action. Let \(N\) be an abelian network that halts on all inputs: that is, \(x,q\) has a finite complete legal execution for all \(x \in \mathbb{N}^A\) and all \(q \in Q\). We will see shortly that \((x,q) \mapsto x \gg q\) defines a monoid action \(\mathbb{N}^A \times Q \to Q\). This global action is the main object of interest in the present paper: we will use this action to define the critical group in §3 and characterize the recurrent states of this action in §5. We have chosen the notation \(\gg\) to distinguish the global action from the local action \(\triangleright\) of [BL14]; below we recall how \(\triangleright\) is defined and relate the two actions.

Lemma 2.1. If \(w\) is a legal execution from \(x,q\) to \(y,r\), then
\[
x \gg q = y \gg r
\]
and
\[
[x,q] = |w| + [y,r].
\]

Proof. Let \(w'\) be a complete legal execution for \(y,r\). Then the concatenation \(ww'\) is a complete legal execution for \(x,q\). \(\square\)

The next lemma verifies that \(\gg\) defines a monoid action of \(\mathbb{N}^A\) on \(Q\).
Lemma 2.2. For $x, y \in \mathbb{N}^A$ and $q \in Q$ we have
\[(x + y) \ggq q = y \ggq (x \ggq q)\]
and
\[[(x + y).q] = [x.q] + [y.(x \ggq q)].\]

Proof. If $w$ is a complete legal execution for $x.q$, then $w$ is a legal execution from $(x + y).q$ to $y.(x \ggq q)$. Since $|w| = |x.q|$ the result follows from Lemma 2.1. □

2.7. The local action. In [BL14] we defined a monoid action $\gg$ of $\mathbb{N}^A$ on $\mathbb{Z}^A \times Q$,
\[x \gg (y.q) = \pi_x((x + y).q).\]
This is called the local action because each processor $P_v$ processes only the letters that were added at $v$ (namely $x_a$ letters $a$ for each $a \in A_v$) and not any additional letters passed from other processors. We write $x \gg q$ as a shorthand for $x \gg (0.q)$.

The next lemma relates the local and global actions.

Lemma 2.3. If $x \gg q = y.r$, then $x \ggg q = y \ggg r$ and $[x.q] = x + [y.r]$.

Proof. If $x \gg q = y.r$ then there is a legal execution $w$ from $x.q$ to $y.r$ with $|w| = x$, so the result follows from Lemma 2.1. □

One way to compute the global action $\ggg$ is by iterative application of $\gg$: given $x \in \mathbb{N}^A$ and $q \in Q$, set $x_0 = x$ and $q_0 = q$ and
\[x_n.q_n = x_{n-1} \ggq q_{n-1}\]
for $n \geq 1$. This amounts to making a particular choice of execution, called parallel update. In parallel update the execution occurs in rounds $n = 1, 2, \ldots$. At the beginning of round $n$ there are $(x_n)_a$ letters $a$ waiting to be processed for each $a \in A$. During round $n$ we execute a word $w_n$ with $|w_n| = x_n$. The concatenation $w = w_1w_2\cdots$ is a legal execution. Since $N$ halts on all inputs, every legal execution is finite, so there is some $N$ such that all words $w_n$ for $n > N$ are empty. Then $x_n = 0$ for all $n \geq N$, and $w = w_1 \cdots w_N$ is a finite complete legal execution. In particular, the final state is given by
\[x \ggg q = q_N\]
and the odometer is
\[|x.q| = |w| = \sum_{n=1}^{N} x_n.\]

We record a few more identities to be used later. We extend the domain of $\ggg$ to $\mathbb{N}^A \times Q$ by defining for $y \in \mathbb{N}^A$
\[x \ggg (y.q) := (x + y) \ggq q.\]

Lemma 2.4. Given $x \in \mathbb{N}^A$ and $q \in Q$, let $k = [x.q]$. Then
\[(i) \quad \pi_k(x.q) = 0.(x \ggq q)\]
\[(ii) \quad k \gg (x.q) = k.(x \ggq q)\]
\[(iii) \quad x \ggg (y.q) = (x + y) \ggq q\]
Proof. Part (i) follows from the definition of $x \triangleright \triangleright q$.

Part (ii) follows from (i) since

$$k \triangleright (x \triangleright q) = \pi_k ((x + k) \cdot q) = k \cdot (x \triangleright \triangleright q).$$

To prove part (iii), both states $x \triangleright \triangleright (y \triangleright q)$ and $(x + y) \triangleright \triangleright q$ are the result of performing a complete legal execution for $(x + y) \cdot q$, and any two complete legal executions for $(x + y) \cdot q$ result in the same final state. □

2.8. Local monoids. The transition monoid of an abelian processor with state space $Q$ and transition maps $t_a : Q \rightarrow Q$ is the submonoid $M = \langle t_a \rangle_{a \in A} \subseteq \mathrm{End}(Q)$, where $\mathrm{End}(Q)$ is the monoid of all set maps $Q \rightarrow Q$ with the operation of composition. Since $M$ is defined as a submonoid of $\mathrm{End}(Q)$ it has a faithful monoid action

$$M \times Q \rightarrow Q.$$

Each processor in an abelian network $N = \langle P_v \rangle_{v \in V}$ has a transition monoid

$$M_v := \langle t_a \rangle_{a \in A_v} \subseteq \mathrm{End}(Q_v).$$

(4)

We call this $M_v$ the local monoid at $v$. The product $\prod_{v \in V} M_v$ acts coordinatewise on $Q = \prod_{v \in V} Q_v$. To relate this action to the global and local actions defined above, let $t_v : N^{A_v} \rightarrow M_v$ be the monoid homomorphism sending basis elements $1_a$ to the commuting generators $t_a$, and (recalling $A = \sqcup A_v$) write

$$t : N^A \rightarrow \prod_{v \in V} M_v$$

for the Cartesian product of the maps $t_v$. Each $t_v$ is surjective, so $t$ is surjective. Equations (2) (there is one equation for each $v \in V$) can be written more succinctly as the single equation

$$x \triangleright \triangleright q = t([x \cdot q])q.$$  (5)

To relate $t$ to the local action, note that if $x \triangleright q = y \cdot r$ then $r = t(x)q$.

2.9. Global monoid. If $N$ halts on all inputs, then we can view the entire network as a single abelian processor (see [BL13, Lemma 4.7]) with input alphabet $A = \sqcup A_v$ and state space $Q = \prod_{v \in V} Q_v$. In this case $N$ has a global monoid, defined by

$$M := \langle \tau_a \rangle_{a \in A} \subseteq \mathrm{End}(Q).$$

(6)

where $\tau_a(q) := 1_a \triangleright \triangleright q$. Note that $M$ is not the same as the product of local monoids $\prod M_v$ of (4): the local monoids depend only on the state transition maps $t_a$, but $M$ depends also on the message passing functions (because $\triangleright \triangleright$ does).

Write

$$\tau : N^A \rightarrow M$$

(7)

for the monoid homomorphism sending basis elements $1_a$ to generators $\tau_a$. By Lemma 2.2 we have $x \triangleright \triangleright q = \tau(x)q$.

If $N$ is a finite abelian network (that is, $V$ is finite, and the alphabet $A_v$ and the state space $Q_v$ of each processor are finite) then $M$ is a finite commutative monoid. In this case we denote by $e$ the minimal idempotent of $M$ (§1.2). We will use the following property of $e$ repeatedly.

