

ABELIAN NETWORKS I. FOUNDATIONS AND EXAMPLES

BEN BOND AND LIONEL LEVINE

ABSTRACT. In Dhar’s model of abelian distributed processors, finite automata occupy the vertices of a graph and communicate via the edges. A local commutativity condition ensures that the output of such a network does not depend on the order in which the automata process their inputs. In this paper (the first of a series) we consider the halting problem for such networks and the critical group, an invariant that governs the behavior of the network on large inputs. Our main results are 1. A finite abelian network halts on all inputs if and only if its Laplacian is positive definite; 2. The critical group of an irreducible abelian network acts freely and transitively on recurrent states of the network; 3. The critical group is 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.

1. INTRODUCTION

In recent years it has become clear that certain interacting particle systems studied in combinatorics and statistical physics have a common underlying structure. These systems are characterized by an *abelian property* which says changing the order of certain interactions has no effect on the final state of the system. Up to this point, the tools used to study these systems – least action principle, local-to-global principles, transition monoids and critical groups – have been developed piecemeal for each particular system. Following Dhar [Dha06], we aim to identify explicitly what these various systems have in common and exhibit them as special cases of what we call an *abelian network*.

The immediate payoff of this perspective is more general theorems with more conceptual proofs. For example, in [HLMPPW08] it was proved that the sandpile group of a graph G has a free and transitive action on the spanning trees of G . In Theorem 8.1 we generalize this result to irreducible abelian networks, and the proof shows how it is really an example of a general mechanism by which group actions arise from commutative monoid actions (Lemma 4.4).

Date: DRAFT of November 20, 2011.

Key words and phrases. abelian distributed processors, chip-firing, commutative monoid action, critical group, finite automata, graph Laplacian, halting problem, least action principle, rotor walk.

A second goal of this paper is to start a philosophical discussion about (non)commutativity. Just as in physics one infers from macroscopic observations the properties of microscopic particles that cannot be observed individually, we would like to be able to infer from large-scale behavior of a cellular automaton something about the local rules that generate that behavior. In particular, are there certain large-scale features that can only be produced by *noncommutative* local interactions?

Intuition suggests that noncommutativity is a major source of dynamical richness and complexity. Yet abelian networks are capable of producing surprisingly rich and intricate large-scale patterns from local rules [Ost03, DSC09, FL10]. To put the question in computational terms, the requirement that a distributed network produce the same output regardless of the order in which processors act would seem to place a severe restriction on the kinds of tasks it can perform. Yet abelian networks can perform some highly nontrivial tasks, such as solving certain integer programs (Corollary 5.4). Are there other computational tasks that *require* noncommutativity?

In this paper, by defining abelian networks and exploring their fundamental properties, we hope to take a step toward making these questions precise and eventually answering them. After giving the formal definition of an abelian network in §2, we investigate a number of examples in §3. Some background on monoid actions is reviewed in §4. The main results begin in §5, where we prove a least action principle for abelian networks and explore some of its consequences. One consequence is that “local abelianness implies global abelianness” in a sense we shall make precise. In §6-7 we give conditions for a finite abelian network to halt on all inputs. Such a network has a natural invariant attached to it, the *critical group*, which is a finite abelian group whose structure we investigate in §8.

This paper is intended as the first of series. Topics to be explored in future papers may include stochastic abelian networks, morphisms between networks, simulation of one abelian network by another, computational strength of various classes of abelian networks, non-unary networks and abelian networks on infinite graphs.

2. DEFINITION OF AN ABELIAN NETWORK

This section begins with the formal definition of an abelian network, which is based on Deepak Dhar’s model of “abelian distributed processors” [Dha06]. The term “abelian network” is convenient when one wants to refer to a collection of communicating processors as a single entity. Some readers may wish to look at the examples in §3 before reading this section in detail.

Let $G = (V, E)$ be a directed graph, which may have self-loops and multiple edges. Associated to each vertex $v \in V$ is a *processor* \mathcal{P}_v , which is an

automaton with a single input feed and multiple output feeds, one for each edge $(v, u) \in E$. Each processor reads the letters in its input feed in first-in-first-out order.

The processor \mathcal{P}_v has an input alphabet A_v and state space Q_v . Its behavior is governed by a *transition function* T_v and *message passing functions* $T_{(v,u)}$ associated to each edge $(v, u) \in E$. Formally, these are maps

$$\begin{aligned} T_v &: A_v \times Q_v \rightarrow Q_v && \text{(new internal state)} \\ T_{(v,u)} &: A_v \times Q_v \rightarrow A_u^* && \text{(messages sent from } v \text{ to } u) \end{aligned}$$

Here A_u^* denotes the free monoid on the alphabet A_u . We interpret these functions as follows. If the processor \mathcal{P}_v is in state q and processes input a , then two things happen:

- (1) Processor \mathcal{P}_v transitions to state $q' = T_v(a, q)$; and
- (2) For each edge $(v, u) \in E$, processor \mathcal{P}_u receives input $T_{(v,u)}(a, q')$.

If more than one \mathcal{P}_v has inputs to process, then changing the order in which processors act may change the order of messages arriving at other processors. Concerning this issue, Dhar writes that

“In many applications, especially in computer science, one considers such networks where the speed of the individual processors is unknown, and where the final state and outputs generated should not depend on these speeds. Then it is essential to construct protocols for processing such that the final result does not depend on the order at which messages arrive at a processor.” [Dha06]

To realize this vision, we ask that the following aspects of the computation *do not depend on the order in which individual processors act*:

- (1) The **halting status** (i.e., whether or not processing eventually stops).
- (2) The **final output** (final states of the processors).
- (3) The **run time** (total number of letters processed by all \mathcal{P}_v).
- (4) The **local run times** (number of letters processed by a given \mathcal{P}_v).
- (5) The **specific local run times** (number of times a given \mathcal{P}_v processes a given letter $a \in A_v$).

A priori it is not obvious that these goals are actually achievable by any nontrivial network. We will show, however, that a simple local commutativity condition suffices to ensure all five goals are achieved. To state this condition, we extend the domain of T_v and $T_{(v,u)}$ to $A_v^* \times Q_v$: if $w = aw'$ is a word in alphabet A_v beginning with a , then set $T_v(w, q) = T_v(w', T_v(a, q))$ and $T_{(v,u)}(w, q) = T_{(v,u)}(a, q)T_{(v,u)}(w', T_v(a, q))$, where the product denotes concatenation of words.

Let \mathbb{N}^A be the free commutative monoid generated by A , and write $w \mapsto |w|$ for the natural map $A^* \rightarrow \mathbb{N}^A$, so that $|w|_a$ for $a \in A$ denotes the number of letters a in the word w .

Definition. (Abelian Processor) The processor \mathcal{P}_v is called *abelian* if for any words $w, w' \in A_v^*$ such that $|w| = |w'|$, we have for all $q \in Q_v$ and all edges $(v, u) \in E$

$$T_v(w, q) = T_v(w', q) \quad \text{and} \quad |T_{(v,u)}(w, q)| = |T_{(v,u)}(w', q)|.$$

That is, permuting the letters input to \mathcal{P}_v does not change the resulting state of the processor \mathcal{P}_v , and may change each output word sent to \mathcal{P}_u only by permuting its letters.

Definition. (Abelian Network) An *abelian network* on a directed graph $G = (V, E)$ is a collection of automata $\mathcal{N} = (\mathcal{P}_v)_{v \in V}$ indexed by the vertices of G , such that each \mathcal{P}_v is abelian.

We make two remarks about the definition:

1. The definition is *local* in the sense that it involves checking a condition on each processor individually. As we will see, these local conditions on the \mathcal{P}_v imply that the network as a whole is abelian in the sense that it satisfies conditions (1)-(5) above (Lemma 5.2).
2. A processor \mathcal{P}_v is called *unary* if its alphabet A_v has cardinality 1. A unary processor is trivially abelian, and any network of unary processors is an abelian network. Most of the examples of abelian networks studied so far (§3) are actually unary networks. Non-unary networks represent an interesting realm for future study.

Comparison with cellular automata. Before moving on to the examples, let us briefly compare and contrast abelian networks with cellular automata.

Abelian networks can update asynchronously. Traditional cellular automata update in parallel: at each time step, all cells *simultaneously* update their states based on the states of their neighbors. Since perfect simultaneity is hard to achieve in practice, the physical significance of parallel updating cellular automata is open to debate. Abelian networks do not require the kind of central control over timing needed to enforce simultaneous parallel updates, because they reach the same final state no matter in what order the updates occur.

Abelian networks do not rely on shared memory. Implicit in the update rule of cellular automata is an unspecified mechanism by which each cell is constantly kept informed of the states of its neighbors. The lower-level interactions needed to facilitate this exchange of information in a physical implementation are missing from the model. Abelian networks include these

lower-level interactions by operating in a “message passing” framework instead of the “shared memory” framework of cellular automata: An individual processor in an abelian network cannot access the states of neighboring processors, it can only read the messages they send.

Abelian networks can have any underlying geometry. Cellular automata are traditionally studied on the grid \mathbb{Z}^d or on other lattices, but they may be defined on any graph G . We would like to suggest that the study of cellular automata on G could be a fruitful means of revealing interesting graph-theoretic properties of G , and this is the perspective we take in developing the theory of abelian networks.

SUMMARY OF NOTATION

$G = (V, E)$	directed graph
d_v	outdegree of vertex v
\mathcal{P}_v	processor at vertex v
A_v	input alphabet of \mathcal{P}_v
Q_v	state space of \mathcal{P}_v
$\mathcal{N} = (\mathcal{P}_v)_{v \in V}$	abelian network
$A = \sqcup_{v \in V} A_v$	total alphabet
$Q = \prod_{v \in V} Q_v$	total state space
A^*	free monoid on alphabet A
\mathbb{N}^A	free commutative monoid on A
\mathbb{Z}^A	free abelian group generated by A
\mathbb{Q}^A	vector space over the rational numbers with basis A
$ w _a$	number of letters a in word w
T_v	transition function of vertex v
$T_{(v,u)}$	message passing function of edge (v, u)
$t(a)$	the map $q \mapsto T_v(a, q)$
M_v	transition monoid of \mathcal{P}_v , generated by $\{t(a)\}_{a \in A_v}$
e_v	minimal idempotent of M_v
$\mathbf{x} \cdot \mathbf{q}$	for $\mathbf{x} \in \mathbb{N}^A$, $\mathbf{q} \in Q$: processors are in state \mathbf{q} , with \mathbf{x}_a messages of type a present for each $a \in A$.
$\mathbf{x} \triangleright \mathbf{y} \cdot \mathbf{q}$	the result $\mathbf{y}' \cdot \mathbf{q}'$ of processing each message in \mathbf{x} once starting from $(\mathbf{x} + \mathbf{y}) \cdot \mathbf{q}$.
$\mathbf{x} \triangleright \triangleright \mathbf{q}$	the state \mathbf{q}' obtained by processing all messages in $\mathbf{x} \cdot \mathbf{q}$ until halting.
M	transition monoid of \mathcal{N} , generated by $\{\mathbf{q} \mapsto 1_a \triangleright \triangleright \mathbf{q}\}_{a \in A}$
e	minimal idempotent of M
$\text{Crit } \mathcal{N}$	critical group ($= eM$)
$\text{Rec } \mathcal{N}$	the set of recurrent states ($= eQ$)

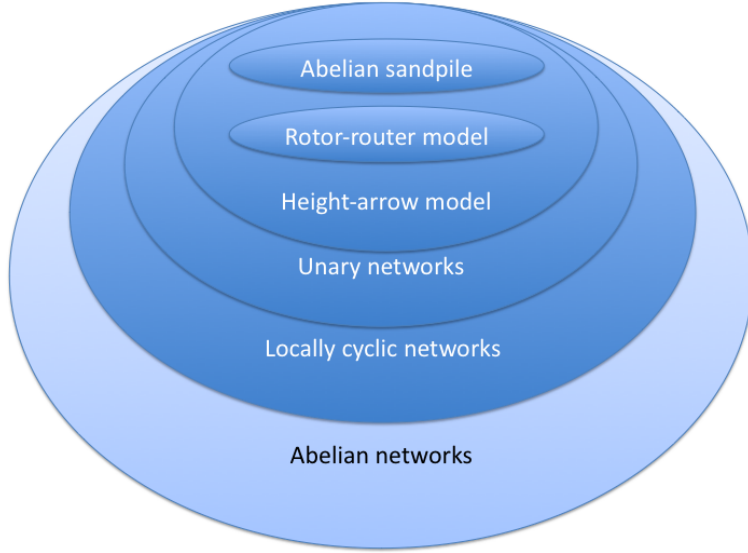


FIGURE 1. Venn diagram illustrating several classes of abelian networks.

