Vector Parking Functions and Tree Inversions

Petar Gaydarov

under the direction of
Mr. Samuel Hopkins,
Dr. Pavel Etingof
Math Department, MIT

Research Science Institute

Abstract

We find a depth-first search version of Dhar's burning algorithm that gives a bijection between the parking functions of a multigraph and its spanning trees. Thus we extend a result by Perkinson, Yang and Yu in response to a problem posed by Stanley. We also find another variant of this algorithm which gives a bijection between vector parking functions and labeled spanning trees closely related to the rooted planar trees. Both bijections have the goal of establishing a relation between the degree of a parking function, the κ-statistic for inversions, and the edge labelling of a tree. In addition, we find intriguing formulas for the number of vector parking functions in a special case of particular interest.

Summary

Parking functions were first introduced in this way: There are n parking spots ordered in a one-way street and n cars which want to park. Each car prefers a spot and drives until reaching it and parks there if it is empty. If it is already taken, the car parks in the next available parking spot down the street. If all the cars can park we have *parking function*.

Imagine now a garage with a different number of parking spots on each of its n levels and n cars which want to park. Again, each car has a preference. A car drives up the levels until the level of its spot and parks there. If a parking spot is already taken on that level, the car parks in the next empty level. Cars cannot go down. If all cars can park the sequence of their preferences is a *generalized parking function*.

Parking functions and generalized parking functions are related to a mathematical object called *spanning trees*. So far, only for parking functions, this connection gives a certain relation between the statistics of the two objects. In this paper we find a way to establish a similar relation for the generalized parking functions.

1 Introduction

Parking functions were first defined in terms of a problem related to parking cars by Konheim and Weiss [1]. Imagine that n parking spots labeled 0 through $n-1$ are arranged in order on a linear street. Cars $C_1, \ldots, C_n$ approach the spaces in ascending order starting at spot 0. Car C_i prefers spot a_i, which means that the car C_i will drive until reaching the spot a_i and park there if it is unoccupied. If the spot a_i is occupied C_i will continue driving until reaching an unoccupied spot and park in this unoccupied spot. If the spot a_i and all the spots after it are occupied, then the car C_i cannot park. A parking function of length n is a sequence $(a_1, \ldots, a_n)$ of parking preferences so that all the cars can park.

Despite the easy way of defining them parking functions have numerous applications in several areas of mathematics including the theory of diagonal harmonics [2], the study of hyperplane arrangements [3, 4, 5, 6] and graph theory.

An important result connecting graph theory objects and parking functions is derived from *Cayley's formula*. It states that the number of labeled spanning trees on $n+1$ vertices is $(n+1)^{n-1}$, which is the same as the number of parking functions of length n. Another connection between labeled spanning trees and parking functions was established by Kreweras [7]. He proved that there is a relation between the *degree* of parking functions, which is the sum of its entries, and the number of inversions in a labeled spanning tree.

Since Kreweras' method was not bijective Stanley [3] posed the problem of finding a bijective proof. Recently a bijective proof was found by Perkinson, Yang and Yu [8] and they generalized it for the setting of *graph parking functions*.

Graph parking function were first given that name by Postnikov and Shapiro [5] but have existed for longer under the names *superstable configurations* in the abelian sandpile model [9] and *q-reduced divisors* in discrete Riemann-Roch theory [10]. The bijection of Perkinson, Yang and Yu gives a relation between the degree of graph parking functions and the κ-*inversions* (a generalized inversion number defined by Gessel [11]) in the spanning trees for simple graphs. Their proof uses a depth-first search (DFS) variant of Dhar's burning algorithm.

Another generalization of parking functions goes under the names *generalized parking functions* or *vector parking functions*. The latter reflect the fact that vector parking functions depend on the choice of a nonnegative vector. In terms of parking, we can imagine vector parking functions as the preferences of cars parking in a leveled garage. Each level has a number of parking spots equal to the entries in the vector. Vector parking functions can also be defined in terms of horizontal strips [12, 13] and series of inequalities [14]. An important special case of vector parking functions is the *rational parking functions* [15, 16].