Lemma 2.5. For any $x \in N^A$ there exists $z \geq x$ such that $\tau(z) = e$. 
2.11. Total kernel and production matrix. In [BL14] we used the local action to associate two basic algebraic objects to an irreducible abelian network, the total kernel $K$ and production map $P : K \to \mathbb{Z}^A$. Together $K$ and $P$ constitute a kind of course-grained description of an abelian network. In this paper we will see that many properties of interest depend only on $K$ and $P$.

Let $e_v$ be the minimal idempotent of the local monoid $M_v$. A state $q \in Q$ is called locally recurrent if $q_v \in e_v Q_v$ for all $v \in V$. By Lemma 1.3, each $m \in M_v$ acts invertibly on $e_v Q_v$, so we have a group action of $\mathbb{Z}^A_v$ on $e_v Q_v$. The total kernel is defined as

$$K = \prod_{v \in V} K_v \subset \mathbb{Z}^A$$

where $K_v$ is the kernel of the action $\mathbb{Z}^A_v \times e_v Q_v \to e_v Q_v$. If $N$ is a finite abelian network, then its total kernel $K$ is a finite index subgroup of $\mathbb{Z}^A$. In particular, it is generated as a group by $K \cap \mathbb{N}^A$. The nonnegative points in $K$ can be characterized as follows.

Lemma 2.7. [BL14, Lemma 4.8] If $N$ is finite and irreducible, then the following are equivalent for $x \in \mathbb{N}^A$. 

2.10. Locally irreducible implies globally irreducible. An abelian processor with transition monoid $M$ and state space $Q$ is called irreducible if the defining action $M \times Q \to Q$ is irreducible (§1.2). Next we prove a local-to-global principle for irreducibility.

Lemma 2.6. Let $N = (P_v)_{v \in V}$ be an abelian network that halts on all inputs. If each processor $P_v$ is irreducible, then $N$ is irreducible.

Proof. Let $q, q' \in Q$. For each $v \in V$, since $P_v$ is irreducible, there exist $m_v, m'_v \in M_v$ such that $m_v q_v = m'_v q'_v$. Since $t$ is surjective we can choose $x, x' \in \mathbb{N}^A$ with $t(x) = m$ and $t(x') = m'$. Write

$$x \triangleright q = y_r, \quad x' \triangleright q' = y'_r$$

where $r = r'$ since $r_v = m_v q_v = m'_v q'_v = r'_v$ for all $v \in V$. Then by Lemma 2.4(iii),

$$(x + y') \triangleright q = y' \triangleright (x \triangleright q) = y' \triangleright (y_r) = (y + y') \triangleright r = y \triangleright (y'.r) = (x' + y) \triangleright q'$$

Hence $\tau(x + y')q = \tau(x' + y)q'$, so the global action $M \times Q \to Q$ is irreducible. □

In light of Lemma 2.6 we will drop “locally” from “locally irreducible” when referring to an abelian network that halts on all inputs.
(1) \( t(x)q = q \) for some locally recurrent \( q \).
(2) \( t(x)q = q \) for all locally recurrent \( q \).
(3) \( x \in K \).

The total kernel depends only on the state transition maps. Next we define the production map, which depends also on the message passing functions. Given a locally recurrent state \( q \) and a vector \( k \in K \cap \mathbb{N}^A \) we have

\[
k \triangleright q = P_q(k).q
\]

for some vector \( P_q(k) \in \mathbb{N}^A \). Nonnegative elements of the total kernel can be thought of as reset vectors: if \( k \in K \cap \mathbb{N}^A \) then processing \( k_a \) letters \( a \) for all \( a \in A \) returns all processors to their initial (locally recurrent) states, and the vector \( P_q(k) \) specifies how many letters of each type are passed as a result.

Lemma 2.8. [BL14, Lemmas 4.6 and 4.9] Let \( N \) be a finite abelian network.

(1) \( P_q : K \cap \mathbb{N}^A \to \mathbb{N}^A \) extends to a group homomorphism \( K \to \mathbb{Z}^A \).
(2) If \( N \) is irreducible then \( P_q = P_r \) for all \( q, r \in Q \).

In light of (2), when \( N \) is irreducible we will often drop the subscript and denote the production map simply by \( P \). In a slight abuse of notation, we also denote by \( P \) the \( A \times A \) matrix of the linear map \( \mathbb{Q}^A \to \mathbb{Q}^A \) obtained by tensoring the production map with \( \mathbb{Q} \). The Laplacian of \( N \) is defined as the \( A \times A \) matrix

\[
L = (I - P)D
\]

where \( I \) is the \( A \times A \) identity matrix and \( D \) is the diagonal matrix with diagonal entries

\[
r_a = \min\{m \geq 1 : m1_a \in K\}.
\]

Theorem 2.9. [BL14, Theorem 5.6 and Corollary 6.4] Let \( N \) be a finite irreducible abelian network \( N \) with production matrix \( P \) and Laplacian \( L \). The following are equivalent.

(1) \( N \) halts on all inputs.
(2) The spectral radius of \( P \) is strictly less than 1.
(3) All principal minors \( L \) are positive.

3. Critical Group

Throughout this section, we take \( N \) to be a finite irreducible abelian network that halts on all inputs.

3.1. Action on recurrent states. Since the global monoid \( M \) of (6) is finite and commutative it has a minimal idempotent \( e \), and \( eM \) is a finite abelian group with identity element \( e \) (see §1.2).

Definition 3.1. The critical group \( \text{Crit} \, N \) is the group \( eM \).

Definition 3.2. A state \( x \in Q \) is recurrent if it satisfies the equivalent conditions of Lemma 1.2 for the defining action \( M \times Q \to Q \).

Denote by \( \text{Rec} \, N \) the set of recurrent states of \( N \).
For example, if \( N = \text{Rotor}(G, s) \) is a simple rotor network on a directed graph \( G \) with sink vertex \( s \), then \( \text{Rec}N \) can be identified with spanning trees of \( G \) oriented toward \( s \) [H+08, Lemma 3.16]. The critical group of \( N \) is isomorphic to the sandpile group \( \text{Crit} \, \text{Sand}(G, s) \) [PDDK96, LL09]. We will deduce this isomorphism as a special case of Theorem 3.11, below.

The following theorem generalizes [H+08, Lemmas 3.13 and 3.17], where it was shown that \( \text{Crit} \, \text{Rotor}(G, s) \) acts freely and transitively on the set of spanning trees of \( G \) oriented toward \( s \).

**Theorem 3.3.** Let \( N \) be a finite irreducible abelian network that halts on all inputs. The action of the critical group on recurrent states

\[ \text{Crit} \times \text{Rec}N \rightarrow \text{Rec}N \]

is free and transitive. In particular, \( \#\text{Crit}N = \#\text{Rec}N \).