3. EXAMPLES

Sandpile networks. Figure 1 shows increasingly general classes of abelian networks. The oldest and most studied is the *abelian sandpile model* [BTW87, Dha90], also called *chip-firing* [BLS91, Big99]. Given a directed graph $G = (V, E)$, the processor at each vertex $v \in V$ has input alphabet $A_v = \{v\}$ and state space $Q_v = \{0, 1, \dots, t_v - 1\}$, where t_v is the outdegree of v . The transition function is

$$T_v(q) = q + 1 \pmod{t_v}.$$

(Formally we should write $T_v(v, q)$, but when $\#A_v = 1$ we omit the redundant first argument.) The message passing functions are

$$T_{(v,u)}(q) = \begin{cases} u, & q = 0 \\ \epsilon, & q > 0 \end{cases}$$

for each edge $(v, u) \in E$. Here $\epsilon \in A^*$ denotes the empty word. Thus each time the processor at vertex v transitions from state $t_v - 1$ to state 0, it sends one message to each of its out-neighbors (Figure 2). When this happens we say that vertex v *topples* (or “fires”).

Toppling networks have the same transition and message passing functions as sandpiles, but we allow the number of states t_v to be different from the outdegree of v . These networks can be concretely realized in terms of “chips”: If a vertex in state q has k messages in its input feed, then we say that there

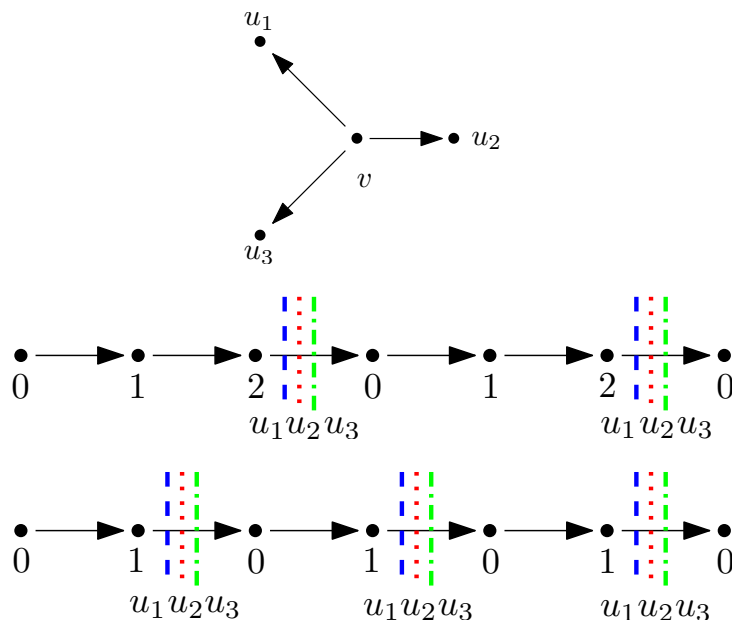


FIGURE 2. Top: portion of graph G showing a vertex v and its outneighbors u_1, u_2, u_3 . Middle: State diagram for v in a sandpile network. Dots represent states, arrows represent transitions when a message is processed, and vertical lines indicate when messages are sent to the neighbors. Bottom: Example state diagram for v in a toppling network with $t_v = 2$.

are $q + k$ chips at that vertex. When v has at least t_v chips, it can *fire*, losing t_v chips and sending one chip along each outgoing edge. In a sandpile network the total number of chips is conserved, but in a toppling network, chips may be created (if t_v is less than the outdegree of v , as in the last diagram of Figure 2) or destroyed (if t_v is larger than the outdegree of v).

Sinks and counters. It is common to consider sandpile networks with a *sink*, a vertex whose processor has only one state and never sends any messages. If every vertex of G has a directed path to the sink, then any finite input to the sandpile network will produce only finitely many topplings. A natural question is, when does the same finiteness hold in a general toppling network – which may have a mix of creative and destructive vertices, but perhaps no sink? We answer this question as a special case of the halting problem for abelian networks, treated in §6.

Note that when viewing a sandpile as an abelian network, some chips are “latent” in the sense that they are encoded by the internal states of the processors. For example if a vertex v of outdegree 2 is in state 0, receives one chip and processes it, then the message representing that chip is gone,

but the internal state increases to 1 representing a latent chip at v . If v receives another chip and processes it, its state returns to 0 and it topples by sending one message to each out-neighbor.

The set of *recurrent states* (defined in §8) of a sandpile network with sink is in bijection with objects of interest in combinatorics such as oriented spanning trees and G -parking functions.

A *counter* is a unary processor with state space \mathbb{N} and transition $T(q) = q + 1$, which never sends any messages. It behaves like a sink, but keeps track of how many messages it has received.

Sometimes it is useful to consider toppling networks where the number of chips at a vertex may become negative. We can model this by enlarging the state space of each processor \mathcal{P}_v to include $-\mathbb{N}$; these additional states have transition function $T_v(q) = q + 1$ and send no messages. In §5 we will see that these enlarged toppling networks solve certain integer programs.

Rotor networks. A *rotor* is a processor \mathcal{P}_v that outputs exactly one message for each message input. That is, for all $a \in A_v$ and all $q \in Q_v$

$$\sum_{(v,u) \in E} \sum_{b \in A_u} |T_{(v,u)}(a, q)|_b = 1.$$

By inputting a single message into a network of rotors with underlying graph G , we obtain in a natural way an infinite walk v_0, v_1, \dots in G . Vertex v_n is the location of the single message present after n processings. This *rotor walk* has been studied under various names: In computer science it was introduced as a model of autonomous agents exploring a territory (“ant walk,” [WLB96]) and studied as a means of broadcasting information through a network []. In statistical physics it was proposed as a model of self-organized criticality (“Eulerian walkers,” [PDDK96]). Propp proposed rotor walk as a way of derandomizing certain features of random walk [CS06, HP10, Pro10].

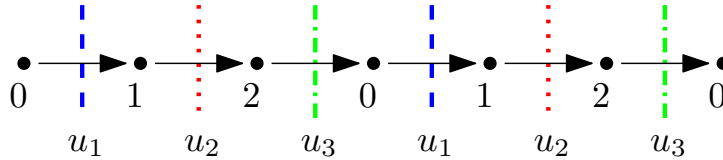


FIGURE 3. State diagram for a vertex v in a simple rotor network. The out-neighbors u_1, u_2, u_3 of v are served repeatedly in a fixed order.

Most commonly studied is the *simple* rotor network on a directed graph G , in which the out-neighbors of vertex v are served repeatedly in a fixed order u_1, \dots, u_{d_v} (Figure 3). Formally, we set $Q_v = \{0, 1, \dots, d_v - 1\}$, and

$A_v = \{v\}$, with transition function $T_v(q) = q + 1 \pmod{d_v}$ and message passing functions

$$T_{(v,u_j)}(q) = \begin{cases} u_j, & q \equiv j \pmod{d_v} \\ \epsilon, & q \not\equiv j \pmod{d_v}. \end{cases}$$

A state of a simple rotor network with sink corresponds to a choice of one outgoing edge from each non-sink vertex (indicating where the last message was sent from that vertex). The recurrent states correspond to choices in which these edges form a spanning tree of G oriented toward the sink. In particular, the recurrent rotor states are equinumerous with the recurrent sandpile states on the same graph. This curious fact has inspired several different bijections between recurrent sandpile states and spanning trees []. Recurrent sandpile states have a natural group structure (about which we say more in §8) whereas oriented spanning trees do not. However, interpreting an oriented spanning tree as a state of a rotor network allows one to construct a free transitive action of the sandpile group on spanning trees [HLMPPW08], which “explains” why these sets are equinumerous. In §8 we show that this group action is a general phenomenon for irreducible abelian networks.

Another process of interest is *rotor-router aggregation*, proposed by Propp [Pro04]. Enlarge each state space Q_v to include a transient state -1 , which transitions to state 0 but emits no message. Starting with all processors in state -1 , the effect is that each vertex “absorbs” the first message it receives, and behaves like a rotor thereafter. If we input n messages to one vertex v_0 , then each message performs a rotor walk starting from v_0 until reaching a site that has not yet been visited by any previous walk, where it gets absorbed. When the underlying graph is \mathbb{Z}^2 , the resulting set of n visited sites is extremely close to circular [LP09], and the final states of the processors display extremely intricate patterns that are still not well understood [FL10].

Height arrow model. Dartois and Rossin [DR04] proposed a common generalization of rotor and sandpile networks called the *height-arrow model*. An example state diagram is shown in Figure 4. For each vertex v we set $A_v = \{v\}$ and choose a threshold $1 \leq \tau_v \leq d_v$, where d_v is the out-degree of v . We set $Q_v = \{0, 1, \dots, d_v\tau_v - 1\}$ with transition function $T_v(q) = q + 1 \pmod{d_v\tau_v}$. Fix an ordering u_1, \dots, u_{d_v} of the out-neighbors of v . The message passing function for the edge (v, u_j) is given by

$$T_{(v,u_j)}(q) = \begin{cases} u_j & q \equiv 0 \pmod{\tau_v} \text{ and } q - \tau_v \pmod{d_v} < j \leq q \pmod{d_v} \\ \epsilon & \text{else.} \end{cases}$$

We think of the state q as representing an arrow pointing to the neighbor $u_{q \bmod d_v}$ and a number of chips $q \pmod{\tau_v}$. When vertex v collects τ_v

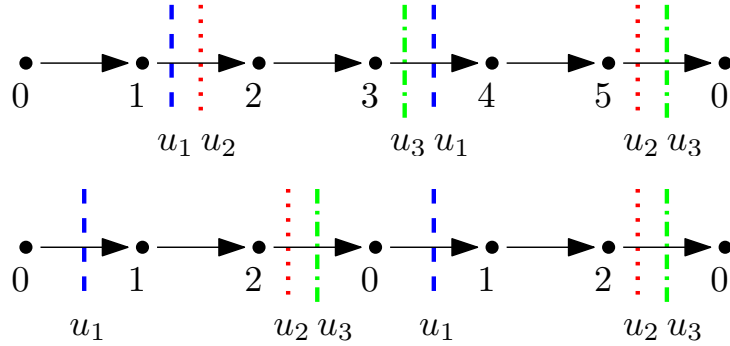


FIGURE 4. Example state diagrams for a vertex v in the Dartois-Rossin height arrow model with $d_v = 3$ and $\tau_v = 2$ (top) and Eriksson’s periodically mutating game (bottom).

chips, it sends one chip along each of the most τ_v recent arrows to neighbors $u_{q-\tau_v+1}, \dots, u_q$ (indices mod d_v).

Unary networks. Diaconis and Fulton [DF91] and Eriksson [Eri96] studied generalizations of chip-firing in which each vertex has a stack of instructions. When a vertex accumulates enough chips to follow the top instruction in its stack, it pops that instruction off the stack and follows it. These and all preceding examples are *unary networks*, that is, abelian networks in which each alphabet A_v has cardinality 1. Informally, a unary network on a graph G is a system of local rules by which *indistinguishable* chips move around on the vertices of G .

Next we discuss some non-unary examples.

Bootstrap percolation. In this simple model of crack formation, each vertex v has a threshold t_v , often chosen to be half its indegree (rounded up). Site v becomes “infected” as soon as at least t_v of its in-neighbors are infected. A question that has received a lot of attention due to its subtle scaling behavior is what initial density of random infected sites causes the entire graph to become infected [Ent87, Hol03]. To realize bootstrap percolation as an abelian network, we take A_v to be the set $N_{in}(v)$ of in-neighbors of v and $Q_v = 2^{N_{in}(v)}$ to be its power set. The transition and message passing functions are given by

$$T_v(a, q) = q \cup \{a\}$$

and

$$T_{(v,u)}(a, q) = \begin{cases} \epsilon, & \#q \neq t_v \\ v, & \#q = t_v. \end{cases}$$

When a site becomes infected, it informs its out-neighbors, and each processor’s internal state keeps track of which of its neighbors have been infected.