This paper deals with finding a bijection between vector parking functions and spanning trees. We aim to find a similar to Kreweras' connection between

the degree of a vector parking function and the number of inversions for spanning trees. To find such a bijection, it turns out that we first need to generalize the result of Perkinson, Yang and Yu to multigraphs.

To do so we extend the DFS-burning algorithm to multigraphs, thus providing a bijection between the set of G-parking functions for a given multigraph G and the set of its spanning trees. This bijection gives the following relation between the degree of a G-parking function $\deg \mathcal{P}$, the κ-statistic (a generalization we introduce of the κ-inversions for multigraphs) and a labelling of the edges $\omega(e)$ of a spanning tree T

$$\deg \mathcal{P} + \kappa(G, T) + \sum_{e \in T} \omega(e) = g, \tag{1}$$

where the genus is $g = |E| - |V| + 1$.

We aim to find a similar relation for vector parking functions. We construct the *DFS-vector burning algorithm* which gives a bijection between the set of $\mathbf{x}$-parking functions, where $\mathbf{x}$ is a vector, and a set of spanning trees closely related to the *rooted planar trees*. In Lemma 4.3 we prove that the results we obtain using this algorithm can also be interpreted as graph parking functions, thus, our first result still holds for vector parking functions.

In the Section 2 we provide the definitions for vector and graph parking functions. In Section 3 we examine the relations between G-parking functions and spanning trees of a multigraph G. We present the DFS-multiburning algorithm and prove that it establishes a bijection with a relation between the

degree and the κ-statistic. In section 4 we consider the $\mathbf{x}$-parking function and their relation to rooted planar trees. We first present a vertex labelling of the vertices of the rooted planar trees and the DFS-vector burning algorithm. Then, we prove Lemma 4.3 which is crucial for our further results as it links vector parking functions and graph parking functions. After that, we prove a relation between the degree of a vector parking function and the κ-inversions of a tree. We proceed by finding a formula for the number of vector parking functions in terms of rooted planar trees. Using this result we prove a formula by Pitman and Stanley [14] in terms of classical parking functions. In Appendix A we present basic facts about parking functions and graph theory. In Appendix B we list the DFS-burning algorithm of Perkinson, Yang and Yu [8]. The vectors $\mathbf{x} = (a, b, b, \ldots, b)$ and $\mathbf{x} = (a, \underbrace{0, \ldots, 0}_{k}, a, \underbrace{0, \ldots, 0}_{k}, a, \ldots, a, \underbrace{0, \ldots, 0}_{k-1})$ are two special cases of particular interest and we examine them in the Appendix C and D.

2 Preliminaries

In this section we introduce definitions used throughout the paper.

Definition 2.1. *An* $\mathbf{x}$*-parking function for a vector of nonnegative integers* $\mathbf{x} = (x_1, \ldots, x_n)$ *is a sequence of nonnegative integers* $(a_1, \ldots, a_n) \in \mathbb{N}^n$ *such that its weakly increasing rearrangement* $a_{i_1} \leq a_{i_2} \leq \cdots \leq a_{i_n}$ *satisfies* $a_{i_j} \leq \sum_{k=1}^{j} x_k - 1$.

We denote the set of $\mathbf{x}$-parking functions of length n by $\mathrm{PF}(\mathbf{x})$. The degree of an $\mathbf{x}$-parking function $\alpha = (a_1, \ldots, a_n)$ is the sum of its elements $\deg(\alpha) = \sum_{i=1}^{n} a_i$.

Perkinson, Perlman and Wilmes [9] use the theory of sandpiles to define a G-parking function for a connected sandpile graph $G = (V, E, v_0)$, where $V = \{v_0, v_1, \ldots, v_n\}$ is the set of vertices, E is the set of edges, and v_0 is a vertex called *sink*. Each edge has a weight denoted by $\mathrm{wt}(e)$. For simplicity we use these notations for every graph G unless it is specially stated otherwise. The graph G can be a multigraph with directed edges, but we only consider the case for undirected edges. For simplicity, when dealing with multigraphs, we will think of the weight $\mathrm{wt}(e)$ of an edge as $\mathrm{wt}(e)$ edges going between its endvertices.