**Proof.** Let \( M \) be the transition monoid of \( N \), and let \( e \) be the minimal idempotent of \( M \). Then \( \text{Crit}N = eM \) and \( \text{Rec}N = eQ \). The monoid action \( M \times Q \rightarrow Q \) is faithful by definition and irreducible by Lemma 2.6. Hence the group action \( eM \times eQ \rightarrow eQ \) is free and transitive by Theorem 1.4. \( \square \)

### 3.2. Markov chain

Next we formalize one way in which \( \text{Crit}N \) and its action on \( \text{Rec}N \) govern the “long term behavior” of \( N \). Let \( \alpha \) be a probability distribution on the total alphabet \( A \). Consider the Markov chain \((q_n)_{n \geq 0}\) on state space \( Q \) where the initial state \( q_0 \) can be arbitrary, and subsequent states are defined by

\[ q_{n+1} = 1_{a_n} \triangleright q_n, \quad n \geq 0 \]  

(10)

where the inputs \( a_n \in A \) for \( n \geq 0 \) are drawn independently at random with distribution \( \alpha \).

Recall that \( \mathbf{q} \) is called *recurrent* for the Markov chain if

\[ \Pr(q_n = \mathbf{q} \text{ for some } n \geq 1 \mid q_0 = \mathbf{q}) = 1. \]  

(11)

Next we relate this notion of recurrence to the algebraic notion. Let \( M_\alpha \) be the submonoid of \( M \) generated by \( \{\tau_a : \alpha(a) > 0\} \). We say that \( \mathbf{q} \) is *accessible* from \( \mathbf{q}' \) if \( \mathbf{q} \in M_\alpha \mathbf{q}' \). By the elementary theory of Markov chains, (11) holds if and only if

\[ \mathbf{q} \text{ is accessible from } m\mathbf{q} \text{ for all } m \in M_\alpha \]  

(12)

(here we are using the assumption that \( Q \) is finite). Mutual accessibility is an equivalence relation on \( Q \). Its equivalence classes are called *communicating classes*.

Write \( e_\alpha \) for the minimal idempotent of \( M_\alpha \).

**Lemma 3.4.** \( \mathbf{q} \) is recurrent for the Markov chain (10) if and only if \( e_\alpha \mathbf{q} = \mathbf{q} \).

**Proof.** Given \( m \in M_\alpha \), let \( g \) be the inverse of \( e_\alpha m \) in the group \( e_\alpha M_\alpha \). Since \( g = ge_\alpha \) we have \( gm = g(e_\alpha m) = e_\alpha \), so for any \( \mathbf{q} \in Q \)

\[ g(m\mathbf{q}) = e_\alpha \mathbf{q}. \]

Therefore if \( e_\alpha \mathbf{q} = \mathbf{q} \) then (12) holds. Conversely, if (12) holds then in particular \( \mathbf{q} \) is accessible from \( e_\alpha \mathbf{q} \), so \( \mathbf{q} \in e_\alpha Q \) and hence \( e_\alpha \mathbf{q} = \mathbf{q} \). \( \square \)
If the support of $\alpha$ is too small, it may happen that $e_\alpha \neq e$, or it may happen that $e_\alpha = e$ but $eM_\alpha$ is a proper subgroup of $eM$. We say that $\alpha$ has adequate support if

$$e_\alpha M_\alpha = eM.$$ 

Note that this holds trivially if $\alpha(a) > 0$ for all $a \in A$ (in which case $M_\alpha = M$).

**Lemma 3.5.** If $\alpha$ has adequate support, then $\text{Rec } N$ is the unique communicating class of recurrent states for the Markov chain (10).

**Proof.** If $e_\alpha M_\alpha \subseteq eM$ then $e_\alpha = e$. By Lemma 3.4 the set of recurrent states for the Markov chain is $e_\alpha Q = eQ = \text{Rec } N$.

By Theorem 3.3, the group $eM$ acts transitively on $\text{Rec } N$, so if $eM \subseteq e_\alpha M_\alpha$ then any two states of $\text{Rec } N$ are mutually accessible. $\square$

The next two theorems generalize results of [Dha90], where they are proved for sandpile networks.

**Theorem 3.6.** For any $\alpha$, the uniform distribution on $\text{Rec } N$ is stationary for the Markov chain (10). If $\alpha$ has adequate support, then the stationary distribution is unique.

**Proof.** Fix $q_0 \in \text{Rec } N$, and let $g$ be uniform random element of $\text{Crit } N$. By Theorem 3.3, $g \triangleright q_0$ is a uniform random element of $\text{Rec } N$. If $a \in A$ is independent of $g$, then $\tau_a g$ is also a uniform random element of $\text{Crit } N$; hence if $q_\alpha$ is uniform on $\text{Rec } N$, then $q_{\alpha+1}$ is again uniform on $\text{Rec } N$.

If $\alpha$ has adequate support, then $\text{Rec } N$ is the unique recurrent communicating class by Lemma 3.5, so the stationary distribution is unique. $\square$

### 3.3. Expected time to halt.

Recall our assumptions that $N$ is finite and irreducible and halts on all inputs. In particular, the production matrix $P$ is well defined and has spectral radius $< 1$ by Theorem 2.9, so $I - P$ is invertible where $I$ is the $A \times A$ identity matrix. Our next result gives an interpretation for the entry $(I - P)^{-1}_{ab}$: it is the expected number of letters $a$ processed before the network halts, when one letter $b$ is input to a uniform recurrent state.

**Theorem 3.7.** (Expected Time To Halt) Let $q$ be a uniform random element of $\text{Rec } N$. Then for all $x \in \mathbb{N}^A$ we have

$$\mathbb{E}[x, q] = (I - P)^{-1}x.$$ 

Later, in Theorem 4.2, we will bound the difference $[x, q] - \mathbb{E}[x, q]$. To build up to the proofs of these results, consider the group homomorphism

$$\phi : \mathbb{Z}^A \rightarrow \text{Crit } N$$

defined on generators by $1_a \mapsto e\tau_a$ for each $a \in A$. For $x \in \mathbb{N}^A$ and $q \in Q$ we have

$$\phi(x)q = \left(\prod_{a \in A} (e\tau_a)^{x_a}\right)q = \left(\prod_{a \in A} \tau_a^{x_a}\right)eq = x \triangleright eq.$$ 

(13)

First we observe that input vectors in the kernel of $\phi$ act trivially on recurrent states.
Lemma 3.8. If $x \in (\ker \phi) \cap \mathbb{N}^A$ and $q \in \text{Rec} N$, then $x \triangleright \triangleright q = q$.

Proof. For $q \in \text{Rec} N$ we have $q = e \cdot q$. Now by (13), since $\phi(x) = e$,
$$x \triangleright \triangleright q = x \triangleright \triangleright e \cdot q = \phi(x)q = e \cdot q = q.$$ 

Write $e_v$ for the minimal idempotent of the local monoid $M_v$ (§2.8).

Lemma 3.9. (Recurrence Implies Locally Recurrence) If $e \cdot q = q$ then $e_v \cdot q_v = q_v$ for all $v \in V$.

Proof. By Lemma 2.5 there exists $z \geq 1$ be such that $\tau(z) = e$. Let $u = [z \cdot q] \geq 1$. For each $v \in V$ let $y_v \in \mathbb{N}^A_v$ be such that $t_v(y_v) = e_v$. Then for all sufficiently large $n$ we have $nu_v \geq y_v$ and hence $t_v(nu_v) \in e_v \cdot M_v$. Since $e_v \cdot M_v$ is a finite group, there exists $n_v \in \mathbb{N}$ such that $t_v(nu_v) = e_v$.