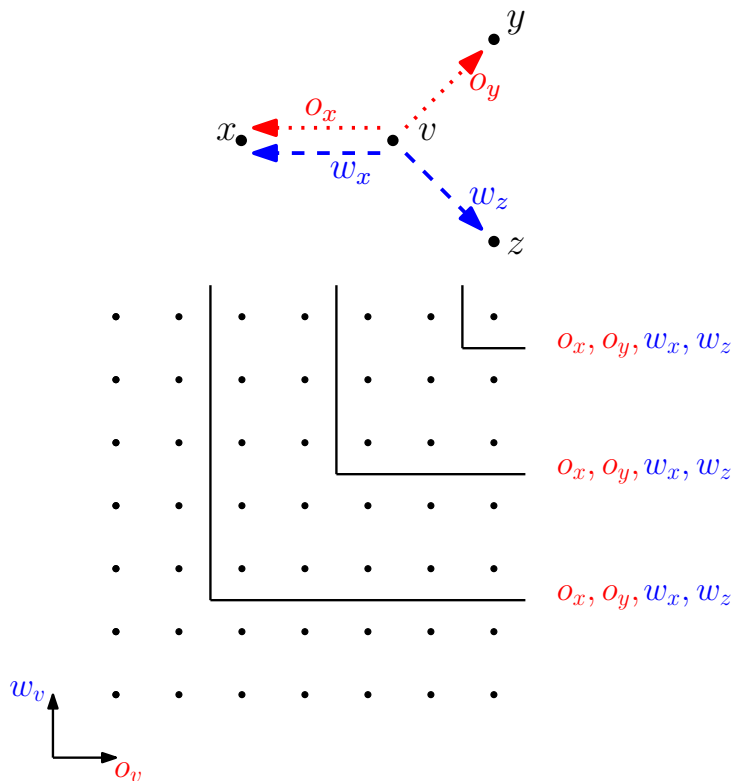


FIGURE 5. Example state diagram for the oil and water model. Top: Vertex v has outgoing oil edges to x and y , and water edges to x and z . Bottom: each dot represents a state in $Q_v = \mathbb{N} \times \mathbb{N}$. A toppling occurs each time the state transition crosses one of the bent lines (for example, by processing an oil message o_v in state $(1, 2)$, resulting in transition to state $(2, 2)$).

Oil and water model. This is a non-unary generalization of sandpiles, inspired by Paul Tseng’s asynchronous algorithm for solving certain linear programs [Tse90].

We first give an informal description in terms of oil chips and water chips. Each edge of the graph is marked either as an oil edge or a water edge, or both. When a vertex topples, it sends out one oil chip along each outgoing oil edge and also one water chip along each outgoing water edge. The interaction between oil and water is that a vertex v is permitted to topple if and only if sufficiently many chips of *both* types are present at v .

In the preceding examples, each processor had a finite state space Q_v . In the oil and water model, however, an arbitrary number of oil chips could accumulate at vertex v and be unable to topple if there are not enough water

chips there. We set $Q_v = \mathbb{N} \times \mathbb{N}$ and $A_v = \{o_v, w_v\}$, with transition function

$$T_v(o_v, q) = q + (0, 1), \quad T_v(w_v, q) = q + (1, 0).$$

Thus the internal state of the processor at v is a vector $q = (q_{oil}, q_{water})$ keeping track of the total number chips of each type it has received (Figure 5).

Write $G = (V, E)$ where $E = E_{oil} \sqcup E_{water}$ (it is convenient although not necessary to make this a disjoint union, so that if some edge is marked both oil and water then we regard it as two distinct edges in the multigraph G). Let d_{oil} and d_{water} be respectively the number of outgoing oil edges and water edges from v . The message passing function for an oil edge (v, u) is given by

$$T_{(v,u)}(o_v, q) = \begin{cases} o_u & \text{if } q_{oil} \equiv 0 \pmod{d_{oil}} \text{ and } q_{water}/d_{water} \geq q_{oil}/d_{oil} \\ \epsilon & \text{else.} \end{cases}$$

$$T_{(v,u)}(w_v, q) = \begin{cases} o_u & \text{if } q_{water} \equiv 0 \pmod{d_{water}} \text{ and } q_{water}/d_{water} \leq q_{oil}/d_{oil} \\ \epsilon & \text{else.} \end{cases}$$

The message passing function for a water edge (v, u) is defined similarly, with the roles of oil and water reversed.

Stochastic abelian networks. In a stochastic abelian network, we allow the transition functions (but not the message passing functions) to depend on a probability space Ω :

$$T_v : A_v \times Q_v \times \Omega \rightarrow Q_v \quad (\text{new internal state})$$

$$T_{(v,u)} : A_v \times Q_v \rightarrow A_u^* \quad (\text{messages sent from } v \text{ to } u)$$

A variety of models in statistical mechanics — including classical Markov chains and branching random walk, certain directed edge-reinforced walks, the Manna model [Man91], internal DLA [LBG92], the Oslo model [Fre93], excited walk [BW03], the Kesten-Sidoravicius infection model [KS05, KS08], low-discrepancy random stack [FL10], activated random walkers [DRS10] and stochastic sandpiles [RS11] — can all be realized as stochastic abelian networks. In at least one case [RS11] the abelian nature of the model enabled a major breakthrough in proving the existence of a phase transition. Stochastic abelian networks are beyond the scope of the present paper and will be treated in a sequel.

4. MONOID ACTIONS AND OTHER PRELIMINARIES

We collect here some algebraic and combinatorial background to be used in subsequent sections. The lemmas below on finite commutative monoid actions can probably be derived with some effort by specializing classical results of Green [Gre51] and Schützenberger [Sch57] to the commutative case; see [Ste10] for a modern treatment. We include their short proofs here

in order to highlight the beauty and simplicity of the commutative case. Our approach continues in the vein of Babai and Toumpakari [BT10], who observed that certain oftenly used facts about sandpiles have purely monoid-theoretic proofs.

4.1. Commutative monoid actions. We briefly review the theory of finite commutative monoids. Such a monoid M contains an abelian group as its minimal ideal, and every action of M induces a corresponding group action. The main result of this section is Lemma 4.4 relating these two actions.

Let M be a finite commutative monoid, with identity element denoted by 0 . Let

$$\mu : M \times X \rightarrow X$$

be an action of M on a set X , i.e., $0x = x$ and $m(m'x) = (mm')x$ for all $m, m' \in M$ and all $x \in X$. We say that μ is *irreducible* if there does not exist a partition of X into nonempty subsets X_1 and X_2 such that $MX_1 \subset X_1$ and $MX_2 \subset X_2$.

Lemma 4.1. *If μ is irreducible, then for any $x, x' \in X$ there exist $m, m' \in M$ such that $mx = m'x'$.*

Proof. Define $x \sim x'$ if there exist $m, m' \in M$ such that $mx = m'x'$. Since M is commutative, \sim is an equivalence relation: to check transitivity, note that $mx = m'x'$ and $m''x' = m'''x''$ imply

$$m''mx = m''m'x' = m'm''x' = m'm'''x''$$

so that $x \sim x''$.

Now fix $x \in X$ and let $X_1 = \{x' \in X \mid x' \sim x\}$ be the equivalence class of x . Let $X_2 = X - X_1$. For any $x' \in X$ and any $m \in M$ we have $x' \in X_1$ if and only if $mx' \in X_1$. Thus $MX_1 \subset X_1$ and $MX_2 \subset X_2$. Since μ is irreducible and X_1 is nonempty (it contains x) we conclude that $X_1 = X$. \square

In general, if μ is not irreducible then X can be written as a disjoint union of irreducible components X_α , which are the equivalence classes of the relation \sim defined in the proof of Lemma 4.1.

An *ideal* of M is a subset $I \subset M$ such that $MI \subset I$. The intersection of ideals is again an ideal. The minimal ideal

$$J = \bigcap_{\text{ideals } I \subset M} I$$

is a nonempty abelian group. Denote its identity element by e . Then $J = eM$. In particular, $e \neq 0$ unless $J = M$. One way to construct e explicitly is

$$e = \prod_{x \text{ idempotent}} x$$

where the product is over all idempotent elements of $x \in M$ (elements such that $xx = x$). Since M is commutative, the product of idempotents is again an idempotent. A characterizing property of e is that it is the unique idempotent accessible from all of M : that is, $ee = e$ and $e \in mM$ for all $m \in M$.

Lemma 4.2. *Let M be a finite commutative monoid and $\mu : M \times X \rightarrow X$ an irreducible action. The following are equivalent for $x \in X$:*

- (1) $x \in My$ for all $y \in X$.
- (2) $x \in mX$ for all $m \in M$.
- (3) $x \in eX$.
- (4) $x = ex$.

Proof. (1) \Rightarrow (2): Fix $m \in M$ and let $y = mx$. If $x \in My$, then there exists $m' \in M$ such that $x = m'y = m'mx = mm'x \in mX$.

(2) \Rightarrow (3): trivial.

(3) \Rightarrow (4): If $x = ey$, then $ex = e(ey) = (ee)y = ey = x$.

(4) \Rightarrow (1): By Lemma 4.1, given $x, y \in X$ there exist $m, m' \in M$ such that $mx = m'y$. Let m'' be such that $m''m = e$. If $x = ex$, then $x = m''mx = m''m'y \in My$. \square

Any commutative monoid action

$$\mu : M \times X \rightarrow X$$

induces by restriction a corresponding group action

$$eM \times eX \rightarrow eX.$$

To see this, note that for any $m \in M$ and $x \in X$ we have $m(ex) = (me)x = (em)x = e(mx) \in eX$, so the action of M on X restricts to a monoid action of M on eX . Since $e(ex) = (ee)x = ex$, the element e acts by identity on eX . Since eM is a group with identity element e , it follows that $eM \times eX \rightarrow eX$ is a group action.

We say that $m \in M$ acts *invertibly* on a subset $Y \subset X$ if $mY = Y$ (so that $y \mapsto my$ is a permutation of Y).

Lemma 4.3. *Let M be a finite commutative monoid and $\mu : M \times X \rightarrow X$ a monoid action. Then every $m \in M$ acts invertibly on eX .*

Proof. For any $m \in M$ and $x \in X$ we have $(em)(ex) = (eme)x = (mee)x = (me)x = m(ex)$, so em and m have the same action on eX . Since $eM \times eX \rightarrow eX$ is a group action, em and hence m acts invertibly on eX . \square

We say that a monoid action $\mu : M \times X \rightarrow X$ is *faithful* if there do not exist distinct elements $m, m' \in M$ such that $mx = m'x$ for all $x \in X$. The next lemma shows that relatively weak properties of a monoid action

(faithful, irreducible) imply much stronger properties of the corresponding group action (free, transitive).

Let G be a group with identity element e . Recall that a group action $G \times Y \rightarrow Y$ is called *transitive* if $Gy = Y$ for all $y \in Y$, and *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 pair $y, y' \in Y$ there is a unique $g \in G$ such that $gy = y'$. In particular, $\#G = \#Y$.

Lemma 4.4. *Let M be a finite commutative monoid. If $\mu : M \times X \rightarrow X$ is an irreducible monoid action, then the restriction of μ to $eM \times eX$ is a transitive group action*

$$eM \times eX \rightarrow eX.$$

If in addition μ is faithful, then this group action is free.

Proof. To show transitivity, let $R = \bigcap_{y \in eX} (My)$. Then for any $x \in eX$ we have

$$R \subset Mx = M(ex) = (Me)x = (eM)x = e(Mx) \subset eX.$$

But $R = eX$ by the equivalence of (1) and (3) in Lemma 4.2, so the above inclusions are equalities. In particular, $(eM)x = eX$ for all $x \in eX$, which shows that eM acts transitively on eX .

To show freeness, suppose that $g(ex) = ex$ for some $g \in eM$ and some $x \in X$. By transitivity, for any $y \in X$ we have $ey = h(ex)$ for some $h \in eM$, hence

$$gey = ghex = hgex = hex = ey.$$

Thus $(ge)y = ey$ for all $y \in X$. Since the action of M on X is faithful, we conclude that $ge = e$ and hence $g = e$. \square

4.2. Sequences in \mathbb{N}^k . The following lemma follows from the Hilbert basis theorem applied to the monomial ideal $(\mathbf{t}^{\mathbf{x}_1}, \mathbf{t}^{\mathbf{x}_2}, \dots)$ in the polynomial ring $\mathbb{Q}[\mathbf{t}] = \mathbb{Q}[t_1, \dots, t_k]$. For the sake of completeness we give a self-contained proof.

