Universiteit Utrecht

Faculteit Bètawetenschappen

Chip-firing games on graphs

Bachelor Thesis

Ruben Vink

Mathematics

Supervisors:

Dr. Valentijn Karemaker
Universiteit Utrecht

Dr. Marieke van der Wegen
Universiteit Utrecht

June 17, 2021

Contents

1 Introduction

Chip-firing games have kept mathematicians busy since the early 1980's. A simple game with simple rules defined on graphs allows for a deceptively easy approach. However, it turns out there is a lot more to it.

All throughout the 1980's many different mathematicians separately came to different definitions of the same idea: the reduced divisor. This reduced divisor allows us to relate seemingly different chip-firing games as the same, and even lets us prove certain games are unwinnable under certain rule sets.

To this end, divisors are put into groups that are generated by the structure of the graph, hinting at the connection to other graph properties such as the Laplacian matrix.

In this thesis we will focus on algorithms to determine the reduced divisors in a way that is efficient, but still intuitive and doable on paper. Some prior knowledge of graph theory and linear algebra is recommended, but most ideas are conveyed informally in addition to the formal mathematical groundwork.

The (informal) ideas in this thesis are combinations of understandings found in papers [1], [2], and part of the book [3], in order to set forth the ideas found in [4] that were also found independently by many other mathematicians as mentioned in the introduction in said paper.

2 Graphs and the chip-firing game

Definition 2.1. A *graph* G is a triplet (V, E, ω) such that V is a set of vertices, and E a set of ordered pairs of vertices such that when $x, y \in V$ and $(x, y) \in E$ there exists a directed connection from x to y. Finally, ω is a function acting on E that assigns a value to every edge.

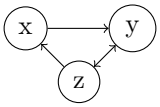


Figure 1: A graphical representation of a graph given by $V = \{x, y, z\}$ and $E = \{(x, y), (y, z), (z, x), (z, y)\}$ with $\omega(e) = 1$ constant for all edges.

For graphs we can define certain properties.

Definition 2.2. A graph G is *undirected* when for every edge $(x, y) \in E$, we have $(y, x) \in E$. Otherwise, the graph is *directed*.

Definition 2.3. A graph $G = (V, E, \omega)$ is *unweighted* when $\omega(e) = 1$ for all $e \in E$. Additionally, when the weight function is left out ($G = (V, E)$), the graph is assumed to be unweighted.

Definition 2.4. A graph G is *connected* when for any pair $x, y \in V$, there exists a series of edges that form a path such that:

$$(x, z_1), (z_1, z_2), ..., (z_{p-1}, z_p), (z_p, y) \in E,$$

where p is the number of other vertices in the path.

Remark 2.5. As a result of the definition of connectedness, we can quickly see that any graph exists of one or more connected sub-graphs. In chip-firing games, unconnected graphs are simply multiple instances of the game played on its connected components, so without loss of generality we restrict ourselves to connected graphs.

Definition 2.6. In an undirected graph we denote the *degree* of a vertex $v \in V$ by $\deg(v)$. The degree of a vertex is the amount of edges incident to the vertex:

$$\deg(v) = |\{w : (v, w) \in E,\ w \in V\}|.$$

Remark 2.7. The set $\{w : (v, w) \in E,\ w \in V\}$ used in the definition of the degree of a vertex is the set that contains all the neighbours of v and will be denoted by $\mathcal{N}(v)$ where $\mathcal{N} : V \to \mathcal{P}(V)$.

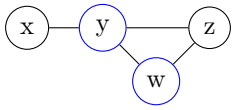


Figure 2: The degree of vertex z is 2, in blue we see $\mathcal{N}(z) = \{y, w\}$

To talk about the chip-firing game, we first need to introduce chips to the graph. We do this by associating a number with every vertex. Informally, we will then perform 'moves' on the graph, which consists of moving chips around the graph in a controlled manner.

Definition 2.8. For a graph $G = (V, E)$ we define $a_v \in \mathbb{Z}$ for every vertex $v \in V$, indicating the number of *chips* on that vertex. Between moves we will talk about a_v as the number of chips before the move, and a'_v as the number of chips after the move. Note that $a_v < 0$ is possible, and we say that vertex v is in debt.

From here on in any examples, the numbers on the vertices will represent the number of chips on that vertex. When relevant, the identifier will also be shown.

Before we define the chip-firing game, we will make a generalization for storing chips on graphs.

Definition 2.9. By looking at all the a_v at the same time, we can define the state of a game by looking at a vector $a \in \mathbb{Z}^n$ with an arbitrary ordering on V such that $V = \{v_i : i \in \{1, 2, ..., n\} \subset \mathbb{N}\}$. Such a vector is also known as a *divisor* on G, and we call $\mathrm{Div}(G)$ the group of all divisors on G.

Remark 2.10. Since there exists a trivial isomorphism between $\mathrm{Div}(G)$ and $\mathbb{Z}^n$ under addition, we will treat elements of $\mathrm{Div}(G)$ as vectors in $\mathbb{Z}^n$ any time we perform calculations or follow algorithms.

Definition 2.11. The amount of chips in the game is the sum of all components of $D \in \mathrm{Div}(G)$. This is also known as the *degree* of the divisor D.

Definition 2.12. We denote $\mathrm{Div}^k(G)$ the set of all divisors of degree k on G.

Remark 2.13. Combining Definition 2.11 and Definition 2.12, we note that $\mathrm{Div}^0(G)$ forms a group under addition. We can show that the degree of $D_1 + D_2$ where $+$ adds components, is the sum of the degrees of D_1 and D_2.

Remark 2.14. Note that $\mathrm{Div}^k(G)$ for $k \neq 0$ is not a group under addition, since for $D_1, D_2 \in \mathrm{Div}^k(G)$:

$$\deg(D_1 + D_2) = \deg(D_1) + \deg(D_2) = k + k = 2k.$$

Therefore $D_1 + D_2 \notin \mathrm{Div}^k(G)$.

Definition 2.15 (*Firing* single vertices)**.** By *firing* a vertex v, we remove $\deg(v)$ chips from v and add one to each of its neighbours.

Definition 2.16 (Firing subsets)**.** By choosing a subset $A \subseteq V$, we can fire multiple vertices in one move. This is equivalent to firing all vertices in the subset sequentially, but is considered one move.

Definition 2.17 (The chip-firing game)**.** Given an undirected, unweighted, and connected graph $G = (V, E)$ we let $n = |V|,\ m = |E|$. We can then play the chip-firing game as follows:

1. Assign a number a_v of chips to every vertex $v \in V$.
2. Pick any subset $A \subseteq V$ such that after firing A, we have $a'_v \geq 0$ for all $v \in A$.
3. Repeat until certain goal is reached.

Definition 2.18. Using the chip-firing game as defined in Definition 2.17, given a state $D \in \mathrm{Div}(G)$, we can calculate a new state $D' \in \mathrm{Div}(G)$ after firing a single vertex v as follows:

1. For every vertex v_i we define $m_{v_i} \in \mathbb{Z}^n$ such that:

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

   Then m_{v_i} is the vector that defines the change to the overall graph after firing v_i, since all neighbours of v_i receive 1, and v_i itself loses an amount equal to the amount of neighbours.