Now if $e \cdot q = q$ then for all $n \in \mathbb{N}$ we have $(nz) \triangleright \triangleright q = q$ and $[nz \cdot q] = nu$ by Lemma 2.2, and hence $t(nu)q = q$ by (5). Taking $n = n_v$ we obtain
$$e_v \cdot q_v = t_v(nu_v)q_v = q_v.$$ 

The converse of Lemma 3.9 is usually false: For example, all states of $\text{Rotor}(G, s)$ and $\text{Sand}(G, s)$ are locally recurrent, but states of the former containing an oriented cycle of rotors are not recurrent, nor are states of the latter containing “forbidden” subconfigurations (the simplest of which is a pair of adjacent vertices both in state 0). Theorem 3.19 gives conditions when the converse does hold.

Now we come to the main ingredient in the proof of Theorem 3.7: if the network starts in a locally recurrent state $q$ and halts in the same state $q$, then the odometer $[x \cdot q]$ belongs to the total kernel $K$.

Lemma 3.10. Fix $x \in \mathbb{N}^A$ and a locally recurrent state $q \in Q$. Let $k = [x \cdot q]$. If $x \triangleright \triangleright q = q$, then $k \in K$ and $x = (I - P)k$.

Proof. If $x \triangleright \triangleright q = q$, then $t(k)q = q$ by equation (5). By Lemma 2.7 it follows that $k \in K$. Now by Lemma 2.4(ii) and the definition (8) of the production matrix,
$$k \cdot q = k \triangleright \triangleright (x \cdot q) = (P(k) + x) \cdot q$$

Hence $k = P(k) + x$. 

Proof of Theorem 3.7. Since $\text{Crit} N$ is a finite group, for any $x \in \mathbb{Z}^A$ there is a positive integer $n$ such that $nx \in \ker \phi$. Fix $q \in \text{Rec} N$ and let $k = [nx \cdot q]$. We have $nx \triangleright \triangleright q = q$ by Lemma 3.8. Moreover $q$ is locally recurrent by Lemma 3.9, so by Lemma 3.10 it follows that $nx = (I - P)k$. In particular, since $I - P$ is invertible, $k$ does not depend on $q$.

Now by Lemma 2.2,
$$k = \sum_{j=0}^{n-1} [x \cdot (jx \triangleright \triangleright q)].$$

By Theorem 3.6, if $q$ is uniform on $\text{Rec} N$ then $jx \triangleright \triangleright q$ is uniform on $\text{Rec} N$ for all $j \in \mathbb{N}$. Taking expectations, we obtain
$$k = E(k) = nE(x \cdot q).$$
Theorem 3.11. The natural map \( \phi : \mathbb{Z}^A \to \text{Crit} N \) induces an isomorphism of abelian groups

\[
\text{Crit} N \cong \mathbb{Z}^A/(I - P)K.
\]

Proof. We must show that \( \ker \phi = (I - P)K \). Fix \( q \in \text{Rec} N \). For any \( k \in K \cap \mathbb{N}^A \) we have

\[
k \triangleright q = P(k)q
\]

so \( k \triangleright q = P(k) \triangleright q \) by Lemma 2.3. Hence \( \phi(k)q = \phi(P(k))q \) by (13). By Theorem 3.3 the action of Crit \( N \) on \( \text{Rec} N \) is free, so we conclude \( \phi(k) = \phi(P(k)) \). This shows that \( (I - P)k \in \ker \phi \) for all \( k \in K \cap \mathbb{N}^A \). Since \( K \) is generated as a group by \( K \cap \mathbb{N}^A \), it follows that \( (I - P)K \subseteq \ker \phi \).

To show the reverse inclusion, given \( x \in (\ker \phi) \cap \mathbb{N}^A \) we have \( \phi(x) = e \) and hence \( x \triangleright e = e q \) for all \( q \in Q \) by (13). By Lemma 3.10, \( x = (I - P)k \) where \( k = [x.e q] \in K \). It follows that \( (\ker \phi) \cap \mathbb{N}^A \subseteq (I - P)K \). Since Crit \( N \) is finite, \( \ker \phi \) is a full rank subgroup of \( \mathbb{Z}^A \), so it is generated as a group by \( (\ker \phi) \cap \mathbb{N}^A \), which shows that \( \ker \phi \subseteq (I - P)K \).

Irreducible abelian networks \( N \) and \( N' \) on the same graph with the same total alphabet are called homotopic, written \( N \cong N' \), if they have the same total kernel \( K \) and the same production map \( P \). An example of homotopic networks are the sandpile and simple rotor networks on a connected graph \( G = (V, E) \). These have total kernel \( K = \prod_{v \in V} (d_v \mathbb{Z}) \) where \( d_v \) is the degree of vertex \( v \): in the sandpile case processing \( d_v \) letters \( v \) causes \( v \) to topple exactly once, while in the rotor case it causes the rotor at \( v \) to serve each neighbor \( u \) of \( v \) exactly once. In both cases exactly one letter is sent to each neighbor, so the production map is given by \( P(k)_u = \sum_{(v,u) \in E} k_v/d_v \).

By Theorem 3.11 the critical group depends only on the total kernel \( K \) and production matrix \( P \), so homotopic networks have isomorphic critical groups.

Corollary 3.12. If \( N \cong N' \), then \( \text{Crit} N \cong \text{Crit} N' \).

Corollary 3.12 generalizes the isomorphism \( \text{Crit Rotator}(G, s) \cong \text{Crit Sand}(G, s) \) between the rotor and sandpile groups of a graph with sink vertex \( s \).

Next we relate the critical group to the cokernel of the Laplacian (9).

Definition 3.13. An abelian network \( N \) is rectangular if its total kernel is \( K = \prod_{a \in A} (r_a \mathbb{Z}) \) (that is, \( K \) is a rectangular sublattice of \( \mathbb{Z}^A \)).
Corollary 3.14. The natural map $\mathbb{Z}^A \to \text{Crit } N$ induces a surjective group homomorphism

$$\phi : \mathbb{Z}^A/L\mathbb{Z}^A \twoheadrightarrow \text{Crit } N.$$ 

If $N$ is rectangular, then $\phi$ is an isomorphism.

Proof. By definition, $D\mathbb{Z}^A \subset K$ with equality if $N$ is rectangular. Since $L = (I - P)D$, we have $L\mathbb{Z}^A \subset (I - P)K$ with equality if $N$ is rectangular. \hfill $\Box$

Note that any unary network (and in particular any toppling network) is rectangular. In [BL14] we defined the sandpilization $S(N)$ as the locally recurrent toppling network with the same Laplacian as $N$.

Corollary 3.15. $\text{Crit } S(N) \simeq \mathbb{Z}^A/L\mathbb{Z}^A$ and $\text{Crit } S(N) \twoheadrightarrow \text{Crit } N$.