Lemma 4.5. *Let $k \geq 1$ be an integer. For any sequence $\mathbf{x}_1, \mathbf{x}_2, \dots \in \mathbb{N}^k$, there exist indices $i < j$ such that $\mathbf{x}_i \leq \mathbf{x}_j$ in the coordinatewise partial ordering.*

Proof. Induct on k . The base case $k = 1$ follows from the fact that \mathbb{N} is well-ordered. If $k \geq 2$, then \mathbb{N}^k is the union of $\mathbf{x}_1 + \mathbb{N}^k$ with a finite number of hyperplanes

$$H_{i,a} = \{\mathbf{y} \in \mathbb{N}^k \mid \mathbf{y}_i = a\}$$

for $i = 1, \dots, k$ and $a = 0, \dots, \mathbf{x}_{1i}$. If $\mathbf{x}_j \in \mathbf{x}_1 + \mathbb{N}^k$ for some $j > 1$, then $\mathbf{x}_1 \leq \mathbf{x}_j$. Otherwise, by the infinite pigeonhole principle, one of the hyperplanes $H_{i,a}$ contains infinitely many terms of the sequence \mathbf{x}_j . Since

$H_{i,a} \simeq \mathbb{N}^{k-1}$ as partially ordered sets, the proof is complete by the inductive hypothesis. \square

4.3. Nonnegative matrices. We will use the following form of the Perron-Frobenius theorem (see, for example, [HJ90, §8] and [Ash87]).

Lemma 4.6. (*Perron-Frobenius*) *Let P be a square matrix with nonnegative real entries. Then P has a nonnegative real eigenvector $\mathbf{x} \geq \mathbf{0}$ with nonnegative real eigenvalue λ , such that the absolute values of all other eigenvalues of P are $\leq \lambda$. Moreover if P has rational entries and λ is rational, then \mathbf{x} can be taken to have integer entries.*

5. LEAST ACTION PRINCIPLE

We begin our study of abelian networks by proving a least action principle, Lemma 5.1. The least action principle says — in a sense to be made precise — that each processor in an abelian network performs the minimum amount of work possible to remove all messages from the network. Various special cases of the least action principle to particular abelian networks have enabled a flurry of recent progress: bounds on the growth rate of sandpiles [FLP10], an exact shape theorem [KL10] and a fast simulation algorithm for growth models [FL10], and proof of a phase transition for activated random walkers [RS11].

The proof of the least action principle follows Diaconis and Fulton [DF91, Theorem 4.1]. Our observation is that their proof actually shows something stronger: it applies to any abelian network, not just to the models studied in [DF91]. Moreover, it applies even to executions that are complete but not legal. To explain the last point requires a few definitions.

Let \mathcal{N} be an abelian network with underlying graph $G = (V, E)$, total state space $Q = \prod Q_v$ and total alphabet $A = \sqcup A_v$. In this section we do not place any finiteness restrictions on \mathcal{N} : the underlying graph may be finite or infinite, and the state space Q_v and alphabet A_v of each processor may be finite or infinite. An *execution* is a word $w = w_1 \cdots w_r \in A^*$. It prescribes an order in which messages in the network are to be processed. For simplicity, we restrict ourselves to finite executions. If V is infinite, then infinite executions (and non-sequential execution procedures) are of interest [FMR09]; but we do not want to obscure the picture by introducing too many complications here.

It will often be convenient to view the entire network \mathcal{N} as a single automaton with state space $\mathbb{Z}^A \times Q$. For its states we will use the notation $\mathbf{x}.\mathbf{q}$, where $\mathbf{x} \in \mathbb{Z}^A$ and $\mathbf{q} \in Q$. The states $\mathbf{x}.\mathbf{q}$ with $\mathbf{x} \in \mathbb{N}^A$ are called *proper*. The proper state $\mathbf{x}.\mathbf{q}$ corresponds to the configuration of the network \mathcal{N} such that

- For each $a \in A$, there are \mathbf{x}_a messages of type a present; and
- For each $v \in V$, the processor at vertex v is in state q_v .

Note that $\mathbf{x}.\mathbf{q}$ encodes only the states of the processors and the *number* of messages present of each type. It gives no information about the order in which messages are to be processed. Indeed, one of our goals is to show that the order does not matter (Lemma 5.2).

We use the symbol \triangleright to denote the state transitions on $\mathbb{Z}^A \times Q$. For any $v \in V$ and $a \in A_v$ we define

$$a \triangleright (\mathbf{x}.\mathbf{q}) = \mathbf{x}'.\mathbf{q}'$$

where $\mathbf{q}' \in Q$ and $\mathbf{x}' \in \mathbb{Z}^A$ are obtained as follows. Let $q'_v = T_v(a, q_v)$ and let $q'_u = q_u$ for all $u \neq v$. Let

$$\mathbf{x}' = \mathbf{x} - \delta_a + N(a, q_v)$$

where δ_{ab} is 1 if $a = b$ and 0 otherwise; and $N(a, q_v)_b$ is the number of b 's produced when processor \mathcal{P}_v in state q_v processes the letter a . In other words, if $b \in A_u$, then

$$N(a, q_v)_b := \begin{cases} |T_{(v,u)}(a, q'_v)|_b, & \text{if } (v, u) \in E \\ 0, & \text{else.} \end{cases}$$

Fix an initial state $\mathbf{x}.\mathbf{q}$ and an execution $w = w_1 \cdots w_r \in A^*$. Set $\mathbf{x}^0 = \mathbf{x}$, $\mathbf{q}^0 = \mathbf{q}$ and

$$\mathbf{x}^i.\mathbf{q}^i = w_i \triangleright (\mathbf{x}^{i-1}.\mathbf{q}^{i-1}), \quad i = 1, \dots, r.$$

The *result* of executing w is $w \triangleright (\mathbf{x}.\mathbf{q}) := \mathbf{x}^r.\mathbf{q}^r$. Our goal is to compare the results of different executions.

For each $i = 1, \dots, r$ we have $w_i \in A_{v_i}$ for some vertex v_i . Let

$$N(w, \mathbf{q}) := \sum_{i=1}^r N(w_i, q_{v_i}^{i-1})$$

so that $N(w, \mathbf{q})_b$ is the total number of b 's produced by executing w starting from state \mathbf{q} .

By the definition of abelian network, $N(w, \mathbf{q})$ depends only on $|w|$ and \mathbf{q} , and if $|w| \leq |w'|$ then $N(w, \mathbf{q}) \leq N(w', \mathbf{q})$. (Recall that $|w| \in \mathbb{N}^A$ and $|w|_a$ is the number of occurrences of letter a in word w ; here and throughout, inequalities on vectors are coordinatewise.)

A letter $a \in A$ is called a *legal move* from $\mathbf{x}.\mathbf{q}$ if $\mathbf{x}_a \geq 1$. An execution $w_1 \cdots w_r$ is called *legal* for $\mathbf{x}.\mathbf{q}$ if w_{i+1} is a legal move from $\mathbf{x}^i.\mathbf{q}^i$ for each $i = 0, \dots, r-1$. An execution $w_1 \cdots w_r$ is called *complete* for $\mathbf{x}.\mathbf{q}$ if $\mathbf{x}_a^r \leq 0$ for all $a \in A$.

Lemma 5.1. (*Least Action Principle*) *If $w = w_1 \cdots w_r$ is legal for $\mathbf{x}.\mathbf{q}$ and $w' = w'_1 \cdots w'_s$ is complete for $\mathbf{x}.\mathbf{q}$, then $|w| \leq |w'|$ and $r \leq s$.*

Proof. Noting that $r = \sum_{a \in A} |w|_a$ and $s = \sum_{a \in A} |w'|_a$, it suffices to prove $|w| \leq |w'|$. Supposing for a contradiction that $|w| \not\leq |w'|$, let i be the smallest index such that $|w_1 \cdots w_i| \not\leq |w'|$. Let $u = w_1 \cdots w_{i-1}$ and $a = w_i$. Then $|u|_a = |w'|_a$, and $|u|_b \leq |w'|_b$ for all $b \neq a$. Since a is a legal move from $\mathbf{x}^{i-1}.\mathbf{q}^{i-1}$, we have

$$\begin{aligned} 1 &\leq \mathbf{x}_a^{j-1} \\ &= \mathbf{x}_a^0 - |u|_a + N(u, \mathbf{q}^0)_a \\ &\leq \mathbf{x}_a^0 - |w'|_a + N(w', \mathbf{q}^0)_a \\ &= \mathbf{x}_a'^s. \end{aligned}$$

Since w' is complete, $\mathbf{x}_a'^s \leq 0$, which yields the required contradiction. \square

Lemma 5.2. (*Halting Dichotomy*) *For a given initial state \mathbf{q} and input \mathbf{x} to an abelian network \mathcal{N} , either*

- (1) *There does not exist a finite complete execution; or*
- (2) *Every legal execution is finite, and any two complete legal executions w, w' satisfy $|w| = |w'|$.*

Proof. If there exists a finite complete word, say of length s , then every legal word has length $\leq s$ by Lemma 5.1. If w and w' are complete legal words, then $|w| \leq |w'| \leq |w|$ by Lemma 5.1. \square

The *halting problem* for abelian networks asks, given \mathcal{N} , \mathbf{x} and \mathbf{q} , whether (1) or (2) of Lemma 5.2 is the case. In case (2) we say that \mathcal{N} *halts* on input $\mathbf{x}.\mathbf{q}$.

Our next lemma illustrates a general theme of *local-to-global principles* in abelian networks. Suppose we are given a partition $V = I \sqcup O$ of the vertex set into “interior” and “output” nodes. The processor at each output node is a counter (§3. If \mathcal{N} halts on all inputs, then we can regard the induced subnetwork $(\mathcal{P}_v)_{v \in I}$ of interior nodes as a single processor \mathcal{P}_I with input alphabet $\sqcup_{v \in I} A_v$, state space $Q_I := \prod_{v \in I} Q_v$, and an output feed for each edge $(v, u) \in I \times O$.

For notational convenience in the proof below, we extend the domain of T_v and $T_{(v,u)}$ to $A^* \times Q$ by setting $T_v(a, q) = T_v(w, q)$ and $T_{(v,u)}(a, q) = \epsilon$ whenever $a \notin A_v$, where ϵ is the empty word.

Lemma 5.3. (*Local Abelianness Implies Global Abelianness*) *If \mathcal{N} halts on all inputs and \mathcal{P}_v is an abelian processor for each $v \in I$, then \mathcal{P}_I is an abelian processor.*

Proof. Fix an initial state $q \in Q_I$. Two inputs ι, ι' to \mathcal{P}_I such that $\iota \sim \iota'$ correspond to the same extended state $|\iota|.q$ of \mathcal{P}_I . By Lemma 5.2, any two complete legal executions w, w' have the same specific runtimes: $|w| = |w'|$.

Since \mathcal{P}_v is abelian and $|w|_a = |w'|_a$ for all $a \in A_v$, we have $T_v(w, q_v) = T_v(w', q_v)$. So the final state of Q_I does not depend on the order of input. Moreover, for each edge $(v, u) \in I \times O$ and each we have

$$|T_{(v,u)}(w, q_v)| = |T_{(v,u)}(w', q_v)|$$

so the number of messages sent along (v, u) does not depend on the order of input. \square

For another example of a local-to-global principle, see Lemma 6.2. Further local-to-global principles in the case of rotor networks are explored in [GLPZ11].

Example. Consider a finite toppling network (§3) with underlying multigraph $G = (V, E)$ and threshold vector $\mathbf{t} = (t_v)_{v \in V}$. Consider the $V \times V$ Laplacian matrix

$$L = D - A$$

where D is the diagonal matrix $D_{vv} = t_v$, and A_{uv} is the number of directed edges from u to v . If the network halts on a given input \mathbf{s}, \mathbf{q} , then for each $v \in V$ let \mathbf{x}_v be the number of topplings that occur at vertex v . The vector $\mathbf{x} = (\mathbf{x}_v)_{v \in V}$ is called the *odometer* of \mathbf{s}, \mathbf{q} .