2. We then define $D' = D + m_v$ where $+ : \mathrm{Div}(G) \times \mathbb{Z}^n \to \mathrm{Div}(G)$ simply adds components.

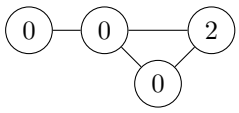
In order to talk about firing subsets, we generalize this to performing a set of moves at once for $A \subseteq V$ by defining:

$$m_A = \sum_{v \in A} m_v,$$

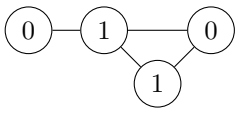
and then calculating $D' = D + m_A$ similarly. As such, any subset $A \subseteq V$ generates a move vector m_A.

The following example illustrates how the m_{v_i} encode the process of chip-firing on a small graph.

Example 2.19. Consider the following graph corresponding to $D = (0, 0, 2, 0) \in \mathrm{Div}(G)$:



(a) D, before firing



(b) D', after firing

Figure 3

The vector m_{v_3} corresponding to firing the vertex with 2 chips on it is given by:

$$m_{v_3} = (0, 1, -2, 1).$$

Indeed, we can see that giving one chip to each of the neighbours yields the configuration shown in Figure 3b which is indeed given by $D' = D + m_{v_3} = (0, 0, 2, 0) + (0, 1, -2, 1) = (0, 1, 0, 1)$. △

Remark 2.20. Note that $m_V = \vec{0}$, since when every vertex gives one to each of its neighbours, and receives one from each of his neighbours, the net result is 0.

3 Chip-firing and the Laplacian

In order to more effectively discuss the properties of the chip-firing game and the groups associated with it, we look at a few theorems appropriate for computation of moves, and move toward a general idea of defining an equivalence relation on divisors under chip-firing moves. We will first look at some theorems that follow from the generalizations we made to firing subsets, to show how the Laplacian of the graph is connected to chip-firing games in Remark 3.10.

Lemma 3.1. *When we have two disjoint subsets $A, B \subseteq V$, $m_A + m_B = m_{A \cup B}$.*

Proof. Since no elements are shared between A and B we get:

$$\begin{aligned} m_A + m_B &= \sum_{v \in A} m_v + \sum_{v \in B} m_v \\ &= \sum_{v \in A \cup B} m_v \\ &= m_{A \cup B}, \end{aligned}$$

which completes the proof. □

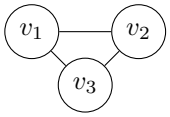
Theorem 3.2. *Firing a subset $A \subseteq V$ has the opposite effect of firing A^c.*

Proof. Combining Remark 2.20 and Lemma 3.1 we know:

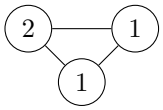
$$\begin{aligned} m_A + m_{A^c} &= m_{A \cup A^c} \\ &= m_V \\ &= 0. \end{aligned}$$

This means that $m_A = -m_{A^c}$. In other words, firing the complement of a subset A undoes the move. □

Example 3.3. An example of an application of Theorem 3.2 is showing that firing everything but one vertex is the same as reversing the effect of firing that vertex. See the following graph:

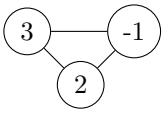


(a) Ordering of the graph

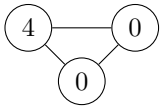


(b) Initial configuration s

We know that $m_{v_1} = (-2, 1, 1)$. It is thus expected from Theorem 3.2 that $m_{\{v_2,v_3\}} = (2, -1, -1)$. We now show the result of first firing v_2 and then firing v_3. Reversing the order results in the same final configuration.



(a) Firing v_2 first.



(b) Firing v_3 second. Final configuration D'.

We can see here that the change to the graph by firing $m_{\{v_2,v_3\}}$ is indeed equal to $(2, -1, -1)$. △

Corollary 3.4. *As a result of Theorem 3.2 we can always calculate a new move by looking at firing at most $\lfloor \frac{n}{2} \rfloor$ vertices by looking at the complement of A whenever $|A| > \frac{n}{2}$.*

Corollary 3.4 is mostly useful for on-paper evaluations.

Before we use these properties to show some alternative representations of graphs, we need one more property associated to divisor groups.

Proposition 3.5. *The rank of* $\mathrm{Div}^0(G)$ *is* $n - 1$.

Proof. Any element of $\mathrm{Div}^0(G)$ can be represented as a vector v in $\mathbb{Z}^n$ like:

$$v = \left(v_1, v_2, ..., v_{n-1}, \sum_{i=1}^{n-1} -v_i \right).$$

Now define u_i as the vector with:

$$(u_i)_j = \begin{cases} 1 & \text{if } j = i, \\ -1 & \text{if } j = n, \\ 0 & \text{otherwise.} \end{cases}$$

Notice that we can now write v as a linear combination of u_i like:

$$v = \sum_{i=1}^{n-1} v_i \cdot u_i.$$

Therefore $\mathrm{Div}^0(G)$ is spanned by $\{u_i : 1 \leq i \leq n-1,\ i \in \mathbb{N}\}$ since all the u_i are linearly independent, and is therefore of rank $(n-1)$. □

In order to define another group of divisors, we first take a look at some matrices determined by graphs.

Definition 3.6. Given a graph $G = (V, E)$ we can construct the $n \times n$ *degree matrix* D of the graph by putting the degrees of all vertices on the diagonal as follows:

$$D_{i,j} = \begin{cases} \deg(v_i) & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases}$$

Definition 3.7. Given a graph $G = (V, E)$ we can construct the $n \times n$ *adjacency matrix* A of the graph by putting the weight of edge (v_i, v_j) at $A_{i,j}$. Since we look at unweighted graphs, our edge weights are considered to be 1:

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

With the degree matrix and the adjacency matrix defined, we can look at the subtraction of the two, called the *Laplacian matrix*.

Definition 3.8. The $n \times n$ *Laplacian matrix* L of a graph G is given by:

$$L := D - A.$$

Per entry this means:

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

As mentioned earlier, the Laplacian is strongly connected to the chip-firing game. We will use our earlier findings to construct a matrix Q such that Q is equal to the negative Laplacian.

Since we have defined n vectors m_{v_i} in Definition 2.17, each of length n; we can stack these vectors following the ordering chosen on V and create a matrix Q that can be used to compute the same m_A as in Definition 2.18 using a matrix-vector multiplication.

Definition 3.9. Let the $n \times n$ matrix Q be the matrix obtained by using m_{v_i} (see Definition 2.17) as row vectors for a given graph G such that:

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

Remark 3.10. Notice that $L = -Q$, as we can quickly see in Definition 3.8 and Definition 3.9.

As a result we can now compute simple moves of firing a subset using a multiplication.

Proposition 3.11. *Let $A \subseteq V$ be a non-empty subset, then χ_A is the characteristic $\{0, 1\}$-valued column vector obtained by:*

$$(\chi_A)_i = \begin{cases} 1 & \textit{if } v_i \in A, \\ 0 & \textit{otherwise.} \end{cases}$$

Then m_A is also defined by:

$$m_A = Q\chi_A.$$

Proof. Since Q consists of the m_{v_i} we need to construct m_A, multiplication with χ_A performs the exact sum required as in Definition 2.18. □

In fact, this definition allows us to compute the change to a divisor by firing any vertex any positive number of times.