Example. To see that the map $\text{Crit } S(N) \to \text{Crit } N$ need not be an isomorphism, consider the following non-rectangular network with vertices $i, j$ where $j$ is a sink. Let $Q_i = \{0, 1\}$, $A_i = \{a, b\}$, $A_j = \{c\}$,

$$T_i(q, a) = T_i(q, b) = q + 1 \pmod{2}$$

and

$$T_{(i,j)}(0, a) = c,$$
$$T_{(i,j)}(1, a) = cc,$$
$$T_{(i,j)}(0, b) = \epsilon,$$
$$T_{(i,j)}(1, b) = c.$$

Figure 1 shows the state diagram of $P_i$. 

\begin{figure}[h]
\centering
\begin{tikzpicture}
\node at (0,0) {$0$};
\node at (0,1) {$1$};
\node at (1,0) {$0$};
\node at (1,1) {$1$};
\node at (2,0) {$0$};
\node at (2,1) {$1$};
\node at (3,0) {$0$};
\node at (3,1) {$1$};
\node at (0,2) {$0$};
\node at (0,3) {$1$};
\node at (1,2) {$0$};
\node at (1,3) {$1$};
\node at (2,2) {$0$};
\node at (2,3) {$1$};
\node at (3,2) {$0$};
\node at (3,3) {$1$};
\draw[->] (0,0) -- (0,3);
\draw[->] (0,1) -- (0,3);
\draw[->] (1,0) -- (1,3);
\draw[->] (1,1) -- (1,3);
\draw[->] (2,0) -- (2,3);
\draw[->] (2,1) -- (2,3);
\draw[->] (3,0) -- (3,3);
\draw[->] (3,1) -- (3,3);
\end{tikzpicture}
\caption{State diagram of an abelian processor with two states 0, 1 and two inputs $a, b$. Its kernel is $\{(m, n) \in \mathbb{Z}^2 : m + n \equiv 0 \pmod{2}\}$, so it is not rectangular. Each line crossed results in output of one letter $c$.}
\end{figure}
The production matrix and Laplacian of this network, with rows and columns indexed by \(a, b, c\) in that order, are:

\[
P = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ \frac{3}{2} & \frac{1}{2} & 0 \end{pmatrix}, \quad L = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ -3 & -1 & 1 \end{pmatrix}
\]

We have \(\text{Crit}_S(N) = \mathbb{Z}^3/L\mathbb{Z}^3 = (\mathbb{Z}/2\mathbb{Z})^2\). On the other hand, \(\tau_a = \tau_b\) in Crit\(N\) (both send \(0 \mapsto 1 \mapsto 0\)) so Crit\(N = \mathbb{Z}/2\mathbb{Z}\).

Processor \(P_i\) in this example has another curious feature: The input words \(aa, ab, bb\) to state 0 all result in the same sequence of states \(0 \mapsto 1 \mapsto 0\) yet they produce different outputs (ccc, cc, c respectively). □

3.5. **Order of the critical group.** Now we turn to the problem of counting recurrent states, or equivalently (by Theorem 3.3) finding the order of the critical group. Let

\[
\iota = [K : D\mathbb{Z}^A]
\]

be the index of \(D\mathbb{Z}^A\) as a subgroup of \(K\). Recalling that \(K = \prod_{v \in V} K_v\), we can write \(\iota = \prod_{v \in V} \iota_v\) as a product of local indices

\[
\iota_v = [K_v : D_v\mathbb{Z}^{A_v}]
\]

where \(D_v\mathbb{Z}^{A_v} := \prod_{a \in A_v} (r_a \mathbb{Z})\). Note \(\iota = 1\) if and only if \(N\) is rectangular.

**Theorem 3.16.**

\[
\#\text{Rec} N = \#\text{Crit} N = \frac{\det L}{\iota}.
\]

**Proof.** The first equality follows from Theorem 3.3. For the second, we have by Theorem 3.11

\[
\#\text{Crit} N = \frac{[\mathbb{Z}^A : (I - P)K]}{[\det L]} = \frac{[\mathbb{Z}^A : L\mathbb{Z}^A]}{[\det L]} = \frac{[(I - P)K : (I - P)D\mathbb{Z}^A]}{[K : D\mathbb{Z}^A]}.
\]

By Theorem 2.9, since \(N\) halts on all inputs, \(I - P\) has full rank and \(\det L > 0\), so the right side equals \(\frac{\det L}{[K : D\mathbb{Z}^A]}\). □

**Definition 3.17.** The production graph of \(N\) is the directed graph with vertex set \(A\) and edge set \(\{(a, b) : P_{ba} > 0\}\).

**Lemma 3.18.** Let \(S\) be a toppling network with production graph \(\Gamma\). For each directed cycle \(a_1 \rightarrow \ldots \rightarrow a_m \rightarrow a_1\) of \(\Gamma\) we have

\[
\max\{q(a_1), \ldots, q(a_m)\} \geq 1
\]

for all \(q \in \text{Rec} S\).

**Proof.** Let \(r = (r_a)_{a \in A}\) be the vector of toppling thresholds. By Lemma 2.5 there exists \(z \geq r\) such that \(\tau(z) = e\). If \(q \in \text{Rec} S\) then \(z \gg q = q\) and each vertex topples at least once during the stabilization of \(zq\). If \(a_i\) is the last vertex on
the cycle to finish toppling, then \( a_{i+1} \) receives a chip from \( a_i \) and does not topple thereafter, so \( q(a_{i+1}) \geq 1 \).

**Theorem 3.19.** Let \( N \) be a finite irreducible abelian network that halts on all inputs. The following are equivalent.

(1) Every locally recurrent state of \( N \) is recurrent.

(2) \( \det L = \det D \).

(3) Every state of \( S(N) \) is recurrent.

(4) The state \( 0 \) of \( S(N) \) is recurrent.

(5) \( \Gamma \) has no directed cycles.

(6) \( P \) is nilpotent.

**Proof.** Recall from §2.11 the action of \( \mathbb{Z}^{A_v} \) on \( e_vQ_v \) for each vertex \( v \). Since \( N \) is irreducible this action is transitive, and its kernel is \( K_v \). So the number of locally recurrent states of \( N \) is

\[
\prod_{v \in V} \#(e_vQ_v) = \prod_{v \in V} \#(\mathbb{Z}^{A_v}/K_v) = [\mathbb{Z}^A : K].
\]

By Theorem 3.16, the number of recurrent states of \( N \) is

\[
\frac{\det L}{[K : D\mathbb{Z}^A]} = \frac{\det L}{\det D}[\mathbb{Z}^A : K].
\]

Now from Lemma 3.9 it follows that (1) \( \Leftrightarrow \) (2).

All states of the sandpilization \( S(N) \) are locally recurrent, and \( S(N) \) has the same matrices \( L \) and \( D \) as \( N \), so (2) \( \Rightarrow \) (3).

Trivially (3) \( \Rightarrow \) (4).

If \( \Gamma \) has a directed cycle, then the state \( 0 \) of \( S(N) \) is not recurrent by Lemma 3.18, which shows (4) \( \Rightarrow \) (5).

If \( \Gamma \) has no directed cycles, then each entry of \( P^k \) is a weighted sum over directed paths in \( \Gamma \) with \( k+1 \) distinct vertices. Taking \( k = \#A \) we obtain \( P^k = 0 \), which shows (5) \( \Rightarrow \) (6).

If \( \lambda_1, \ldots, \lambda_n \) are the eigenvalues of \( P \) with multiplicity, then

\[
\frac{\det L}{\det D} = \det(I - P) = \prod_{i=1}^n (1 - \lambda_i).
\]

If \( P \) is nilpotent then \( \lambda_1 = \cdots = \lambda_n = 0 \), so \( \det L = \det D \). Hence (6) \( \Rightarrow \) (2), completing the proof.