Corollary 5.4. (*Toppling Networks Solve Certain Integer Programs*) Let \mathcal{N} be a finite toppling network with threshold \mathbf{t} and Laplacian L . Fix $\mathbf{a}, \mathbf{b} \in \mathbb{Z}^V$ with $\mathbf{a}_v > 0$ for all $v \in V$. Let $\mathbf{q}_v = \min(\mathbf{b}_v + t_v - 1, 0)$ and $\mathbf{s}_v = \max(\mathbf{b}_v + t_v - 1, 0)$. If \mathcal{N} halts on input \mathbf{s}, \mathbf{q} , then the odometer of \mathbf{s}, \mathbf{q} is the unique solution $\mathbf{x} \in \mathbb{Z}^V$ to the integer program

$$\begin{aligned} & \text{Minimize} && \mathbf{a}^T \mathbf{x} \\ & \text{subject to} && \mathbf{x} \geq \mathbf{0} \quad \text{and} \quad L^T \mathbf{x} \geq \mathbf{b}. \end{aligned} \quad (1)$$

If \mathcal{N} does not halt on input \mathbf{s}, \mathbf{q} , then (1) has no solution $\mathbf{x} \in \mathbb{Z}^V$.

Proof. Toppling vertex v decreases the number of chips at v by t_v and increases the number of chips at each $u \in V$ by A_{vu} . At the end of a complete execution, the number of chips at each vertex v is at most $t_v - 1$. Hence, if $\mathbf{x}_v \geq 0$ is the number of times vertex v topples in a finite complete execution for \mathbf{s}, \mathbf{q} , then

$$\mathbf{q} + \mathbf{s} - L^T \mathbf{x} \leq \mathbf{t} - \mathbf{1}$$

so \mathbf{x} satisfies (1). Conversely, if $\mathbf{x} \in \mathbb{Z}^V$ is any solution of (1), then there exists a finite complete execution in which each vertex v topples \mathbf{x}_v times.

If \mathcal{N} halts on input \mathbf{s}, \mathbf{q} , then by Lemma 5.1, the odometer \mathbf{x} is given by

$$\mathbf{x}_v = \min\{|w|_v : w \text{ is a complete execution for } \mathbf{s}, \mathbf{q}\}.$$

Hence the odometer \mathbf{x} simultaneously minimizes $\mathbf{a}_v \mathbf{x}_v$ for all $v \in V$, among solutions $\mathbf{x} \in \mathbb{Z}^V$ of (1). In particular, it minimizes $\mathbf{a}^T \mathbf{x}$. \square

Remark. An interesting feature of the integer program (1) is that the solution does not depend on the vector \mathbf{a} provided $a_v > 0$ for all $v \in V$.

Strictly speaking, the topping network “computes” not \mathbf{x} but $\mathbf{q} + \mathbf{s} - L^T \mathbf{x}$. However, it is easy to design an abelian network that computes \mathbf{x} itself. For each $v \in V$ add a vertex v' whose processor is a counter (§3), and have v send one message to v' each time v topples. Then the final states of the processors in V' are exactly \mathbf{x} .

6. HALTING PROBLEM

If (2) holds for all inputs $\mathbf{x} \in \mathbb{N}^A$ and all states $\mathbf{q} \in Q$, then we say that \mathcal{N} *halts on all inputs*. In this section we give several conditions equivalent to the statement that \mathcal{N} halts on all inputs.

Definition. A proper state $\mathbf{x} \cdot \mathbf{q}$ is an *amplifier* if $\mathbf{x} \neq \mathbf{0}$ and there exists a sequence of legal moves m_1, \dots, m_r from $\mathbf{x} \cdot \mathbf{q}$ such that $m_1 \cdots m_r(\mathbf{x} \cdot \mathbf{q}) = \mathbf{y} \cdot \mathbf{q}$ for some $\mathbf{y} \geq \mathbf{x}$.

Example. Consider a abelian network with $V = \{i\}$, $E = \{(i, i)\}$, i.e. a single vertex with a loop. $Q_i = \{0, 1\}$, $A_i = \{a\}$, and

$$T_i(0, a) = 1, \quad T_i(1, a) = 1$$

$$T_{(i,i)}(0, a) = \emptyset \quad T_{(i,i)}(1, a) = a$$

In this case every $\mathbf{x} \in M$ is an amplifier, because if a single message is input at state 1, then i repeatedly transitions from state 1 to 1 as the message is processed, and in doing so sends a message a to itself.

Definition. A proper state $\mathbf{x} \cdot \mathbf{q}$ is a *strong amplifier* if $\mathbf{x} \neq \mathbf{0}$ and $\mathbf{x} \triangleright \mathbf{q} = \mathbf{y} \cdot \mathbf{q}$ for some $\mathbf{y} \geq \mathbf{x}$.

In words, a strong amplifier is a collection of messages \mathbf{x} such that for some initial state q , after processing all messages once, the network has returned to state q with at least as many messages of each type as before.

Example. In a sandpile network on an undirected graph with no sink, a strong amplifier is the configuration $\mathbf{x} = (d_v)_{v \in V}$ where each vertex has the same number of messages as its degree. For any initial state q , processing all messages once causes each vertex to topple once, so that each vertex v receives one message from each of its d_v neighbors. Hence $\mathbf{x} \triangleright q = \mathbf{x} \cdot \mathbf{q}$.

Lemma 6.1. *The following are equivalent for a finite abelian network \mathcal{N} .*

- (1) \mathcal{N} has an amplifier.
- (2) \mathcal{N} has a strong amplifier.
- (3) \mathcal{N} fails to halt on some input.

Proof. We will prove (1) \Rightarrow (3) \Rightarrow (2) \Rightarrow (1). Supposing that \mathcal{N} has an amplifier $\mathbf{x}.\mathbf{q}$, there is a sequence of legal moves a_1, \dots, a_r starting from $\mathbf{x}.\mathbf{q}$ and ending with $\mathbf{y}.\mathbf{q}$ for some $\mathbf{y} \geq \mathbf{x}$. For each $n \geq 0$ the word $w = a_1 \cdots a_r$ is legal starting from $(\mathbf{x} + n(\mathbf{y} - \mathbf{x})).\mathbf{q}$ and produces $(\mathbf{x} + (n+1)(\mathbf{y} - \mathbf{x})).\mathbf{q}$. Hence w^n is a legal word starting from $\mathbf{x}.\mathbf{q}$ for all $n \geq 0$. By Lemma 5.2, since there exist arbitrarily long legal words, \mathcal{N} does not halt on input $\mathbf{x}.\mathbf{q}$, which shows (1) \Rightarrow (3).

Conversely, suppose that \mathcal{N} does not halt on some input $\mathbf{y}_0.\mathbf{q}_0$. For $n \geq 1$ let

$$\mathbf{y}_n.\mathbf{q}_n = \mathbf{y}_{n-1} \triangleright \mathbf{q}_{n-1}.$$

Since \mathcal{N} does not halt on input $\mathbf{y}_0.\mathbf{q}_0$, there does not exist a finite complete word, so $\mathbf{y}_n \neq \mathbf{0}$ for all $n \geq 0$. Since the total state space Q is finite, there exists $\mathbf{q} \in Q$ such that $\mathbf{q}_n = \mathbf{q}$ for infinitely many n . By Lemma 4.5, there exist indices $j < k$ such that $\mathbf{q}_j = \mathbf{q}_k = \mathbf{q}$ and $\mathbf{y}_j \leq \mathbf{y}_k$. We claim that $\mathbf{x} := \mathbf{y}_j + \cdots + \mathbf{y}_{k-1}$ is a strong amplifier. Indeed, we have

$$\begin{aligned} \mathbf{x} \triangleright \mathbf{q} &= (\mathbf{y}_{k-1} + \cdots + \mathbf{y}_{j+1}) \triangleright (\mathbf{y}_j \triangleright \mathbf{q}_j) \\ &= (\mathbf{y}_{k-1} + \cdots + \mathbf{y}_{j+1}) \triangleright (\mathbf{y}_{j+1}.\mathbf{q}_{j+1}) \\ &= (\mathbf{y}_{k-1} + \cdots + \mathbf{y}_{j+2}) \triangleright (\mathbf{y}_{j+1} \triangleright (\mathbf{y}_{j+1}.\mathbf{q}_{j+1})) \\ &= (\mathbf{y}_{k-1} + \cdots + \mathbf{y}_{j+2}) \triangleright ((\mathbf{y}_{j+2} + \mathbf{y}_{j+1}).\mathbf{q}_{j+2}) \\ &= \dots \\ &= (\mathbf{y}_k + \cdots + \mathbf{y}_{j+1}).\mathbf{q}_k. \end{aligned}$$

Since $\mathbf{x} \leq \mathbf{y}_k + \cdots + \mathbf{y}_{j+1}$ and $\mathbf{q}_k = \mathbf{q}$, we conclude that \mathbf{x} is a strong amplifier, which shows (3) \Rightarrow (2).

A strong amplifier is trivially an amplifier, which shows (2) \Rightarrow (1). \square

If \mathcal{N} halts on all inputs, then there is another action of \mathbb{N}^A on the total state space Q by the rule ‘‘process all messages until halting.’’ We will denote this action by $\triangleright\triangleright$. Formally, given $\mathbf{x} \in \mathbb{N}^A$ and $\mathbf{q} \in Q$, set $\mathbf{x}_0 = \mathbf{x}$, $\mathbf{q}_0 = \mathbf{q}$ and

$$\mathbf{x}_n.\mathbf{q}_n = \mathbf{x}_{n-1} \triangleright \mathbf{q}_{n-1}$$

for $n \geq 0$. Since \mathcal{N} halts on all inputs, there exists N such that $\mathbf{x}_n = \mathbf{0}$ for all $n \geq N$. We define

$$\mathbf{x} \triangleright\triangleright \mathbf{q} := \mathbf{q}_N.$$

We extend $\triangleright\triangleright$ to an action of \mathbb{N}^A on $\mathbb{N}^A \times Q$ by defining

$$\mathbf{x} \triangleright\triangleright (\mathbf{y}.\mathbf{q}) := (\mathbf{x} + \mathbf{y}) \triangleright\triangleright \mathbf{q}.$$

Let \mathcal{P} be an abelian finite automaton with state space Q and alphabet A . That is, for each $a \in A$ we have a transition map $t_a : Q \rightarrow Q$, and for all $a, b \in A$ we have $t_a \circ t_b = t_b \circ t_a$. The *transition monoid* of \mathcal{P} is the submonoid $M \subset \text{End}(Q)$ generated by $\{t_a\}_{a \in A}$, where $\text{End}(Q)$ denotes the

monoid of all set maps $Q \rightarrow Q$. We say that \mathcal{P} is *irreducible* if the monoid action $M \times Q \rightarrow Q$ is irreducible.

For an abelian network $\mathcal{N} = (\mathcal{P}_v)_{v \in V}$, each processor \mathcal{P}_v is an abelian finite automaton, so it has a transition monoid $M_v \subset \text{End}(Q_v)$. We call M_v the *local monoid* at vertex v . If \mathcal{N} halts on all inputs, then we can view \mathcal{N} itself as an abelian finite automaton with state space $Q = \prod_{v \in V} Q_v$. The *global monoid* $M \subset \text{End}(Q)$ is generated by the maps $t_a(q) = 1_a \triangleright q$ for $a \in A$. We say that \mathcal{N} is irreducible if the corresponding action $M \times Q \rightarrow Q$ is irreducible (§4). Next we prove a local-to-global principle for irreducibility.

Lemma 6.2. *Let $\mathcal{N} = \{\mathcal{P}_v\}_{v \in V}$ be an abelian network that halts on all inputs. If each processor \mathcal{P}_v is irreducible, then \mathcal{N} is irreducible.*

Proof. Let $\mathbf{q}, \mathbf{q}' \in Q = \prod_{v \in V} Q_v$. For each $v \in V$, by Lemma 4.1 there exist $m_v, m'_v \in M_v$ such that $m_v \mathbf{q}_v = m'_v \mathbf{q}'_v$. Let

$$\mathbf{m} \triangleright \mathbf{q} = \mathbf{n} \cdot \mathbf{r}, \quad \mathbf{m}' \triangleright \mathbf{q}' = \mathbf{n}' \cdot \mathbf{r}'$$

where $\mathbf{r} = \mathbf{r}'$ since $\mathbf{r}_v = m_v \mathbf{q}_v = m'_v \mathbf{q}'_v = \mathbf{r}'_v$ for all $v \in V$. Then

$$\begin{aligned} (\mathbf{n}' \mathbf{m}) \triangleright \mathbf{q} &= \mathbf{n}' \triangleright (\mathbf{m} \triangleright \mathbf{q}) \\ &= \mathbf{n}' \triangleright (\mathbf{n} \cdot \mathbf{r}) \\ &= (\mathbf{n}' \mathbf{n}) \triangleright \mathbf{r} \\ &= \mathbf{n} \triangleright (\mathbf{n}' \cdot \mathbf{r}) \\ &= \mathbf{n} \triangleright (\mathbf{m}' \triangleright \mathbf{q}') \\ &= (\mathbf{n} \mathbf{m}') \triangleright \mathbf{q}'. \end{aligned}$$

Hence M acts irreducibly on Q , which means that \mathcal{N} is irreducible. \square

Definition. A state $\mathbf{q} \in Q$ is *locally recurrent* if $\mathbf{q}_v \in e_v Q_v$ for all $v \in V$. (Equivalently, $\mathbf{q}_v = e_v \mathbf{q}_v$ for all $v \in V$.)