Definition 3.12. Given a sequence of moves, decomposed in subsets $A_i \subseteq V$, we can define the column vector $(f) \in \mathrm{Div}(G)$ such that:

$$(f) = \sum_i \chi_{A_i}.$$

We can then compute the change to the graph by:

$$\begin{aligned}\sum_i Q\chi_{A_i} &= Q\sum_i \chi_{A_i} \\ &= Q(f).\end{aligned}$$

This idea leads us to another interesting group.

Definition 3.13. Let $\mathrm{Prin}(G)$ be the group of *principal divisors*. A divisor $D \in \mathrm{Prin}(G)$ is such that there is a $(f) \in \mathrm{Div}(G)$ such that:

$$D = Q(f).$$

Since $Q = -L$, this is equivalent to the group of all divisors D such that:

$$D = L(f),$$

which allows us to use known properties of the Laplacian.

More concretely, the group of principal divisors are all the changes that can happen to a divisor on a graph under any sequence of chip-firing moves. We will show next that it is in fact the group of changes that can happen under any 'legal' sequence of chip-firing moves. Legal in this context means that we only use positive components in (f), but still span the entire group.

Proposition 3.14. $\mathbb{1} \in \ker(Q)$ *and by extension also* $\mathbb{1} \in \ker(L)$.

Proof. We know that $m_V = \vec{0}$. Since $m_V = Q\chi_V = \vec{0}$, and we also know that $\chi_V = \mathbb{1}$, we can conclude that $\mathbb{1} \in \ker(Q)$. □

Corollary 3.15. *Given (f) a sequence of moves. If and only if $(f)_v = (f)_w$ for all $v, w \in V$, then $Q(f) = 0$. That means,* $\ker(Q)$ *is spanned by* $\mathbb{1}$.

Proof. From Proposition 3.5 we know that the rank of Q is $n-1$. We know that the rank of a matrix plus the dimension of the null space (Rank Theorem in [5]) is equal to n. This means that the dimension of the null space is one, and we know from Proposition 3.14 that $\mathbb{1} \in \ker(Q)$. We conclude that $\ker(Q)$ is spanned by $\mathbb{1}$. □

Lemma 3.16. $Q(f) = Q(g)$ *where* $(g) = (f) + c \cdot \mathbb{1}$ *and* $c \in \mathbb{Z}$.

Proof. By distributivity of matrix multiplication and Proposition 3.14:

$$\begin{aligned}Q(g) &= Q((f) + c \cdot \mathbb{1}), \\ &= Q(f) + c \cdot Q\mathbb{1}, \\ &= Q(f).\end{aligned}$$

□

Remark 3.17. The group $\mathrm{Prin}(G)$ is generated by the divisors $Q(f_i)$ such that the i-th component of (f_i) is one, and the rest is zero, for $1 \leq i \leq n-1$ and $i \in \mathbb{N}$. Note that we do not need $Q(f_n)$, since using the complement property described in Theorem 3.2,

$$Q(f_n) = -Q \sum_{1 \leq i \leq n-1} (f_i).$$

Important to notice is that since every row sum of Q is 0, we know that any $D \in \text{Prin}(G)$ is also in $\text{Div}^0(G)$. Therefore, $\text{Prin}(G) \subseteq \text{Div}^0(G)$.

We have now reached a point where we can define an interesting equivalence relation on divisors, or in the context of the chip-firing games, starting configurations of the game.

Definition 3.18. We call two divisors D_1 and D_2 linearly equivalent, denoted by $D_1 \sim D_2$, if there is a sequence of moves such that we can move from D_1 to D_2. In other words $D_1 \sim D_2$ implies the existence of a $\Delta D \in \text{Prin}(G)$, the result of a sequence of moves such that $D_2 = D_1 + \Delta D$.

This might raise the question, what if our $\Delta D \in \text{Prin}(G)$ is generated by an $(f) \in \text{Div}(G)$ with a negative component? To solve this, we need one more idea, namely *effective* divisors.

Definition 3.19. A divisor $D \in \text{Div}(G)$ is called *effective* if $D_v \geq 0$ for all $v \in V$.

This means that we can assume a given (f) to be effective without loss of generality, since Lemma 3.16 tells us there always is a (g) such that (g) is effective and $Q(f) = Q(g)$. We will later use this definition more generally by saying that a divisor D is effective outside a subset $A \subseteq V$, which means that the divisor is non-negative for all vertices in the complement of A.

4 Dhar's burning algorithm

Since $\text{Prin}(G)$ is an infinite group, it is not trivial to check whether $D_1 \sim D_2$. We can however, show that there exists some notion of a shortest path.

Theorem 4.1. *We can find the shortest number of moves between effective divisors D_1 and D_2 given that we know some effective $(f) \in \text{Div}(G)$ such that $\Delta D = Q(f)$ gives $D_2 = D_1 + \Delta D$ and thus $D_1 \sim D_2$.*

Proof. Using Lemma 3.16 we know we can add $\mathbb{1}$ to (f) any number of times. Therefore we can subtract $\mathbb{1}$ in such a way that (f) is not only effective, but the minimum component is 0. Our maximum component $M = \max_{v \in V}(f)_v$ is now the minimal number of subsets we need to fire in order to move from D_1 to D_2. □

Lemma 4.2. *If we have divisors D_1 and D_2 such that $D_1 \sim D_2$, and we know effective divisors (f) and (g) such that:*

$$\begin{aligned} D_2 &= D_1 + Q(f), \\ D_1 &= D_2 + Q(g). \end{aligned}$$

Then $(f) + (g) = M \cdot \mathbb{1}$ where $M \in \mathbb{Z}$ is such that $M \geq \max_{v \in V}(f)_v$.

Proof. Adding both equations gives us:

$$\begin{aligned} D_1 + D_2 &= D_1 + D_2 + Q(f) + Q(g), \\ 0 &= Q(f) + Q(g), \\ Q((f) + (g)) &= 0. \end{aligned}$$

And therefore $(f) + (g) \in \ker(Q)$. Since $\ker(Q)$ is spanned by $\mathbb{1}$ by Corollary 3.15, we know that for all $v \in V$ that $(f)_v + (g)_v = M$ for some $M \in \mathbb{Z}$. Since both (f) and (g) are effective, both $(f)_v \geq 0$ and $(g)_v \geq 0$ following Theorem 4.1, and we can conclude that M is at least equal to $\max_{v \in V}(f)_v$. □

Perhaps a more interesting concept to look at is the concept of equivalence classes under the relation described in Definition 3.18. A nice set of representatives for these equivalence classes are the q-reduced divisors. We will use the definition as described in the paper by Baker-Shokrieh [4].

Definition 4.3. Choose any vertex $q \in V$. A divisor $D \in \text{Div}(G)$ is called *q-reduced* if it satisfies the following conditions:

1. $D_v \geq 0$ for all $v \in V \setminus \{q\}$, or in other words, D is effective outside q.
2. There is no subset $A \subseteq V \setminus \{q\}$ that can be fired such that after firing all vertices in A still have a non-negative number of chips.

Note from Property 1 that it is possible that the amount of chips at q is negative. This is to allow this definition to span all divisors of any degree.