\( \square \)

4. **Time to halt**

In this section we show that the production matrix of \( N \) determines its running time on any input up to an additive constant (Theorem 4.2). In the special case of rotor networks, additive error bounds of this type appear in the work of Cooper and Spencer [CS06] and Holroyd and Propp [HP10]. For an upper bound in the case of sandpiles, see [H+08, Prop. 4.8]. A related but distinct question is: given \( \mathbf{x} \) and \( \mathbf{q} \), how quickly can one compute the final state \( \mathbf{x} \rightarrow \mathbf{q} \) and the odometer \([\mathbf{x}, \mathbf{q}]\)? According to Theorem 4.2 the abelian network \( N \) performs this
computation (asynchronously) in time approximately $1^T (I - P)^{-1} x$, but in some cases [BS13, FL13] one can design a (sequential) algorithm that is much faster.

As usual we take $N$ to be a finite irreducible abelian network that halts on all inputs. Let $K$ be its total kernel and $P : K \to \mathbb{Z}^A$ its production map. Write $I$ for the inclusion $K \hookrightarrow \mathbb{Z}^A$.

Lemma 4.1. Suppose $k \in K$ satisfies $P k \leq k$. If $q \in \text{Rec } N$, then

$$(I - P) k \gg q = q.$$ 

Proof. Let $x = (I - P) k \in \mathbb{N}^A$. By Theorem 3.11, if $k \in K$ then $x \in \ker \phi$. The conclusion now follows from Lemma 3.8. □

Remark. Recalling that $I - P$ is injective (Theorem 2.9) and that the matrix of $(I - P)^{-1}$ has nonnegative entries (Theorem 3.7), we see that the condition $P k \leq k$ in the above lemma implies $k \geq 0$.

For $u \in \mathbb{Z}^A$ write $\|u\|_\infty = \max_{a \in A} |u_a|$.

Theorem 4.2. (Time To Halt) Let $N$ be an irreducible finite abelian network that halts on all inputs. There is a constant $C$ depending only on $N$, such that for all $x \in \mathbb{N}^A$ and all $q \in Q$,

$$\| [x.q] - (I - P)^{-1} x \|_\infty \leq C.$$ 

Proof. Since $(I - P) K$ has full rank in $\mathbb{Z}^A$, there is a constant $c$ such that for any $x \in \mathbb{N}^A$ there exists $y = (I - P) k \in (I - P) K$ such that $0 \leq y - x \leq c1$. Since $(I - P)^{-1}$ has nonnegative entries,

$$0 \leq k - (I - P)^{-1} x \leq (I - P)^{-1} c1.$$ 

Therefore it suffices to bound $\| [x.q] - k \|_\infty$.

To do this, fix $z \in \mathbb{N}^A$ with $\tau(z) = e$. We will show

$$\| [x.eq] - k \|_\infty \leq C_1$$  

and

$$\| [x.eq] - [x.q] \|_\infty \leq C_2$$  

where $C_1 = \max_{q' \in Q} \| [c1.q'] \|_\infty$ and $C_2 = \max_{q' \in Q} \| [z.q'] \|_\infty$.

Since $y \geq 0$, we have $P k \leq k$, so $y \gg e q = eq$ by Lemma 4.1. Hence by Lemmas 3.10 and 2.2

$$k = [y.eq] = [x.eq] + [(y - x). (x \gg e q)]$$

which shows (14). Two more applications of Lemma 2.2 give

$$[z.q] + [x.eq] = [(x + z).q] = [x.q] + [z. (x \gg q)]$$

which shows (15). □

By Theorem 4.2 and the triangle inequality,

$$\| [x.q] - [x.r] \|_\infty \leq 2C$$

for all $x \in \mathbb{N}^A$ and all $q, r \in Q$. We will use this bound in the next section.
5. Burning Test

Dhar’s burning test [Dha90] is an efficient algorithm for determining whether a state of \( \text{Sand}(G, s) \) is recurrent. In its simplest form it applies to Eulerian directed graphs \( G \) (strongly connected, indegree\( (v) = \) outdegree\( (v) \) for \( v \in V \)); see [H+08, §4]. Speer [Spe93] treated general directed graphs. A variant of Speer’s algorithm can be found in [PPW11]. The goal of this section is to generalize these algorithms to the setting of an arbitrary finite irreducible abelian network \( N \) that halts on all inputs.

**Definition 5.1.** A *burning element* for \( N \) is a vector \( \beta \in \mathbb{N}^A \) such that for \( q \in Q \)

\[
q \in \text{Rec} \, N \iff \beta \triangleright \! \! \! \! \! \! \triangleright q = q.
\]

To see that burning elements exist, we can use Lemma 1.2(5): \( q \in \text{Rec} \, N \) if and only if \( q = e q \), where \( e \) is the minimal idempotent of the transition monoid \( M \). Recalling the map \( \tau \) of (7), any \( \beta \in \mathbb{N}^A \) such that \( \tau(\beta) = e \) is a burning element. However, such elements are typically large. The power of the burning test derives from the fact that one can often identify a small burning element \( \beta \) to reduce the running time \( [\beta, q] \).

In the classical setting of \( N = \text{Sand}(G, s) \), where \( G \) is a directed graph with globally accessible sink \( s \), there is a pointwise minimal burning element \( \beta \). In the case that \( G \) is Eulerian, the usual formula for the burning element is \( \beta_v = d_{vs} \), the number of edges into \( v \) from \( s \). The special role of the sink is undesirable for us (since in general, an abelian network need not have a sink in order to halt on all inputs). To remove it, notice that

\[
\beta = L 1
\]

where \( L \) is the Laplacian of (9). (To make the connection to the graph Laplacian \( \Delta_G = D_G - A_G \), where \( D_G \) is the diagonal matrix of outdegrees, and \( A_G \) is the adjacency matrix of \( G \), we have \( L = \Delta_G \) if \( G' \) is the graph obtained from \( G \) by removing all outgoing edges from \( s \). Since \( G \) is Eulerian we have \( \Delta_G 1 = 0 \), so \( L 1 = -\Delta_G 1_s = \beta \).)

When \( G \) is not Eulerian, \( L 1 \) may have negative entries. Speer [Spe93] observed that there is a pointwise smallest vector \( y \geq 1 \) such that \( L y \geq 0 \), and showed that \( \beta = L y \) is a burning element. Our burning test for abelian networks, Theorem 5.5, reduces to Speer’s in the case \( N = \text{Sand}(G, s) \).

5.1. Large inputs. To lay the ground for the burning test, we show in this section that states obtained from sufficiently large inputs to \( N \) must be recurrent. We consider two interpretations of “large”: either the number of letters in the input \( x \) is large (Lemma 5.2), or the odometer \( [x, q] \) is large (Lemmas 5.3 and 5.4).

**Lemma 5.2.** Let \( z \in \mathbb{N}^A \) be such that \( \tau(z) = e \). If \( x \geq z \) then \( x \triangleright \! \! \! \! \! \! \triangleright q \) is recurrent for all \( q \in Q \).