Recall from Lemma 4.3 that every $m \in M_v$ acts invertibly on $e_v Q_v$. Thus for each $a \in A_v$ the map $q_v \mapsto a q_v$ is a permutation of $e_v Q_v$, so we have a group action

$$\mathbb{Z}^{A_v} \times e_v Q_v \rightarrow e_v Q_v.$$

Let K_v be the kernel of this action, and let $K = \prod_{v \in V} K_v \subset \mathbb{Z}^A$. Thus

$$K = \{\mathbf{x} \in \mathbb{Z}^A \mid \mathbf{x}_v q_v = q_v \text{ for all locally recurrent } q \in Q \text{ and all } v \in V\}. \quad (2)$$

Lemma 6.3. *If \mathcal{N} is a finite abelian network, then K is a subgroup of finite index in \mathbb{Z}^A .*

Proof. Each K_v is a subgroup of \mathbb{Z}^{A_v} . Since $e_v Q_v$ is a finite set, for any $\mathbf{x} \in \mathbb{Z}^{A_v}$ we have $n\mathbf{x} \in K_v$ for some $n \geq 1$, so K_v has finite index in \mathbb{Z}^{A_v} . Since V is finite, $K = \prod K_v$ has finite index in \mathbb{Z}^A . \square

Fix a locally recurrent state $q \in Q$. For any $\mathbf{k} \in K \cap \mathbb{N}^A$ we have

$$\mathbf{k} \triangleright q = P_q(\mathbf{k}) \cdot \mathbf{q}$$

for some vector $P(\mathbf{k}) = P_q(\mathbf{k}) \in \mathbb{N}^A$. Next we will show that

$$P_q : K \cap \mathbb{N}^A \rightarrow \mathbb{N}^A$$

extends to a group homomorphism $K \rightarrow \mathbb{Z}^A$, which does not depend on the choice of locally recurrent q .

We say that \mathcal{N} is *locally irreducible* if for all $v \in V$ the monoid action $M_v \times Q_v \rightarrow Q_v$ is irreducible.

Lemma 6.4. *Let \mathcal{N} be a finite abelian network. Then P extends to a group homomorphism $K \rightarrow \mathbb{Z}^A$. Moreover, if \mathcal{N} is locally irreducible, then P does not depend on the choice of locally recurrent q .*

Proof. Let $\mathbf{k}_1, \mathbf{k}_2 \in K \cap \mathbb{N}^A$. Since

$$(\mathbf{k}_1 + \mathbf{k}_2) \triangleright q = \mathbf{k}_1 \triangleright (P(\mathbf{k}_2) \cdot \mathbf{q}) = (P(\mathbf{k}_1) + P(\mathbf{k}_2)) \cdot \mathbf{q}$$

we have

$$P(\mathbf{k}_1 + \mathbf{k}_2) = P(\mathbf{k}_1) + P(\mathbf{k}_2). \quad (3)$$

By Lemma 6.3, K contains a vector with all coordinates strictly positive, which implies that K is generated as a group by $K \cap \mathbb{N}^A$, so every $\mathbf{x} \in K$ can be written as $\mathbf{k}_1 - \mathbf{k}_2$ for $\mathbf{k}_1, \mathbf{k}_2 \in K \cap \mathbb{N}^A$. Define $P(\mathbf{x}) = P(\mathbf{k}_1) - P(\mathbf{k}_2)$. Equation (3) now implies that this extension is well defined and a group homomorphism.

Let $q_1, q_2 \in Q$ be locally recurrent. If \mathcal{N} is locally irreducible, then by Lemma 4.4 each group action $e_v M_v \times e_v Q_v \rightarrow e_v Q_v$ is transitive, so there exists $\mathbf{x} \in \mathbb{N}^A$ such that $\mathbf{x}q_1 = q_2$. Let \mathbf{z} be such that

$$\mathbf{x} \triangleright q_1 = \mathbf{z} \cdot \mathbf{q}_2.$$

Fix $\mathbf{k} \in K \cap \mathbb{N}^A$, and let $\mathbf{y}_i = P_{q_i}(\mathbf{k})$ for $i = 1, 2$. Then

$$\mathbf{x} \triangleright (\mathbf{k} \triangleright q_1) = \mathbf{x} \triangleright (\mathbf{y}_1 \cdot \mathbf{q}_1) = (\mathbf{y}_1 + \mathbf{z}) \cdot \mathbf{q}_2$$

while

$$\mathbf{k} \triangleright (\mathbf{x} \triangleright q_1) = \mathbf{k} \triangleright (\mathbf{z} \cdot \mathbf{q}_2) = (\mathbf{y}_2 + \mathbf{z}) \cdot \mathbf{q}_2.$$

By the local abelian property,

$$\mathbf{k} \triangleright (\mathbf{x} \triangleright q) = (\mathbf{k} + \mathbf{x}) \triangleright q = \mathbf{x} \triangleright (\mathbf{k} \triangleright q)$$

hence $\mathbf{y}_1 + \mathbf{z} = \mathbf{y}_2 + \mathbf{z}$, and hence $\mathbf{y}_1 = \mathbf{y}_2$. This shows that $P_q(\mathbf{k})$ does not depend on q . \square

By tensoring $P : K \rightarrow \mathbb{Z}^A$ with \mathbb{Q} , we obtain a linear map $P : \mathbb{Q}^A \rightarrow \mathbb{Q}^A$. To be more explicit, for any $\mathbf{x} \in \mathbb{Q}^A$, by Lemma 6.3 there is an integer $n \geq 1$ such that $n\mathbf{x} \in K$, and we define

$$P(\mathbf{x}) := \frac{1}{n}P(n\mathbf{x}).$$

So far we have defined P_q only for locally recurrent q . We extend the definition to all states $q \in Q$ by setting $P_q := P_{\hat{q}}$, where $\hat{q} = (e_v q_v)_{v \in V}$.

Definition. The *production matrix* of a finite abelian network \mathcal{N} with initial state q is the matrix of the linear map $P_q : \mathbb{Q}^A \rightarrow \mathbb{Q}^A$.

The (a, b) entry p_{ab} of the production matrix says “on average” how many messages of type a are created when we process a message of type b : specifically, if $n\delta_b \in K$, then p_{ab} equals $1/n$ times the number of a ’s created by processing n messages of type b . The term “production matrix” was chosen to evoke [DFR05]. Indeed the succession rules studied in that paper can be modeled by an abelian network whose underlying graph is a single vertex with a loop.

By Lemma 6.4, if \mathcal{N} is locally irreducible then the production matrix does not depend on the initial locally recurrent state q . The state space of an arbitrary finite abelian network \mathcal{N} can be decomposed into locally irreducible components, each with its own production matrix.

Theorem 6.5. *A finite abelian network \mathcal{N} halts on every input to initial state q if and only if every eigenvalue of the production matrix P_q has absolute value strictly less than 1.*

Proof. The locally irreducible component \mathcal{N}_q contains all states accessible from q , so we may assume without loss of generality that \mathcal{N} is locally irreducible. Let $P = P_q$, and let \mathbf{x} and λ be the Perron-Frobenius eigenvector and eigenvalue of P (see Lemma ??). By Lemma 6.1 it suffices to show that (1) if $\lambda \geq 1$, then \mathcal{N} has an amplifier; and (2) if \mathcal{N} has a strong amplifier, then $\lambda \geq 1$.

If $\lambda > 1$, then there is a vector $\mathbf{y} \in \mathbb{Q}^A$ such that $\mathbf{x} \leq \mathbf{y} \leq \lambda\mathbf{x}$. Then $P\mathbf{y} \geq P\mathbf{x} = \lambda\mathbf{x} \geq \mathbf{y}$. Letting $n \in \mathbb{N}$ be large enough so that $n\mathbf{y} \in K$, we have $n\mathbf{y} \triangleright q = P(n\mathbf{y}) \cdot \mathbf{q}$, so $n\mathbf{y} \cdot \mathbf{q}$ is an amplifier.

If $\lambda = 1$, then \mathbf{x} can be taken to have integer entries. Letting $n \in \mathbb{N}$ be large enough so that $n\mathbf{x} \in K$, we have $n\mathbf{x} \triangleright q = n\mathbf{x} \cdot \mathbf{q}$, so $n\mathbf{x} \cdot \mathbf{q}$ is an amplifier.

If $\lambda < 1$, then for all $\mathbf{y} \in \mathbb{Q}^A$ we have $P^n\mathbf{y} \downarrow \mathbf{0}$ as $n \rightarrow \infty$. Suppose that \mathbf{y} is a strong amplifier. Then there exists $q' \in Q$ such that $\mathbf{y} \triangleright q' = \mathbf{z} \cdot \mathbf{q}'$ for some $\mathbf{z} \geq \mathbf{y}$. Since \mathcal{N} is locally irreducible, q' is locally recurrent, so $P(\mathbf{y}) = \mathbf{z}$. Since all entries of P are nonnegative it follows that $P^n(\mathbf{y}) \geq \mathbf{y}$ for all $n \geq 1$, contradicting $P^n(\mathbf{y}) \downarrow \mathbf{0}$. \square

Corollary 6.6. *If a finite abelian network \mathcal{N} is locally irreducible and halts on all inputs to one initial state $q \in Q$, then \mathcal{N} halts on all inputs to any initial state.*

7. LAPLACIAN MATRIX; SANDPILIZATION

For each letter $a \in A$, let d_a be the smallest positive integer such that $d_a a \in K$. Let D be the $A \times A$ diagonal matrix with diagonal entries d_a .

Definition. The *Laplacian* of a finite abelian network \mathcal{N} is the $A \times A$ matrix

$$L = (I - P)D$$

where I is the $A \times A$ identity matrix, and P is the production matrix of \mathcal{N} .

Note that L has integer entries. The following is immediate from Theorem 6.5.

Corollary 7.1. *A finite abelian network halts on all inputs if and only if its Laplacian is positive definite.*

If $L_{aa} \leq 0$ for some $a \in A$, then $d_a a$ is an amplifier, so \mathcal{N} does not halt on all inputs. Otherwise, the diagonal entries of L are positive and the off-diagonal entries are nonpositive. Positive definite matrices with this sign pattern are sometimes called “ M -matrices” and have many equivalent characterizations:

Lemma 7.2. ([?]) *Let L be a square matrix such that $L_{ii} > 0$ for all i , and $L_{ij} \leq 0$ for all $i \neq j$. The following are equivalent:*

- (1) *The principal minors of L are positive.*
- (2) *There exists a vector $x > 0$ such that $Lx > 0$.*
- (3) *There exists a vector $y > 0$ such that $L^T y > 0$.*
- (4) *L is invertible, and all entries of L^{-1} are nonnegative.*

For any locally finite abelian network \mathcal{N} we associate a corresponding toppling network $\mathcal{S}(\mathcal{N})$ called its *sandpilization*. The underlying graph of $\mathcal{S}(\mathcal{N})$ has vertex set A and directed edge set $\{(a, b) \mid p_{ba} > 0\}$. The threshold of vertex a is d_a , and toppling vertex a sends $-L_{ba}$ messages to each out-neighbor b of a . Formally, the transition and message passing functions of $\mathcal{S}(\mathcal{N})$ are given by

$$T_a(q, a) = q + 1 \pmod{d_a}$$

$$T_{(a,b)}(q, a) = \begin{cases} b^{-L_{ba}}, & q = 0 \\ \emptyset, & q > 0. \end{cases}$$

For example, the sandpilization of a rotor network on directed graph G is the sandpile network on G .

By construction, \mathcal{N} and $\mathcal{S}(\mathcal{N})$ have the same Laplacian.