We can now show that the equivalence classes created under $\sim$ are represented by the q-reduced divisors. To this end, we need to show that q-reduced divisors are unique. In other words, we will have to show that if D_1 and D_2 are both q-reduced divisors, and we have an arbitrary divisor D such that $D \sim D_1$ and $D \sim D_2$ hold, then this implies that $D_1 = D_2$ and the q-reduced divisor is therefore unique. In order for this to be possible we first need some intermediate steps. The following proof is based on Baker-Norine [1, Proposition 3.1].

Remark 4.4. In the following proofs we will use the notion of 'lending' chips. This means that we force the vertex we want chips from to fire, resulting in the informal idea of lending chips.

Lemma 4.5. *Any divisor $D \in \mathrm{Div}(G)$ is equivalent to a q-reduced divisor.*

Proof. We take two steps in order to show that any $D \in \mathrm{Div}(G)$ can be reduced with respect to q.

1. Order the vertices in V arbitrarily such that q is the first vertex and every vertex has a neighbour preceding it in the ordering (except for q). Then start, from the last vertex, by lending from one of its preceding neighbours until the amount of chips on that vertex is greater than or equal to 0. Repeat this process for every vertex (except q) by only lending from vertices preceding it in the ordering. We now have that all $D_i \geq 0$ for $i \neq q$. The divisor then satisfies the first property as in Definition 4.3.
2. To achieve the second property, we simply keep firing all subsets $A \subseteq V \setminus \{q\}$ until we are not able to anymore. This is guaranteed to terminate since chips will end up on q, thus reducing the amount of chips in the rest of the graph. Since the amount of chips is finite, there will eventually not be any subset left to fire and the second property is also fulfilled.

With both properties of Definition 4.3 fulfilled, we have succesfully reduced an arbitrary divisor D to vertex q. □

The interesting part about Lemma 4.5 is how we can efficiently perform the two steps in the described algorithm.

Theorem 4.6. *For any divisor $D \in \mathrm{Div}(G)$ there is a unique q-reduced divisor D' such that $D \sim D'$.*

Proof. Using Lemma 4.5 we know any D is linearly equivalent to at least one q-reduced divisor. Suppose we have D_1 and D_2 such that $D_1 \sim D_2$ and $D_1 \neq D_2$. Assume D_1 and D_2 are both q-reduced such that D is linearly equivalent to both. Suppose we know (f) such that $D_2 = D_1 + Q(f)$. Informally, this means that (f) is a characteristic function for what vertices we should fire in D_1 to get to D_2. From Theorem 4.1 we can assume (f) effective without loss of generality such that:

$$\min_{v \in V}((f)_v) = 0. \tag{4.1}$$

Since this argument needs to work both ways; there exists (g) such that $D_1 = D_2 + Q(g)$ and $(f)+(g) = M \cdot \mathbb{1}$ following Lemma 4.2 since Equation (4.1) holds for both (f) and (g). Since $D_1 \neq D_2$, we know that there is a vertex $v \in V \setminus \{q\}$ such that $(f)_v \neq (f)_q$. Since we know that $(f) + (g) = M \cdot \mathbb{1}$, we can assume that there is a $(f)_v > (f)_q$ without loss of generality. As a result there is a subset $A \subseteq V \setminus \{q\}$ with $v \in A$ that can be fired, contradicting the fact that D_1 and D_2 are both q-reduced, thus proving that D is linearly equivalent to a *unique* q-reduced divisor. □

Now we know that we can properly represent the equivalence classes, we can in fact figure out whether two divisors belong to the same equivalence class, and thus whether they are linearly equivalent. From now on we will only look at *effective* divisors unless explicitly stated. To efficiently reduce a state to its q-reduced representative, we can employ Dhar's [6] algorithm (Algorithm 1) to take a divisor D and perform moves on it such that it is q-reduced. Before we formally describe Dhar's algorithm, we need an extra function:

Definition 4.7. Let $\mathcal{N}(v)$ be the neighbours function as described before. We generalize (with $A \subseteq V$) by:

$$\mathcal{N}_A(v) = \mathcal{N}(v) \cap A.$$

This allows us to look at the number of incident edges from a subset of V to an arbitrary vertex v. Furthermore we define $\mathcal{N}(A)$ as follows:

$$\mathcal{N}(A) = \left(\bigcup_{v \in A} \mathcal{N}(v) \right) \cap A^c.$$

In other words, $\mathcal{N}(A)$ gives all vertices adjacent to A but not in A.

Next we will be looking at an algorithm based on work by Dhar [6] and adapted from the algorithm in Baker-Shokrieh [4] that is used to determine whether a given divisor is q-reduced. This algorithm takes in any divisor D and a vertex q, and returns the same divisor reduced with respect to q, assuming the input is effective, as opposed to Algorithm 1 in Baker-Shokrieh, which gives a true or false depending on whether the divisor is q-reduced or not. If the divisor D is already q-reduced, it returns that divisor in the same time complexity as the algorithm described by Baker-Shokrieh, otherwise it performs chip-firing moves to perform the reduction.

In order to understand the algorithm we present the following analogy. We assume the chips on the graph represent firefighters, and we start a fire at vertex q. Edges do not have firefighters stationed on them, so all edges adjacent to q 'burn'. Any vertices with an incoming burning edge are now at risk. If they have enough firefighters stationed there, they are safe and nothing happens. However, in the case that the amount of burning edges exceeds the number of firefighters, the vertex is unfortunately lost to the fire, and all of its adjacent edges burn.

As a result of these steps, when the fire is controlled and nothing burns anymore, all vertices that are still unburned together form a subset $A \subseteq V \setminus \{q\}$ that can be fired. A formal proof is given in Theorem 4.11.

Algorithm 1: Dhar's Burning Algorithm

```
Input  : Effective divisor D and vertex q ∈ V
Output: q-reduced divisor D' such that D ~ D'
D' := D
while B ≠ V do
    B := {q} // Set of burned vertices
    while B changed do
        A := N(B) // Set of vertices with burning incident edges
        for a ∈ A do
            if D'_a < |N_B(a)| then
                // a burns through
                B ← B ∪ {a}
            end
        end
    end
    D' ← D' + k · m_{B^c} // Fire all unburned vertices
end
```

In the current shape this algorithm runs particularly badly in specific cases, since we are firing every vertex only once. Figure 6 shows the graph of 2 vertices connected by one edge for example. Put an arbitrary large amount N of chips on the second vertex, and then reduce the graph for the first vertex. We would have to step through the outer while loop at least that large number N times to move every chip one at a time.

(a) Initial configuration

(b) Configuration after firing once

Figure 6: Example of bad case

To achieve a significant speed up in cases like this, we take a look at the line where we fire the unburned vertices:

$$D' \leftarrow D' + k \cdot m_{B^c}.$$

We can find a coefficient $k \in \mathbb{Z}$ to m_{B^c} such that we perform the same firing move as many times as possible. We can find the change to the amount of chips for every vertex $v \in B^c$ as follows:

$$\Delta v = |\mathcal{N}_B(v)| = (m_{B^c})_v = a'_v - a_v.$$

Using this idea we can calculate the number of times we can fire without going into the negative as:

$$k = \min_{v \in \mathcal{N}(B)} \left(\left\lfloor \frac{D_v}{-\Delta v} \right\rfloor \right).$$

This solves in particular the earlier example of 2 vertices with a lot of chips on one vertex, but for any divisors with a large number of chips (degree of divisor much larger than n) this allows for more efficient computation. However, we have not yet proven that this does indeed do what it says. First, we will state why this algorithm terminates.