**Proof.** Writing \( x = y + z \) for \( y \in \mathbb{N}^A \), we have

\[
x \triangleright \! \! \! \! \! \! \triangleright q = z \triangleright \! \! \! \! \! \! \triangleright (y \triangleright \! \! \! \! \! \! \triangleright q) = emq.
\]

where \( m = \tau(y) \). By Lemma 1.2(4) the state \( emq \) is recurrent. \( \square \)
Given $q, r \in Q$ we say that $q$ is \textit{locally accessible} from $r$ if $q =_mr$ for some $m \in \prod_{t \in T} M_t$. In particular, if there is an execution (say $w$) from $x.r$ to $y.q$, then $q$ is locally accessible from $r$ (namely $q = t(|w|)r$).

**Lemma 5.3.** If there exists $x \in N^A$ such that $x \gg r = r$ and $[x.q] \geq 1$ for all states $q$ locally accessible from $r$, then $r$ is recurrent.

**Proof.** Let $z \in N^A$ be such that $\tau(z) = e$. We will find a $y \geq z$ and a state $q$ such that $r = y \gg q$.

Let $n = \sum_{a \in A} z_a$. Fix any sequence $a_1, \ldots, a_n$ such that $z = \sum_{i=1}^n 1_{a_i}$. Let $q^0 = r$ and inductively define states $q^1, \ldots, q^n$, all locally accessible from $r$, as follows.

For each $i = 1, \ldots, n$, since $q^{i-1}$ is locally accessible from $r$ we have $[x.q^{i-1}]_{a_i} \geq 1$, so there is a legal execution $w_i$ from $x.q^{i-1}$ to some state $y^i.q^i$ satisfying $y^i_{a_i} \geq 1$. The concatenation $w_{i+1} \cdots w_n$ is a legal execution from $(nx).r$ to $y.q^n$ with $y = y^1 + \cdots + y^n \geq z$. Hence by Lemma 2.1,

$$r = (nx) \gg r = y \gg q^n$$

and the right side is recurrent by Lemma 5.2.

For $u \in N^A$ and $q \in Q$, define

$$R_u(q) = \{ x \gg q \mid x \in N^A, [x.q] \geq u \}.$$  

and write $R_u$ for the set of states $q \in Q$ such that $q \in R_u(q)$.

We have defined recurrent states in Lemma 1.2 by a list of equivalent monoid-theoretic properties. Now we can add to this list a characterization that is specific to abelian networks. According to the next lemma, a state $q$ is recurrent if and only if $q$ there exist inputs with arbitrarily large odometers that fix $q$.

**Lemma 5.4.**

$$\text{Rec } N = \bigcap_{u \in N^A} R_u.$$  

**Proof.** Suppose that $r \in \bigcap R_u$. Then for any $u \in N^A$ there is an input $x \in N^A$ with $r = x \gg r$ and $[x.r] \geq u$. Take $u = (2C + 1)1$. By Theorem 4.2 it follows that $[x.q] \geq 1$ for all $q \in Q$. Now from Lemma 5.3 it follows that $r$ is recurrent.

It remains to show that $\text{Rec } N \subset R_u$ for all $u \in N^A$. Since $\text{Crit } N$ is a finite group, the kernel of $\phi : Z^A \to \text{Crit } N$ has nonempty intersection with $N^A + u$. Let $x$ be a point in this intersection. Then for any $q \in \text{Rec } N$ we have $x \gg q = q$ by Lemma 3.8. Since $[x.q] \geq x \geq u$, it follows that $q \in R_u$.

Finally, we show that if $k \geq 1$ then the converse to Lemma 4.1 holds.

**Theorem 5.5.** (Burning Test) Let $k \in K$ be such that $k \geq 1$ and $P^k \leq k$. Then $q \in Q$ is recurrent if and only if $(I - P)^k \gg q = q$.

**Proof.** If $q$ is recurrent, then $(I - P)^k \gg q = q$ by Lemma 4.1.
For the converse, let \( x = (I - P)k \) and \( u = [x, q] \). Suppose that \( x \triangleright \triangleright q = q \). Then \( x = (I - P)u \) by Lemma 3.10. Since \( I - P \) is injective, we obtain \( u = k \). Now for any \( n \in \mathbb{N} \) we have \( nx \triangleright \triangleright q = q \) and \( [nx, q] = nk \). Hence
\[
q \in \bigcap_{n \geq 1} R_{nk}.
\]
Since \( k \geq 1 \) the right side equals \( \bigcap_{u \in A} R_u \), which equals \( \text{Rec} N \) by Lemma 5.4. □

Remark. The preceding theorem can be improved slightly by weakening \( k \geq 1 \) to \( k \geq 1_C \), where \( C \) is the set of all \( a \in A \) that lie on a directed cycle of the production graph \( \Gamma \) (Definition 3.17). The reason is that if \( a \in A \) does not lie on a directed cycle, then \( \{a\} \) is a strong component of \( \Gamma \), and all locally recurrent states of the strong component \( N^a \) are recurrent. Writing \( a^\omega \) for the minimal idempotent of the transition monoid of \( N^a \), one checks that \( e = \prod_{a \in C} a^\omega \) and hence that \( \text{Rec} N = \bigcap_{a \in C} \text{Rec} N^a \). See [C+13] where this improvement is carried out in detail for sandpile networks.

5.2. Finding a burning element. In this section we show that there is always a burning element \( \beta \) satisfying \( 0 \leq \beta \leq r \), where \( r_a \) is the smallest positive integer such that \( r_a 1_a \in K \).

Definition 5.6. A burning odometer is a vector \( k \) satisfying
\[
(I - P)k \geq 0, \quad k \geq 1, \quad k \in K.
\]  

The corresponding burning element is \( \beta = (I - P)k \).

According to Theorem 5.5, to check whether \( q \) is recurrent it suffices to find a burning odometer \( k \) and then compute \( \beta \triangleright \triangleright q \), where \( \beta = (I - P)k \). The proof also shows that if \( q \) is recurrent, then the local run time of this computation is \( [\beta, q] = k \) (that is, for each \( a \in A \) it requires processing \( k_a \) letters \( a \)). Therefore we are interested in finding a burning odometer \( k \) as small as possible.

Recall that \( K \subset DZ^A \), with equality if \( N \) is rectangular. For a sandpile network \( N = \text{Sand}(G, s) \), a burning odometer is \( k = Dy \) where \( y \) is the “burning script” of [Spe93]. More generally, if \( N \) is rectangular, then writing \( k = Dy \), (16) is equivalent to
\[
Ly \geq 0, \quad y \geq 1, \quad y \in Z^A
\]  

where \( L = (I - P)D \) is the Laplacian of \( N \). Setting \( x = y - 1 \) we find that (16) is equivalent to
\[
Lx \geq -L1, \quad x \geq 0, \quad x \in Z^A.
\]  

Minimizing \( 1^Tx \) subject to these constraints is an integer program of the class solved by toppling networks [BL13, Remark 4.9]. Specifically, consider the sandpileization \( S(N) \), enlarged to allow negative chip counts. In this network, vertex \( a \) has toppling threshold \( r_a \), where \( r \) is the vector of diagonal entries of the diagonal matrix \( D \). By Theorem 2.9, if \( N \) halts on all inputs then \( S(N) \) halts on all inputs (since \( S(N) \) has the same Laplacian as \( N \)). For a chip configuration \( q \in Z^A \), write \( q^\circ \) for the stabilization of \( q \) in \( S(N) \).
Corollary 5.7. \((S(N)\) computes a minimal burning element for \(N\)) Let \(N\) be an irreducible rectangular network that halts on all inputs. Then \(N\) has a pointwise smallest burning odometer \(k = Dy\). The corresponding burning element is given by

\[
\beta = Ly = r - 1 - (r - 1 - L1)^\circ.
\]

Moreover, \(0 \leq \beta \leq r\).