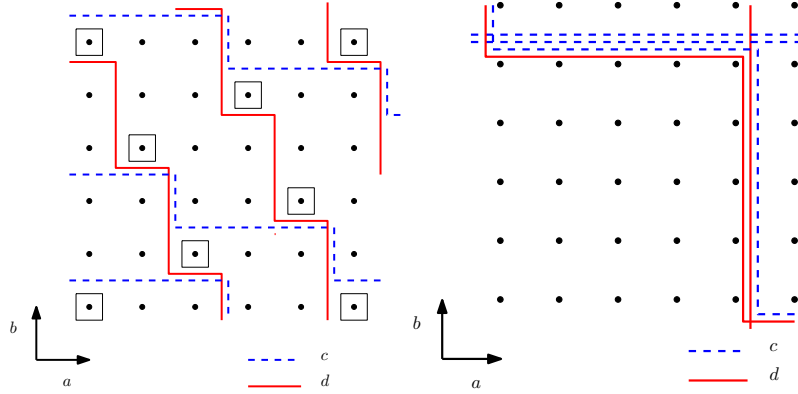


FIGURE 6. Left: State diagram of an abelian processor with 5 states and input alphabet $\{a, b\}$. Right: State diagram of its sandpilation.

Example. Consider the first example of §3. For the sandpilation of this network, there are three vertices, i_a, i_b, j , with thresholds 3, 2, 3, respectively. i_a, i_b, j output 2, 1, 1 messages to j, j, i_a , respectively when they topple. The laplacian of the sandpilation is:

$$L = \begin{pmatrix} 3 & 0 & -2 \\ 0 & 2 & -1 \\ -1 & 0 & 3 \end{pmatrix}$$

Since \mathcal{N} and $\mathcal{S}(\mathcal{N})$ have the same Laplacian, the following is immediate from Corollary 7.1.

Corollary 7.3. *Let \mathcal{N} be a finite locally irreducible abelian network. Then \mathcal{N} halts on every input if and only if $\mathcal{S}(\mathcal{N})$ halts on every input.*

Example. Let \mathcal{N} be a toppling network with three vertices, a, b, c , with thresholds 3, 4, 5. For the messages passed we have:

$$\begin{aligned} a \text{ topples} &\rightarrow b, c \text{ receive } 2, 2 \text{ chips.} \\ b \text{ topples} &\rightarrow a, c \text{ receive } 1, 2 \text{ chips.} \\ c \text{ topples} &\rightarrow a, b \text{ receive } 0, 2 \text{ chips.} \end{aligned}$$

The Laplacian of this network is

$$L = \begin{pmatrix} 3 & -2 & -2 \\ -1 & 4 & -2 \\ 0 & -2 & 5 \end{pmatrix}.$$

Since L is positive definite, \mathcal{N} halts on every input.

8. THE CRITICAL GROUP OF AN ABELIAN NETWORK

In this section we define and study an important algebraic invariant of an abelian network, its critical group. The critical group governs the “long-term” behavior of the network, i.e., its behavior on sufficiently large inputs. The term critical group is due to Biggs [Big99], but the idea goes back to Lorenzini [Lor89, Lor91] and Dhar [Dha90]. These authors all considered, in various guises, the group $\text{Crit } \mathcal{N}$ associated to a sandpile network \mathcal{N} on a graph G with one vertex acting as a sink. Our construction is more general in that it applies to a larger class of abelian networks.

The terms *critical group* and *sandpile group* are used more or less interchangeably in the mathematical literature. In setting of abelian networks, we can make a distinction between them. If \mathcal{N} is a finite irreducible abelian network that halts on all inputs, then it has an associated critical group $\text{Crit } \mathcal{N}$. The sandpile group associated to a directed graph G with marked vertex s is the group $\text{Crit } \mathcal{N}$, where \mathcal{N} is the sandpile network on G with sink at s .

Throughout this section, we take \mathcal{N} to be a finite irreducible abelian network that halts on all inputs. The *transition monoid* of \mathcal{N} is the submonoid $M \subset \text{End } Q$ generated by the input maps $q \mapsto a \triangleright \triangleright q$ for $a \in A$. Since M is a finite commutative monoid, it contains an abelian group as its minimal ideal (see §4).

Definition. The *critical group* $\text{Crit } \mathcal{N}$ is the minimal ideal eM of M .

Definition. A state $x \in Q$ is *recurrent* if the equivalent conditions of Lemma 4.2 hold. Denote by $\text{Rec } \mathcal{N}$ the set of recurrent states of \mathcal{N} .

For example, if \mathcal{N} is a rotor network on a directed graph G with a sink vertex s , then the recurrent states of \mathcal{N} can be identified with spanning trees of G oriented toward s . The following theorem generalizes [HLMPPW08], where it was shown that the sandpile group of G with sink at s acts freely and transitively on oriented spanning trees.

Theorem 8.1. *Let \mathcal{N} be a finite irreducible abelian network that halts on all inputs. The action of the critical group on recurrent states*

$$\text{Crit } \mathcal{N} \times \text{Rec } \mathcal{N} \rightarrow \text{Rec } \mathcal{N}$$

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

Proof. Let M be the transition monoid of \mathcal{N} , and let e be the identity element of the minimal ideal of M . Then $\text{Crit } \mathcal{N} = eM$ and $\text{Rec } \mathcal{N} = eQ$. The monoid action $M \times Q \rightarrow Q$ is faithful by definition and locally transitive by Lemma 6.2. Hence the group action $eM \times eQ \rightarrow eQ$ is free and transitive by Lemma 4.4. \square

Example. We formalize one way in which $\text{Crit } \mathcal{N}$ and its action on $\text{Rec } \mathcal{N}$ govern the “long term behavior” of \mathcal{N} . Let μ be a probability distribution on A such that $\mu(\{a\}) > 0$ for all $a \in A$. Consider the Markov chain $(q_n)_{n \geq 0}$ on states of \mathcal{N} where the initial state q_0 can be arbitrary, and subsequent states are defined by

$$q_{n+1} = a_n \triangleright \triangleright q_n, \quad n \geq 0$$

where the inputs $a_n \in A$ for $n \geq 0$ are drawn independently at random with distribution μ .

Corollary 8.2. *For any input distribution μ , the stationary distribution of the Markov chain $(q_n)_{n \geq 0}$ is uniform on $\text{Rec } \mathcal{N}$.*

Proof. Fix $q_0 \in \text{Rec } \mathcal{N}$, and let g be uniform random element of $\text{Crit } \mathcal{N}$. By Theorem 8.1, $g \triangleright \triangleright q_0$ is a uniform random element of $\text{Rec } \mathcal{N}$. If $a \in A$ is independent of g , then ag is also a uniform random element of $\text{Crit } \mathcal{N}$; hence if q_n is uniform on $\text{Rec } \mathcal{N}$, then q_{n+1} is again uniform on $\text{Rec } \mathcal{N}$. \square

Next we turn to the problem of describing the critical group by generators and relations. Consider the group homomorphism

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

defined on generators by $a \mapsto et_a$ for each $a \in A$. This map is surjective by definition, since $\text{Crit } \mathcal{N} = eM$ and M is generated by $\{t_a\}_{a \in A}$. To identify its kernel, recall the *production map*

$$P : K \rightarrow \mathbb{Z}^A$$

defined in §6, where K is the kernel of the local action (2). Write I for the inclusion $K \hookrightarrow \mathbb{Z}^A$.

Theorem 8.3. *The natural map $\phi : \mathbb{Z}^A \rightarrow \text{Crit } \mathcal{N}$ induces an isomorphism*

$$\text{Crit } \mathcal{N} \simeq \mathbb{Z}^A / (I - P)K.$$

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

$$\mathbf{k} \triangleright q = P(\mathbf{k}) \cdot \mathbf{q}$$

so that $\mathbf{k} \triangleright \triangleright q = P(\mathbf{k}) \triangleright \triangleright q$. Hence $\phi(\mathbf{k})$ and $\phi(P(\mathbf{k}))$ have the same action on q . By Theorem 8.1 the action of $\text{Crit } \mathcal{N}$ on $\text{Rec } \mathcal{N}$ is free, so we conclude $\phi(\mathbf{k}) = \phi(P(\mathbf{k}))$. This shows that $(I - P)(\mathbf{k}) \in \ker \phi$ for all $\mathbf{k} \in K \cap \mathbb{N}^A$. Since K is generated as a group by $K \cap \mathbb{N}^A$, we obtain $(I - P)K \subset \ker \phi$.

To show the reverse inclusion, given $\mathbf{x} \in \ker \phi$ we have $\mathbf{x} \triangleright \triangleright q = e \triangleright \triangleright q = q$ for all $q \in \text{Rec } \mathcal{N}$. Let $\mathbf{u} = (u_a)_{a \in A}$, where u_a be the number of times letter a is processed in reducing $\mathbf{x} \cdot \mathbf{q}$ to q . We will show that $\mathbf{u} \in K$ and $\mathbf{x} = \mathbf{u} - P(\mathbf{u})$. Write t_v for the natural map $\mathbb{N}^{A_v} \rightarrow M_v$. Since $\mathbf{u} \triangleright q = q$, we have $t_v(\mathbf{u}_v)q_v = q_v$ for all $v \in V$. By Lemma 4.4 the action of $e_v M_v$

on $e_v Q_v$ is free, so $e_v t_v(\mathbf{u}_v) = e_v$. Hence for any $r = e_v r' \in e_v Q_v$ we have $t_v(\mathbf{u}_v)r = t_v(\mathbf{u}_v)e_v r' = e_v r' = r$. This shows that $\mathbf{u} \in K$. Moreover

$$\mathbf{u} \cdot \mathbf{q} = \mathbf{u} \triangleright (\mathbf{x} \cdot \mathbf{q}) = (P(\mathbf{u}) + \mathbf{x}) \cdot \mathbf{q}$$

hence $\mathbf{x} = \mathbf{u} - P(\mathbf{u})$. \square

Call \mathcal{N} *rectangular* if $K = \prod_{a \in A} d_a \mathbb{Z}$ (that is, K is a rectangular sublattice of \mathbb{Z}^A).

Corollary 8.4. *The natural map $\mathbb{Z}^A \rightarrow \text{Crit } \mathcal{N}$ induces a surjective group homomorphism*

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

If \mathcal{N} is rectangular, then ϕ is an isomorphism.

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

Note that any unary network (and in particular any toppling network) is rectangular.

We remark that Corollary 8.4 can also be proved by defining an equivalence relation on Q and showing that each equivalence class contains a unique recurrent state.

Corollary 8.5. *$\text{Crit } \mathcal{S}(\mathcal{N}) \simeq \mathbb{Z}^A / L\mathbb{Z}^A$ and $\text{Crit } \mathcal{S}(\mathcal{N}) \twoheadrightarrow \text{Crit } \mathcal{N}$.*

Finally we turn to the problem of counting recurrent states. Let

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

be the index of $D\mathbb{Z}^A$ in 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 $D_v \mathbb{Z}^{A_v} := \prod_{a \in A_v} d_a \mathbb{Z}$. Note $\iota = 1$ if and only if \mathcal{N} is rectangular.

Corollary 8.6.

$$\#\text{Rec } \mathcal{N} = \#\text{Crit } \mathcal{N} = \frac{\det L}{\iota}.$$

Proof. The first equality follows from Theorem 8.1. For the second, we have by Theorem 8.3

$$\begin{aligned} \#\text{Crit } \mathcal{N} &= [\mathbb{Z}^A : (I - P)K] = \frac{[\mathbb{Z}^A : L\mathbb{Z}^A]}{[(I - P)K : L\mathbb{Z}^A]} \\ &= \frac{|\det L|}{[(I - P)K : (I - P)D\mathbb{Z}^A]}. \end{aligned}$$

Since \mathcal{N} halts on all inputs, $\det L > 0$ by Corollary 7.1. Likewise, $I - P$ has full rank by Theorem 6.5, so the denominator equals $[K : D\mathbb{Z}^A]$, which is ι . \square

Example. This example addresses non-rectangular networks. Let \mathcal{N} be a network with 2 vertices, i and j , with $Q_i = \{0, 1\}$, $A_i = \{a, b\}$,

$$T_i(n, x) = n + 1 \pmod{2}, \quad x \in A_i$$

and for some vertex j , with $c \in A_j$,

$$T_{(i,j)}(n, a) = c \quad n = 0$$

$$T_{(i,j)}(n, a) = 2c \quad n = 1$$

$$T_{(i,j)}(n, b) = \emptyset \quad n = 0$$

$$T_{(i,j)}(n, b) = c \quad n = 1$$

The vertex j has a single state, and outputs no messages.