Proposition 4.8. *In every step of the algorithm, one of two things happens:*

1. *The amount of chips on $V \setminus \{q\}$ goes down and the number of chips on q goes up.*
2. *The unburned portion B^c of the graph changes and the amount of chips on $V \setminus \{q\}$ stays the same.*

Proof. We know that if $q \in \mathcal{N}(B^c)$ that D_q will grow. We also know that the amount of chips in the game does not change and thus the number of chips on $V \setminus \{q\}$ goes down. This shows the first property.

For the second property, we now consider $q \notin \mathcal{N}(B^c)$. Since we fire the subset exactly as many times as is possible, we know that subset B^c has to change from one iteration to the other. □

Lemma 4.9. *Algorithm 1 terminates.*

Proof. Firstly, the inner while loop terminates, since B can only grow to be as big as V. Then, $\mathcal{N}(B)$ is empty, and thus B can no longer change and the loop will terminate.

From Property 2 in Proposition 4.8 we know that each iteration, we have a different B^c. As a result, any movement we make in the chips can only be undone by firing the complement, which includes q. We never fire q and thus it is impossible to return to previous states. Since there are finitely many chips, there are finitely many states we can have before q will be a part of $\mathcal{N}(B^c)$ (Property 1 in Proposition 4.8), or the entire graph will burn.

Since the amount of chips on the graph is finite, eventually the number of chips on $V \setminus \{q\}$ will be sufficiently low, and thus the graph will burn and we have $B = V$ and the outer while loop terminates. We conclude that Algorithm 1 indeed terminates. □

Lemma 4.10. *Algorithm 1 only does legal moves.*

Proof. We know that for any legal move from effective divisor D to D', all D'_v should be non-negative. Firstly, all vertices in B are not fired, and thus do not lose any chips and remain at a nonnegative amount of chips. Secondly, for all $v \in B^c$ we know the following relation:

$$D'_v = D_v - |\mathcal{N}_B(v)|.$$

In the algorithm we check for every vertex adjacent to B (so $|\mathcal{N}_B(v)| > 0$)) whether the following holds:

$$D_v < |\mathcal{N}_B(v)|.$$

If this holds, we add v to B, which results in v not being fired and thus not losing any chips. That ensures that $D'_v \geq D_v$ and since D is effective, we know $D'_v \geq 0$. If it does not hold, then $D_v \geq |\mathcal{N}_B(v)|$ must be true, and thus $D'_v \geq 0$. □

Theorem 4.11. *Algorithm 1 reduces any effective divisor with respect to q.*

Proof. We need to show that when the algorithm terminates, there is no more subset $A \subseteq V \setminus \{q\}$ such that it can be fired and all $D'_v \geq 0$ for $v \in V \setminus \{q\}$.

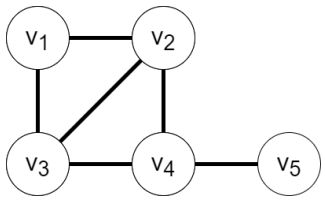
When the algorithm terminates, we have $B = V$. This means that all vertices in $V \setminus \{q\}$ have been added to B. A vertex v is added to B if and only if it has fewer chips than it has burning neighbours. Any burning neighbour can not fire and therefore v will lose one chip per burning neighbour. If v is then added to B, this is because v itself can not fire without D'_v going below 0.

Using this idea, we know that q never fires. This means that there is a vertex $w \in \mathcal{N}(q)$ that can not fire since q will not fire. Propagate this through the graph and we can conclude that there is no more subset $A \subseteq V \setminus \{q\}$ that can be fired, and the second property of Definition 4.3 is fulfilled.

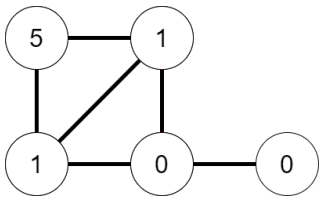
Using Lemma 4.10 we know the first property of Definition 4.3 is also fulfilled, and thus D' is q-reduced. □

Lastly, we will cover an example of Dhar's burning algorithm in an informal way.

Example 4.12. We will use Algorithm 1 to v_5-reduce the following graph and configuration:



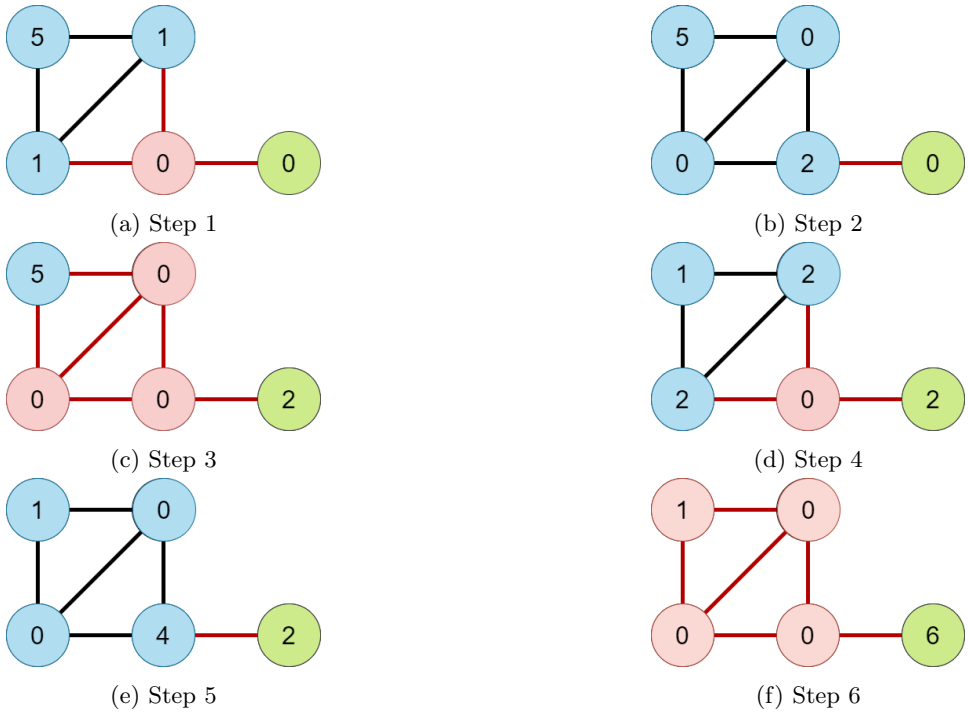
(a) Labelling



(b) Configuration

We will now use colours to represent the steps in Algorithm 1. In *green* we will show the vertex we are reducing to, q in the pseudo-code. In *red* we will indicate burning edges and vertices. In *blue* we will show those untouched by the fire. In terms of the algorithm, the set B consist of all the red vertices and the green (since q is always burning), and B^c, which will be fired, consists of all the blue vertices.