Graph parking functions are closely related to *sandpile configurations*.

Definition 2.2. *A sandpile configuration* $c = \sum c_i v_i$ *on* G *is an element of* $\mathbb{Z}\widetilde{V}$*, the free Abelian group on* $\widetilde{V}$*, where* $\widetilde{V} = V/\{v_0\}$.

We can think of c as a pile of sand on nonsink vertices of G having c_i grains at vertex v_i for each $i = 1, \ldots, n$. Furthermore, we define an operation on the configurations.

We can see an example of a sandpile configuration in Figure 1. The vertex v_0 is the sink. Vertices v_1, v_2, and v_4 have 0 sand grains on them. The vertex v_3 has 4 sandgrains on it.

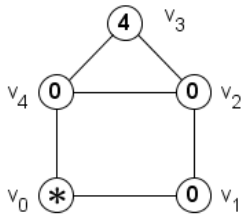

Figure 1: A sandpile configuration

Definition 2.3. *Vertex-firing from a vertex v_i is an operation on a configuration c of nonnegative integers which results in a new configuration $\widetilde{c}$ of nonnegative integers. It is given by $\widetilde{c_i} = c_i - \deg v_i$ and $\widetilde{c_j} = c_j + \mathrm{wt}(v_i v_j)$ for each positive $j \neq i$.*

When v_i *fires*, we imagine $\mathrm{wt}(e)$ grains of sand traveling along each edge e emanating from v_i and being deposited at its other end vertex. If that vertex is v_0, then sand sent along e disappears down the sink.

An equivalent definition is listed in the Appendix.

We also define a similar operation on c. Instead of firing a specific vertex, we can now *fire a set of vertices*.

Definition 2.4. *Set-firing is an operation on a configuration c such that we simultaneously perform vertex firing for a set of vertices as long as we obtain a configuration $\widetilde{c}$ of nonnegative integers.*

As one can see vertex firing is actually a special case of set-firing for sets of only one vertex. We call configurations on which set-firing cannot be performed *superstable*. The idea of a *G-parking function* is essentially the same as that of a superstable configuration.

Definition 2.5. *A G-parking function is a function $\mathcal{P} : \widetilde{V} \to \mathbb{Z}$ such that the configuration given by $c_i = \mathcal{P}(v_i)$ for each $i = 1, \ldots, n$ is superstable.*

In Figure 2 we see an example for a sandpile and set-firing. The vertex with the star is the sink v_0 and the numbers inside each nonsink vertex are the numbers of sand grains. The hatched vertices are the ones that are being fired. The last configuration is superstable because there is no set of vertices that can be fired.

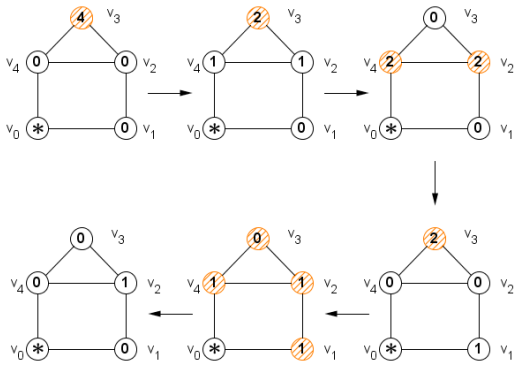

Figure 2: Set-firing and superstable configurations

3 DFS-Multiburning Algorithm

In this section we extend the DFS-burning algorithm (defined in the Appendix) to multigraphs and call it the *DFS-multiburning algorithm.*

We introduce a notation for the edge-labelling. Multiple edges between the same vertices are labeled from 0 to $\mathrm{wt}(e) - 1$ so that we can distinguish between them. The edge with number l between v_i and v_j is denoted by $(v_i v_j)_l$. We also introduce the *edge number function.*

Definition 3.1. *The edge number function $\omega : E \to \mathbb{Z}$ is defined the following way: for a given edge e_l the value of the function is $\omega(e_l) = l$.*

Given a G-parking function $\mathcal{P}$ on a finite multigraph G we obtain a spanning tree T of G by running the DFS-multiburning algorithm. At each step of the algorithm a vertex is either deleted or added to the tree. If the vertex is added to the tree then we say that the vertex is *burnt.*

DFS-multiburning algorithm

1: We start *burning* vertices from the sink v_0.