When \mathcal{N} is locally faithful, if we cycle through the states for one input, the same message is output. However, in this case there are 3 ways to cycle through the same set of states, namely aa, ab, ba , and each outputs a different number of messages of type c .

Notice the Laplacian of this network is

$$\begin{pmatrix} 2 & 0 & -3 \\ 0 & 2 & -1 \\ 0 & 0 & 1 \end{pmatrix}$$

So $\text{Crit } \mathcal{S}(\mathcal{N}) = \mathbb{Z}_2 \times \mathbb{Z}_2$, and $\text{Crit } \mathcal{N} = \mathbb{Z}_2$.

We now consider a slight variation \mathcal{N}' of \mathcal{N} . It consists of two vertices, with j as before, but i sends slightly different messages:

$$T_i(n, x) = n + 1 \pmod{2}, \quad x \in A_i$$

and for some vertex j , with $c \in A_j$,

$$T_{(i,j)}(n, a) = c \quad n = 0$$

$$T_{(i,j)}(n, a) = b, c \quad n = 1$$

$$T_{(i,j)}(n, b) = \emptyset \quad n = 0$$

$$T_{(i,j)}(n, b) = b \quad n = 1$$

The Laplacian of \mathcal{N}' is

$$\begin{pmatrix} 2 & -1 & -2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

So $\text{Crit } \mathcal{S}(\mathcal{N}) = \text{Crit } \mathcal{N} = \mathbb{Z}_2$. In particular, the isomorphism of theorem ?? is not if and only if.

Let \mathcal{N} be an arbitrary abelian network. Since most theorems presented in this paper require the hypothesis of local irreducibility, we briefly discuss how to decompose \mathcal{N} into its locally irreducible components.

We begin by decomposing the states of each vertex v into its irreducible components. Call them $Q_v^1, \dots, Q_v^{n_v}$. To obtain locally irreducible components of \mathcal{N} , we simply choose one irreducible component of the states of each vertex. We then create a network with the same vertex and edge sets, such that the states of each vertex v is one of the Q_v^i . Message passing functions are given by restricting those of \mathcal{N} . Notice the resulting network is indeed irreducible by 6.2.

This is most simply seen in the case of the Dartois-Rossin Height Arrow Model (HAM). A vertex starting in state (q, c) may only access states $(q + n\tau_v, c')$, for $n \in \mathbb{N}$. If $\tau_v | d_v$ and $\tau_v \neq 1$, then Q_v is not irreducible.

For each vertex, choose an irreducible component of Q_v , and form an irreducible network from these, as above. Notice that for any choice of irreducible components, the p_{ab} of the production matrix are the same, so each irreducible component in a HAM Network has the same critical group. Proposition 3.9 of [DR04] counts the recurrent configurations of a HAM Network (i.e. configurations which are recurrent in one of the irreducible components), which counts the sum of the orders of the critical groups over all irreducible components. There are

$$\prod_v \gcd(\tau_v, d_v) |\text{Crit } \mathcal{N}'|$$

recurrent configurations, where \mathcal{N}' is one of the irreducible components of a HAM network \mathcal{N} . This is exactly the number we obtain from the decomposition process above.

ACKNOWLEDGMENTS

The authors thank Olivier Bernardi, Anne Fey, Michael Hochman, Benjamin Iriarte, James Propp and Leonardo Rolla for helpful discussions.

This research was supported by an NSF postdoctoral fellowship and NSF DMS-1105960, and by the UROP and SPUR programs at MIT.

REFERENCES

- [Ash87] Jonathan Ashley, On the Perron-Frobenius eigenvector for nonnegative integral matrices whose largest eigenvalue is integral, *Lin. Alg. Appl.* **94**:103–108, 1987.
- [BT10] László Babái and Evelin Toumpakari, A structure theory of the sandpile monoid for directed graphs, 2010. <http://people.cs.uchicago.edu/~laci/REU10/evelin.pdf>
- [BTW87] Per Bak, Chao Tang and Kurt Wiesenfeld, Self-organized criticality: an explanation of the $1/f$ noise, *Phys. Rev. Lett.* **59**(4):381–384, 1987.

- [BW03] Itai Benjamini and David B. Wilson, Excited random walk, *Elect. Comm. Probab.* **8**:86–92, 2003.
- [Big99] Norman L. Biggs, Chip-firing and the critical group of a graph, *J. Algebraic Combin.* **9**(1):25–45, 1999.
- [BLS91] Anders Björner, László Lovász and Peter Shor, Chip-firing games on graphs, *European J. Combin.* **12**(4):283–291, 1991.
- [CS06] Joshua Cooper and Joel Spencer, Simulating a random walk with constant error, *Combin. Probab. Comput.* **15**:815–822, 2006.
- [DR04] Arnoud Dartois and Dominique Rossin, Height-arrow model, *Formal Power Series and Algebraic Combinatorics*, 2004.
- [Dha90] Deepak Dhar, Self-organized critical state of sandpile automaton models, *Phys. Rev. Lett.* **64**:1613–1616, 1990.
- [Dha99] Deepak Dhar, Studying self-organized criticality with exactly solved models, 1999. [arXiv:cond-mat/9909009](#)
- [Dha06] Deepak Dhar, Theoretical studies of self-organized criticality, *Physica A* **369**:29–70, 2006.
- [DSC09] Deepak Dhar, Tridib Sadhu and Samarth Chandra, Pattern formation in growing sandpiles, *Europhysics Lett.* **85**:48002, 2009.
- [DF91] Persi Diaconis and William Fulton, A growth model, a game, an algebra, Lagrange inversion, and characteristic classes, *Rend. Sem. Mat. Univ. Pol. Torino* **49**(1):95–119, 1991.
- [DFR05] Emeric Deutsch, Luca Ferrari and Simone Rinaldi, Production matrices. *Adv. Appl. Math.* **34**(1):101–122, 2005.
- [DRS10] Ronald Dickman, Leonardo T. Rolla and Vladas Sidoravicius, Activated random walkers: facts, conjectures and challenges, *J. Stat. Phys.* **138**(1-3):126–142, 2010.
- [Ent87] A. C. D. van Enter, Proof of straleys argument for bootstrap percolation. *J. Stat. Phys.* **48**(3-4):943–945, 1987.
- [Eri96] Kimmo Eriksson, Chip-firing games on mutating graphs, *SIAM J. Discrete Math.* **9**(1):118–128, 1996.
- [FLP10] Anne Fey, Lionel Levine and Yuval Peres, Growth rates and explosions in sandpiles, *J. Stat. Phys.* **138**:143–159, 2010. [arXiv:0901.3805](#)
- [FMR09] Anne Fey, Ronald Meester, and Frank Redig, Stabilizability and percolation in the infinite volume sandpile model, *Ann. Probab.* **37**(2):654–675, 2009.
- [Fre93] Vidar Frette, Sandpile models with dynamically varying critical slopes, *Phys. Rev. Lett.* **70**:2762–2765, 1993.
- [FL10] Tobias Friedrich and Lionel Levine, Fast simulation of large-scale growth models, 2010. [arXiv:1006.1003](#).
- [GLPZ11] Giuliano Giacaglia, Lionel Levine, James Propp and Linda Zayas-Palmer, Local-to-global principles for rotor walk, 2011. [arXiv:1107.4442](#)
- [Gre51] J. A. Green, On the structure of semigroups, *Ann. of Math.* **54**:163–172, 1951.
- [Hol03] Alexander E. Holroyd, Sharp metastability threshold for two-dimensional bootstrap percolation, *Probab. Theory Related Fields*, **125**(2):195–224, 2003. [arXiv:math/0206132](#)
- [HLMPPW08] Alexander E. Holroyd, Lionel Levine, Karola Mészáros, Yuval Peres, James Propp and David B. Wilson, Chip-firing and rotor-routing on directed graphs, *In and out of equilibrium 2*, 331–364, Progr. Probab. **60**, Birkhäuser, 2008. [arXiv:0801.3306](#)

- [HP10] Alexander E. Holroyd and James G. Propp, Rotor walks and Markov chains, in *Algorithmic Probability and Combinatorics*, American Mathematical Society, 2010. [arXiv:0904.4507](#)
- [HJ90] Roger A. Horn and Charles R. Johnson, *Matrix Analysis*, Cambridge Univ. Press, 1990.
- [KL10] Wouter Kager and Lionel Levine, Rotor-router aggregation on the layered square lattice, *Electr. J. Combin.* **17**:R152, 2010. [arXiv:1003.4017](#)
- [KS05] Harry Kesten and Vladas Sidoravicius, The spread of a rumor or infection in a moving population, *Ann. Probab.* **33**:2402–2462, 2005. [arXiv:math/0312496](#)
- [KS08] Harry Kesten and Vladas Sidoravicius, A shape theorem for the spread of an infection, *Ann. of Math.* **167**:701–766, 2008. [arXiv:math/0312511](#)
- [LL09] Itamar Landau and Lionel Levine, The rotor-router model on regular trees, *J. Combin. Theory A* **116**: 421–433, 2009. [arXiv:0705.1562](#)
- [LBG92] Gregory F. Lawler, Maury Bramson and David Griffeath, Internal diffusion limited aggregation, *Ann. Probab.* **20**(4):2117–2140, 1992.
- [LP09] Lionel Levine and Yuval Peres, Strong spherical asymptotics for rotor-router aggregation and the divisible sandpile, *Potential Anal.* **30**:1–27, 2009. [arXiv:0704.0688](#)
- [Lor89] Dino J. Lorenzini, Arithmetical graphs, *Math. Ann.* **285**(3):481–501, 1989.
- [Lor91] Dino J. Lorenzini, A finite group attached to the Laplacian of a graph, *Discrete Math.* **91**(3):277–282, 1991.
- [Man91] S. S. Manna, Two-state model of self-organized criticality, *J. Phys. A: Math. Gen.* **24**:L363, 1991.
- [Ost03] Srdjan Ostojic, Patterns formed by addition of grains to only one site of an abelian sandpile, *Physica A* **318**:187–199, 2003.
- [PB74] George Poole and Thomas Boullion, A survey on M -matrices, *SIAM Review* **16**(4):419–421, 1974.
- [PDDK96] V. B. Priezzhev, Deepak Dhar, Abhishek Dhar and Supriya Krishnamurthy, Eulerian walkers as a model of self-organised criticality, *Phys. Rev. Lett.* **77**:5079–5082, 1996. [arXiv:cond-mat/9611019](#)
- [Pro04] James Propp, Three lectures on quasirandomness, 2004. Available at <http://faculty.uml.edu/jpropp/berkeley.html>.
- [Pro10] James Propp, Discrete analog computing with rotor-routers. *Chaos* **20**:037110, 2010. [arXiv:1007.2389](#)
- [RS11] Leonardo T. Rolla and Vladas Sidoravicius, Absorbing-state phase transition for driven-dissipative stochastic dynamics on \mathbb{Z} , *Inventiones Math.*, to appear. [arXiv:0908.1152](#)
- [Sch57] Marcel-Paul Schützenberger, \overline{D} représentation des demi-groupes, *C. R. Acad. Sci. Paris* 244:1994–96, 1957.
- [Spe93] Eugene R. Speer, Asymmetric abelian sandpile models. *J. Stat. Phys.* **71**:61–74, 1993.
- [Ste10] Benjamin Steinberg, A theory of transformation monoids: combinatorics and representation theory, *Electr. J. Combin.* **17**:R164, 2010. [arXiv:1004.2982](#)
- [Tse90] Paul Tseng, Distributed computation for linear programming problems satisfying a certain diagonal dominance condition, *Mathematics of Operations Research* **15**(1):33–48, 1990.
- [WLB96] Israel A. Wagner, Michael Lindenbaum and Alfred M. Bruckstein, Smell as a computational resource — a lesson we can learn from the ant, *Proc. ISTCS96* pp. 219–230.

BEN BOND, DEPARTMENT OF MATHEMATICS, MASSACHUSETTS INSTITUTE OF TECHNOLOGY, CAMBRIDGE, MA 02139.

LIONEL LEVINE, DEPARTMENT OF MATHEMATICS, CORNELL UNIVERSITY, ITHACA, NY 14853. <http://math.mit.edu/~levine>