Informally, we can consider these blue vertices to be safe because they have enough firefighters (chips) stationed there to fight off the amount of incoming fires. Firing a vertex would then, in the same analogy, be equivalent to sending out the fire fighters to help the neighbouring vertices.



(a) Step 1 (b) Step 2

(c) Step 3 (d) Step 4

(e) Step 5 (f) Step 6

Figure 8: Steps of Dhar's algorithm

As we can see in Figure 8f, the algorithm terminates when everything is marked ($B = V$) and we can no longer fire any subset of $V \setminus \{q\}$. Between step 1-2 we can also see how the number of chips only changes in blue nodes that have a red or green neighbour, and vice versa.

Note that in all steps but 1 and 6 (in which we do not fire at all), the firing is done multiple times. It is most clear in step 5, where we fire all blue vertices 4 times to reach the configuration in Figure 8f.

Our (f) in this case is equal to the sum of all blue nodes multiplied by how many times they were fired in that step. As Example 4.12 shows:

Step	χ_{B^c}	k
1	(1, 1, 1, 0, 0)	1
2	(1, 1, 1, 1, 0)	2
3	(1, 0, 0, 0, 0)	2
4	(1, 1, 1, 0, 0)	2
5	(1, 1, 1, 1, 0)	4
6	(0, 0, 0, 0, 0)	N/A

Table 1: Steps taken in the example, where k is the number of times we fire B^c.

Our total for (f) is then the weighted sum of the χ_{B^c} column. In this case:

$$\begin{aligned}(f) &= 1 \cdot (1,1,1,0,0) + 2 \cdot (1,1,1,1,0) + 2 \cdot (1,0,0,0,0) + 2 \cdot (1,1,1,0,0) + 4 \cdot (1,1,1,1,0), \\ &= (11,9,9,6,0).\end{aligned}$$

This also shows how the algorithm can be adapted to keep track of this (f) vector and return it in addition to the q-reduced divisor. △

Lastly we offer an algorithm for reducing any divisor (including non-effective divisors) with respect to a vertex q on paper. This is less efficient than Algorithm 5 in Baker-Shokrieh [4], but does not require the user

to compute a rather cumbersome matrix ($L_{(q)}$ in [4]) on paper. Since we have already provided Algorithm 1 to q-reduce any effective divisor, we now note that any divisor effective outside of q is also q-reduced since Definition 4.3 does not require the amount of chips on q to be positive. This means that we can introduce an ordering on V using the following lemma.

Lemma 4.13. *We can order all vertices $v \in V \setminus \{q\}$ such that any v has a neighbour preceding it in the ordering.*

Proof. Start by setting q to be the first vertex in the ordering, say v_0. Then for all neighbours of q we can set them to be $v_1, v_2, ..., v_{|\mathcal{N}(q)|}$ arbitrarily, since any of them will have q as a neighbour preceding them. Proceeding with this process for all newly labelled vertices and assigning the ordering achieves the intended result. □

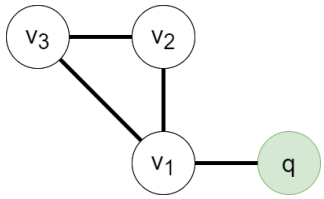
Since our goal is to do this on paper, the following step is not computationally optimal but allows for manual computation.

Lemma 4.14. *There exists an ordering, such that by starting at the last vertex in the ordering, and lending only from neighbours that precede it in the ordering, we can make any graph G effective outside q.*

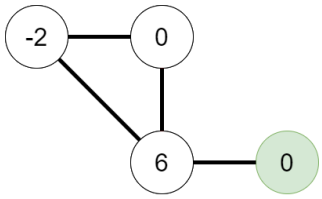
Proof. Suppose the ordering as in Lemma 4.13 is used. Then starting at the last vertex, and lending from a preceding vertex is always possible, since every vertex $v \in V \setminus \{q\}$ has a preceding neighbour. □

In practice, it is recommended to choose the preceding vertex with the lowest degree such that the number of affected vertices by repeated firing is lowest if the intent is to make the divisor effective outside q. However, if the goal is to q-reduce the divisor, it is advised to first borrow from all neighbours that can still fire without going negative. In many cases this reduces the number of vertices that need to do any lending. The following example shows the difference between the two, and how it affects the resulting divisor.

Example 4.15. First we look at the initial ordering and chip assignment:



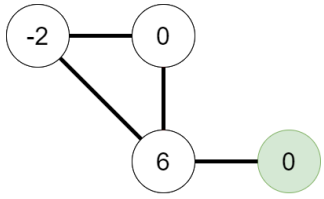
(a) Ordering on G.



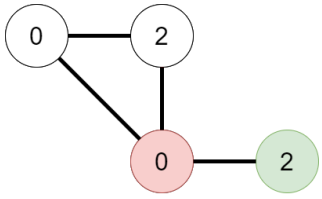
(b) Chips on G.

Figure 9: Graph G.

We will first look at the resulting choices when first firing as many neighbours as possible that can still fire without going negative. We will indicate in red the vertex that fired to reach that particular state.



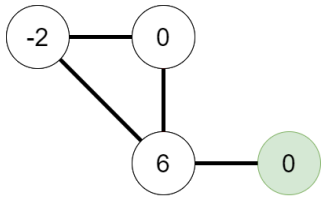
(a) Before changes.



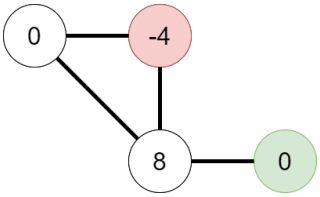
(b) After firing v_1 twice.

Figure 10: Lend from rich neighbours first.

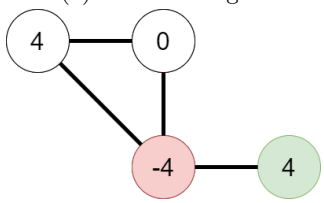
As shown in Figure 2, we are done after just one move of firing v_1 twice, the divisor is now effective outside q. Next we will see why this is beneficial.



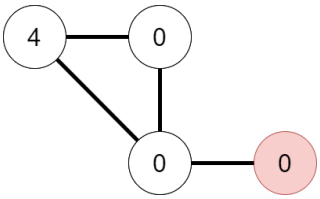
(a) Before changes.



(b) After firing v_2 twice.



(c) After firing v_3 4 times.



(d) After firing q 4 times.

Figure 11: Lend from lowest degree neighbour

We see that we get a different result, but not only is it a different result it required twice as many steps, and in total 9 individual chip-firing moves were performed, in contrast to the 2 of the other method. It is left to the reader to check that both of these divisors do indeed reduce to the same q-reduced divisor represented by $(1, 0, 0, 3)$. △

5 Spanning trees and reduced divisors

In this chapter we will discuss the relation between spanning trees and q-reduced divisors on graphs and show a bijection between spanning trees and q-reduced divisors.

Definition 5.1. A *spanning tree* on an undirected, connected graph G is a subset $T \subseteq E$ such that $|T| = n-1$ where $n = |V|$, and the graph $G' = (V, T)$ is still connected.

Remark 5.2. Any spanning tree T of a graph $G = (V, E)$ is a subset of E of minimal size such that $G' = (V, T)$ is connected.

Since we will be working more with edges we generalize $\mathcal{N}$ once more.