2: We are currently at vertex v_i.

3: If v_i is connected to an unburnt vertex we choose the unburnt vertex with the highest index connected to v_i and the edge with the highest number connecting the two vertices. Let these be v_j and $(v_i v_j)_l$.

4: If v_j has a positive number of grains we delete the edge $(v_i v_j)_l$

from the graph and remove 1 grain of sand from v_j. We go back to step 2 for v_i.

5: If v_j has no grains of sand we burn the edge $(v_iv_j)_l$ and the vertex v_j. Then, we go to step 2 for vertex v_j.

6: If v_i is not connected to an unburnt vertex, then we go to step 2 for the parent of v_i.

7: We stop when we cannot burn any more vertices.

The main difference between the DFS-burning algorithm and the DFS-multiburning algorithm is that we consider multiple edges and give priority to the edge assigned the highest index.

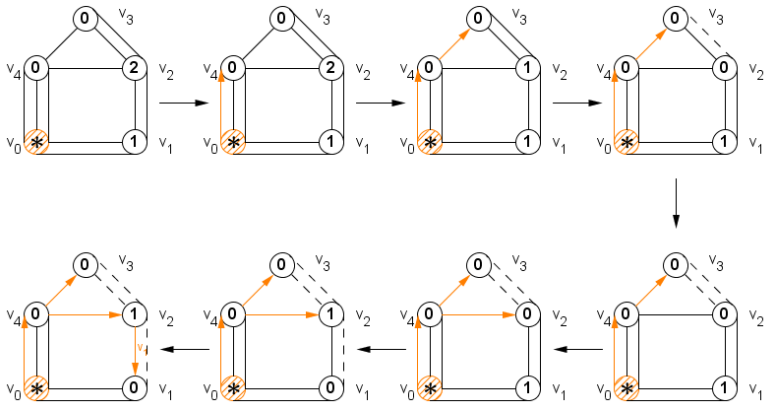

Figure 3: DFS-multiburning algorithm

We can see the DFS-multiburning algorithm working in Figure 3. At each step we either burn or delete one of the edges of the graph. The vertex with the star is the sink v_0 and the numbers inside each nonsink vertex are the

numbers of sand grains. The orange vertices and edges are already burnt. The dashed edges are already deleted.

We also present the inverse DFS-multiburning algorithm, which is a modified version of the inverse of the normal DFS-burning algorithm [8]. We start with a spanning tree T of G and obtain a G-parking function $\mathcal{P}$.

Inverse DFS-multiburning algorithm

1: We start burning vertices from the sink v_0 with $\mathcal{P}(v_i) = (0)$ for $i = 1, \ldots, n$.

2: We are currently at vertex v_i.

3: If v_i is connected in G to an unburnt vertex we check the vertex with the highest index v_j and the edge $(v_iv_j)_l$ with the highest number between v_i and v_j.

4: If $(v_iv_j)_l$ is not in T we add 1 to $\mathcal{P}(v_j)$, delete the edge $(v_iv_j)_l$ from G and go to step 2 for v_i.

5: If $(v_iv_j)_l$ is in T we burn v_j and the edge $(v_iv_j)_l$ and go to step 2 for v_j.

6: If v_i is not connected in G to an unburnt vertex we go to step 2 for the parent of v_i.

7: We stop when we cannot burn any more vertices.

It is straightforward to see that this algorithm is inverse to the DFS-multiburning algorithm.

We prove that the DFS-multiburning algorithm describes a bijection between the set of G-parking functions $\mathrm{PF}(G)$ and the set of spanning trees $\mathrm{SPT}(G)$ of G.