Proof. Let \(q = r - 1 - L1\), and for \(a \in A\) let \(x_a\) be the number of times \(a\) topples during the stabilization of \(q\) in \(S(N)\). By [BL13, Remark 4.9], \(x\) is the pointwise smallest vector satisfying (18), and \(q^\circ = q - Lx\). Setting \(y = x + 1\), we conclude that \(k = Dy\) is the pointwise smallest vector satisfying (16), and

\[
\beta = (I - P)k = L(x + 1) = L1 + q - q^\circ = r - 1 - q^\circ.
\]

Noting that \(L1 \leq r\), we have \(-1 \leq q^\circ \leq r - 1\) and hence \(0 \leq \beta \leq r\). \(\square\)

Remark. If we replace 1 by \(1_C\) in (16), then we obtain a slightly smaller burning element \(\beta\) which satisfies \(0 \leq \beta \leq r - 1\). To compute this smaller element using \(S(N)\), take \(q = r - 1 - L1_C\).

In practice, it is more direct to find the burning element by “untoppings” instead of topplings, which amounts to the following procedure to find the minimal \(y\) satisfying (17).

Procedure 5.8. Start with \(y = 1\). If \(Ly \geq 0\), then stop. Otherwise, choose some \(a \in A\) such that \((Ly)_a < 0\) and increase \(y_a\) by 1. Repeat until \(Ly \geq 0\).

Remark. In the case that the row sums of \(L\) are nonnegative, the procedure halts immediately with \(y \equiv 1\). In particular, this includes the special case \(N = Sand(G, s)\) for an Eulerian graph \(G\).

In the case \(N\) is not rectangular, the inclusion \(K \subset DZA\) is strict. Corollary 5.7 and Procedure 5.8 will identify the minimal burning odometer \(Dy \in DZA\). Unlike the rectangular case, there may not be a unique minimal burning odometer in \(K\).

If \(k = Dy\) is the minimal burning odometer in \(DZA\), then the global burning test \((I - P)k \gg q\) runs in time \(k\). Since \(\beta = (I - P)k \leq r\), an upper bound for this run time is \(k \leq (I - P)^{-1}r\).

6. Concluding Remarks

We conclude with a few directions for future research.

Combinatorics of recurrent states. The recurrent states of the rotor network \(Rotor(G, s)\) are the oriented spanning trees of \(G\) rooted at \(s\). The recurrent states of the sandpile network \(Sand(G, s)\) on an undirected (or Eulerian directed) graph \(G\) have a characterization in terms of “forbidden subconfigurations” [Dha90] which puts them naturally in bijection with the \(G\)-parking functions of Postnikov and Shapiro [PS04]. Recently Guzmán and Klivans [GK14] have generalized this correspondence to toppling networks. Hopkins and Perkinson [HP14] relate the \(G\)-parking functions to chambers of a bigraphical hyperplane arrangement. It would
be interesting to find combinatorial characterizations of the recurrent states of other abelian networks.

The rank function in Baker and Norine’s Riemann-Roch theorem for graphs [BN07] has the following interpretation: given a state of the sandpile network $\text{Sand}(G)$ with no sink, what is the smallest number of letters that can be input to cause it to run forever? Are there analogues of the Baker-Norine theorem and the Lorenzini zeta function [Lor12] for more general abelian networks?

Duval, Klivans and Martin [DKM09, DKM13] define a higher dimensional critical group as the cokernel (over $\mathbb{Z}$) of the Laplacian of a simplicial complex. Does this group have a dynamical interpretation in terms of a network in which a set of nodes can interact if they form a face of the simplicial complex?

**Critical networks.** Let us call an abelian network $N$ **critical** if its production matrix has spectral radius 1. By Theorem 2.9, a critical network has inputs that cause it to run forever, so it does not have a critical group in the sense of §3. However, one could try to define $\text{Crit}N$ by generalizing the sinkless construction of the sandpile group: $(\mathbb{Z}^V)_0/L\mathbb{Z}^V$, where $(\mathbb{Z}^V)_0$ is the kernel of the map $x \mapsto 1^T x$.

For $\alpha, \beta \in \mathbb{N}^A \times Q$, write $\alpha \rightarrow \beta$ if there exists a legal execution from $\alpha$ to $\beta$. Let us call $\alpha$ **recurrent** if for any $\beta$ such that $\alpha \rightarrow \beta$ we have $\beta \rightarrow \alpha$. In a critical network $N$, is there an efficient test analogous to the burning algorithm to check whether $\alpha$ is recurrent?

In the case of a simple rotor network $\text{Rotor}(G)$ with no sink, a state $\alpha = 1_e \cdot q$ with just one letter is recurrent in this sense if and only if its rotors $(u, q_u)_{u \in V}$ form a cycle-rooted spanning tree (a spanning subgraph with a single oriented cycle) with $v$ lying on the cycle [H+08, Theorem 3.8]. Does this result extend to networks of abelian mobile agents (a non-unary generalization of rotors proposed in [BL13])? What about states $\alpha$ with more than one letter?

**Finer algebraic invariants.** The critical group $\text{Crit}N$ depends only on the homotopy type of $N$ (Corollary 3.12). On the other hand, the global monoid $M$ of (6) can detect finer information about $N$. To see that the monoid action $M \times Q \rightarrow Q$ is not a homotopy invariant, note that the sandpile network $\text{Sand}(G, s)$ has a state $0 \in Q$ which can access all other states: $M0 = Q$. If $G - \{s\}$ has two directed cycles that share an edge, then $\text{Rotor}(G, s)$ has no such state, because a progressed cycle of rotors once broken can never be reformed.

Neither is $M$ itself a homotopy invariant: for instance, for the discrete torus $G = \mathbb{Z}/n \times \mathbb{Z}/n$ one can show that the minimal burning element $\beta$ of $M(\text{Rotor}(G, s))$ has $\beta^n = e$, whereas the corresponding exponent for $\beta$ in $M(\text{Sand}(G, s))$ grows quadratically in $n$.

Recall that there are many distinct rotor networks $\text{Rotor}(G, s)$ depending on the choice of ordering of the outgoing edges of each vertex. An interesting question is whether $M$ can distinguish between these networks.
Acknowledgments

This research was supported by an NSF postdoctoral fellowship and NSF grants DMS-1105960 and DMS-1243606, and by the UROP and SPUR programs at MIT. A few of the concluding remarks were inspired by discussions at the AIM workshop on generalizations of chip-firing and the critical group in July, 2013. The full list of open problems proposed at the workshop can be found at http://aimath.org/WWN/chipfiring/aim_chip-firing_problems.pdf.

References


Benjamin Bond, Department of Mathematics, Stanford University, Stanford, California 94305. http://stanford.edu/~benbond

Lionel Levine, Department of Mathematics, Cornell University, Ithaca, NY 14853. http://www.math.cornell.edu/~levine