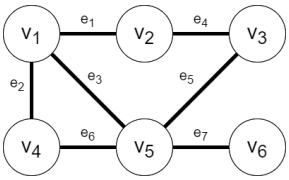
Definition 5.3. We define $\mathcal{N}^T(v)$ to be $\{w : (v, w) \in T\}$ where T is a subset of E. More concretely, $\mathcal{N}^T(v)$ gives all neighbours of v such that for any neighbour w, the edge in between is included in T.

To show how we can go from any q-reduced divisor to a spanning tree, we give an informal example.

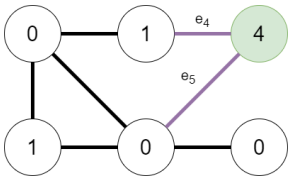
Example 5.4. We first present a graphical representation of the algorithm that is based on the same idea as Algorithm 1, except now we deterministically traverse edges. We use colours to indicate a few properties; green for the vertex to which the divisor is reduced to. Purple are edges to be considered, and red are the edges and vertices that have been burned.

Informally, the steps taken in this algorithm are almost the same as in Algorithm 1. We start at the vertex to which the divisor is reduced. We then, for each vertex, burn all of its neighbours. In contrast to Algorithm 1, where we choose neighbours arbitrarily, we now choose the neighbour along the edge that comes first in the ordering, all other edges we mark for later (purple).

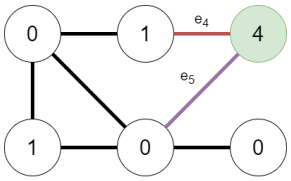
When we burn (process) an edge, we check if the number of firefighters (chips) is enough to fight the incoming fires. If not, we burn the vertex and add that edge to the tree; all surrounding neighbours are added to the edges that will burn (purple). In this way we deterministically burn through all vertices in an order that we will use to show that Algorithm 3 is indeed the inverse of Algorithm 2.



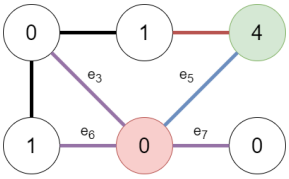
(a) Labelling of vertices and edges.



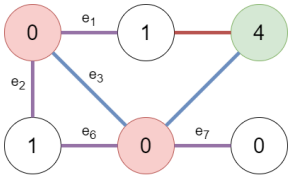
(b) Initial state.



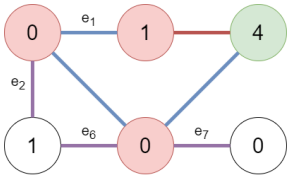
(c) Edge e_4 burns, but v_2 does not burn.



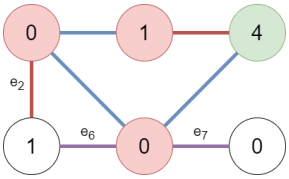
(d) Edge e_5 burns and is added to T. Vertex v_5 burns.



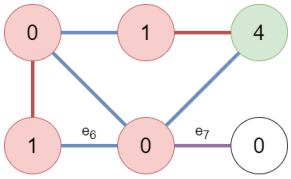
(e) Edge e_3 burns and is added to T. Vertex v_1 burns.



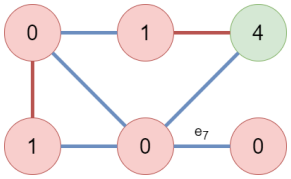
(f) Edge e_1 burns and is added to T. Vertex v_2 burns.



(g) Edge e_2 burns.



(h) Edge e_6 burns and is added to T. Vertex v_4 burns.



(i) Edge e_7 burns and is added to T. Vertex v_6 burns.

Figure 12: From v_3-reduced divisor to spanning tree.

We see that in Figure 12i the blue lines represent the tree obtained from the v_3 reduced divisor $(0, 1, 4, 1, 0, 0)$:

$$T = \{e_1, e_3, e_5, e_6, e_7\}.$$

△

We now present the algorithm formally, after which we prove its properties.

Algorithm 2: Divisor to spanning tree

Input : $q \in V$ and q-reduced divisor D
Output: Spanning tree T
$T := \emptyset$
$H := \mathcal{N}(q)$ `// Set of unburned edges between burned and unburned vertices`
$B := \emptyset$ `// Set of burned edges`
$X := \{q\}$ `// Set of burned vertices`
while $|H| > 0$ **do**
  $(x, y) := \min\{e : e \in H\}$ `// Current edge`
  if $(x, y) \in B$ *or* $x, y \in X$ **then**
    `// In this case the edge has or the vertices have already been burned`
    Pass
  end
  `// Assume` $x \in X$, $y \notin X$
  if $D_y = |\mathcal{N}^B(y)|$ **then**
    `//` y `burns through and` (x, y) `is therefore added to the tree`
    $X \leftarrow X \cup \{y\}$
    $T \leftarrow T \cup \{(x, y)\}$
    for $w \in \mathcal{N}(y)$ **do**
      `// For all neighbours, add the corresponding edge to the set to be handled`
      $H \leftarrow H \cup \{(y, w)\}$
    end
  end
  $H \leftarrow H \setminus \{(x, y)\}$ `// Current edge processed, remove from` H
  $B \leftarrow B \cup \{(x, y)\}$ `// Add the edge to the burned edges`
end
Output T

Remark 5.5. Comparing Algorithm 2 to Example 5.4, we see that the red vertices (and the green starting vertex q) are contained in X at every step. The blue edges are contained in T and B, and the red edges in B exclusively. The purple edges that are marked for processing are contained in H at every step.

Theorem 5.6. *Algorithm 2 outputs a spanning tree of G.*

Proof. Recall properties of a q-reduced divisor from Definition 4.3. It holds that there always is a $(x, y) \in H$ such that $D_y = |\mathcal{N}^B(y)|$ and thus has exactly as many burning incident edges as it has chips since otherwise the divisor would not be q-reduced. The resulting T is by definition a tree since if edges (x, y) and (y, z) are added to T, that means $x, z \in X$ and thus the edge (x, z) is not handled and can never be added to T. We can extend this to a path with the same argument. □

Remark 5.7. Note that Algorithm 2 returns a spanning tree deterministically since we obey the ordering on E when picking the minimal edge, and q-reduced divisors represent their equivalence class uniquely.

Next we will formally show the inverse algorithm, which takes a vertex q to which the resulting divisor is

reduced, a spanning tree T as a base, and a degree d of the desired resulting divisor.

Algorithm 3: Spanning tree to reduced divisor

Input : $q \in V$ and spanning tree T and degree d
Output: q-reduced divsor D
$D := (0, ..., 0)$ `// Vector in` $\mathbb{Z}^n$
$H := \mathcal{N}(q)$ `// Set of unburned edges between burned and unburned vertices`
$B := \emptyset$ `// Set of burned edges`
$X := \{q\}$ `// Set of burned vertices`
while $|H| > 0$ **do**
  $(x, y) := \min\{e : e \in H\}$ `// Current edge`
  if $(x, y) \in B$ *or* $x, y \in X$ **then**
    `// In this case the edge has or the vertices have already been burned`
    Pass
  end
  `// Assume` $x \in X$, $y \notin X$
  if $(x, y) \in T$ **then**
    `//` y `burns through and we assign it a number of chips`
    $X \leftarrow X \cup \{y\}$
    $D_y := |\mathcal{N}^B(y)|$
    for $w \in \mathcal{N}(y)$ **do**
      $H \leftarrow H \cup \{(y, w)\}$
    end
  end
  $H \leftarrow H \setminus \{(x, y)\}$ `// Current edge processed, remove from` H
  $B \leftarrow B \cup \{(x, y)\}$ `// Add the edge to the burned edges`
end
$D_q = d - \sum_{v \in V \setminus \{q\}} D_v$ `// To guarantee the right degree`
Output D

Now to show that the divisor defined in Algorithm 3 is a indeed q-reduced divisor, we will first informally describe the proof. Since we are looking at all edges incident to burning vertices, there has to be at the very least one vertex that can not stop the incoming fire in the first iteration. Since we have an unweighted graph, this results in the fact that at least one vertex has 0 chips. We can now assume that the firefighters as mentioned before, are always enough to stop any incoming fires, except for when the burning edge is part of tree T. By stepping through the edges in the same ordering as in Algorithm 2 we obtain the same result.

Remark 5.8. We can look at Figure 12i in Example 5.4, and notice the aforementioned property. We see that the vertices with one chip on them, are the vertices with one 'incoming' burned edge that is not part of the resulting tree. The notion of 'incoming' only makes sense when we consider the order in which the algorithm burned the edges.

We will now formalize this argument based on the idea provided in Baker-Shokrieh [4].

Theorem 5.9. *Algorithm 3 returns a q-reduced divisor.*

Proof. At the moment any $v \in V$ burns through, we set $D_v = |\{\text{burned edges adjacent to } v\}| - 1$ since the edge that the fire is coming in on has not been added to the burned edges B yet. This means that following Algorithm 1 we know that v would burn when running Dhar's algorithm for checking for q-reduced divisors. Since this holds for any $v \in V \setminus \{q\}$, the entire graph will burn and thus the divisor is q-reduced. □

It is however more interesting to note that the ordering on E does indeed provide a bijection by means of Algorithm 2 and Algorithm 3:

Theorem 5.10. *Algorithm 2 and Algorithm 3 are each others inverse and thus provide a bijection between q-reduced graph divisors and spanning trees.*

Proof. This can be seen by the structure of the algorithm, since both traverse through the graph over the same edges as a result of using the same ordering on E and starting from the same vertex q.

Any time we add an edge to T in Algorithm 2, the condition under which this happens is explicitly enforced whenever an edge is part of T in Algorithm 3.

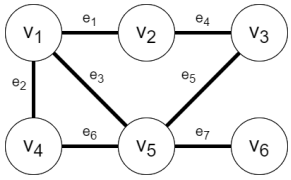
Vice versa the same holds: the condition under which we set the amount of chips on a vertex in Algorithm 3 is explicitly enforced by adding the edge to T in Algorithm 2. □

Lastly, we give an example of Algorithm 3 such that it is the inverse of Example 5.4.

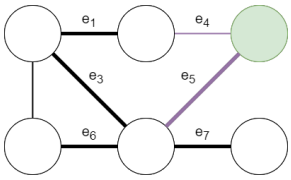
Example 5.11. We use the same graph as in Example 5.4. We wish to obtain a v_3-reduced divisor of degree six. In order to do so we start with the tree given by:

$$T = \{e_1, e_3, e_5, e_6, e_7\},$$

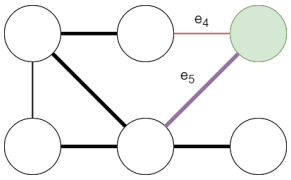
which we found in Example 5.4. Like in the other example, we show in green q, in purple edges marked for processing, and in red burned edges and vertices. We use thick edges to indicate that an edge is part of T.



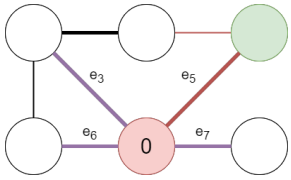
(a) Labelling of vertices and edges.



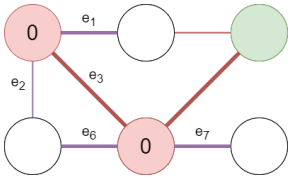
(b) Initial state with no chips on the graph.



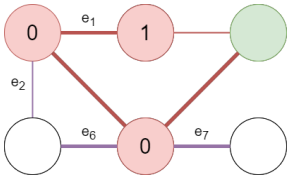
(c) Edge e_4 burns, but v_2 does not burn because $e_4 \notin T$.



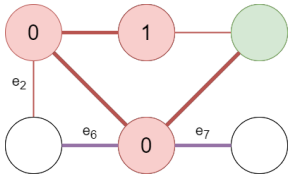
(d) Edge e_5 burns and zero chips placed on v_5 because v_5 burns.



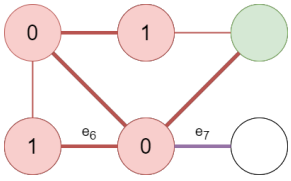
(e) Edge e_3 burns and zero chips placed on v_1 because v_1 burns.



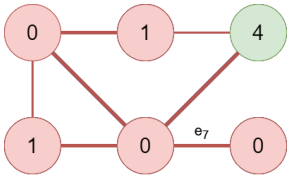
(f) Edge e_1 burns and one chip placed on v_2 because v_2 burns and e_4 already burned.



(g) Edge e_2 burns, but v_4 does not burn because $e_2 \notin T$.



(h) Edge e_6 burns and one chip is placed on v_4 because v_4 burns and e_2 already burned.



(i) Edge e_7 burns and zero chips are placed on v_6 because v_6 burns.

Figure 13: From spanning tree to v_3 reduced divisor.

In the last step in Figure 13i we place 4 chips on v_3, since at the end of all iterations we simply fill it up to satisfy the degree given as a parameter of the function. △

Remark 5.12. Note that in each image in Example 5.4 and Example 5.11 we handle the same edge exactly. We also end up at the end of Example 5.11 with the divisor we started Example 5.4 with, showing the bijection.

References

[1] Matthew Baker and Serguei Norine. "Riemann-Roch and Abel-Jacobi theory on a finite graph". In: *Advances in Mathematics* 215 (May 2007), pp. 766–788.

[2] Anders Björner, László Lovász, and Peter W. Shor. "Chip-firing Games on Graphs". In: *European Journal of Combinatorics* 12.4 (1991), pp. 283–291. ISSN: 0195-6698.

[3] Scott Corry and David Perkinson. *Divisors and Sandpiles*. 2018, pp. 20–55. ISBN: 978-1470442187.

[4] Matthew Baker and Farbod Shokrieh. "Chip-firing games, potential theory on graphs, and spanning trees". In: *Journal of Combinatorial Theory, Series A* 120.1 (Jan. 2013), pp. 164–182. ISSN: 0097-3165.

[5] David C. Lay, Steven R. Lay, and Judi J. McDonald. *Linear Algebra and its Applications (fifth edition)*. Pearson, p. 251. ISBN: 978-1-292-09223-2.

[6] D. Dhar. "Self-Organized Critical State of Sandpile Automaton Models". In: *The American Physical Society* 64.14 (Apr. 1990), pp. 1613–1616.