Theorem 3.2. *The DFS-multiburning algorithm induces a bijection* $\phi_{\mathrm{multi}} : PF(G) \to SPT(G)$ *between the set of* G*-parking functions and the set of spanning trees of* G*.*

Proof. We need to prove that ϕ_{multi} is injective, surjective and well-defined.

When we run the algorithm it only burns edges between a burnt and an unburnt vertex, hence, there cannot be a cycle of burnt vertices — so we obtain a tree.

Suppose that the result of the algorithm is not a spanning tree, which means that not all of the vertices are burnt. Let S be the set of all l vertices which are not burnt. Without loss of generality, let those vertices be $v_1, \ldots, v_l$. The edges from the vertices in S to the vertices in G/S are not burnt. However, all the vertices in G/S are burnt, so these edges are deleted. This means that each vertex has at least as many sand grains, so $a_i \geq \deg_{G/S}(v_i)$ for each $i = 1, \ldots, l$, where $\deg_G(v_i)$ denotes the number of edges between a vertex v_i and the vertices in the graph G. But this means the set S can be fired, so the configuration is not superstable — a contradiction.

Suppose that ϕ_{multi} is not surjective. This means that when we use the inverse of the DFS-multiburning algorithm we do not obtain a superstable configuration. This means that there is a set S that can be fired. Let v_s be the

first burnt vertex from S. Then its degree for G/S is equal to $a_s + 1$, because we have deleted a_s edges and burnt one more. But then $\deg_{G/S} v_s > a_s$, so S cannot be fired — a contradiction.

Suppose ϕ_{multi} is not injective. Then there are two G-parking functions $\mathcal{P}_1$ and $\mathcal{P}_2$ that correspond to the same spanning tree. First, assume that the DFS-multiburning algorithm follows the same steps for both G-parking functions. We prove by induction on the number of burnt vertices that then $\mathcal{P}_1(v_i) = \mathcal{P}_2(v_i)$ for each $i = 1, \ldots, n$.

Base case. Perkinson, Wilmes and Perlman define the sink to have -1 sand grains [9]. We always start from the sink v_0, so $\mathcal{P}_1(v_0) = \mathcal{P}_2(v_0) = -1$.

Assumption. By w_i we denote the i^{th} burnt vertex. We assume that $\mathcal{P}_1(w_i) = \mathcal{P}_2(w_i)$ for each $i = 1, \ldots, t$.

Indunctive step. We now prove that $\mathcal{P}_1(w_{t+1}) = \mathcal{P}_2(w_{t+1})$. We have deleted the same number of edges between burnt vertices and w_{t+1} for both $\mathcal{P}_1$ and $\mathcal{P}_2$. Since we choose the same edge, the edges we delete on this step are also the same. But this means that the number of sand grains on w_{t+1} is the same for $\mathcal{P}_1$ and $\mathcal{P}_2$. This completes the last induction step.

So if $\mathcal{P}_1$ and $\mathcal{P}_2$ are different, then the order in which we burn the edges and vertices is different. Suppose that with different order of burning we still have the same spanning trees. Let the edge e be the first edge in which the DFS-multiburning algorithm differs for $\mathcal{P}_1$ and $\mathcal{P}_2$. Since e is the first such edge, all previous actions are the same. Therefore the algorithms for both G-parking functions will perform an action with e. However, there are only

two options — it can be burnt or deleted. Without loss of generality let e be burnt for $\mathcal{P}_1$ and deleted for $\mathcal{P}_2$. But then the edge can no longer be burnt for $\mathcal{P}_2$ so it cannot be the same spanning tree — a contradiction. □

The DFS-multiburning algorithm induces a relation between inversions, edge weights and degree of a parking function. We extend the definition of a κ-inversions to a multigraph.

Definition 3.3. *The κ-statistic is equal to*

$$\kappa(G,T) = \sum_{(v_i,v_j)} wt(v_i v_l),$$

where (v_i, v_j) is an inversion and v_l is v_j's parent.

The DFS-multiburning algorithm induces a relation between the degree of a G-parking function and the κ-statistic. It is closely related to Kreweras' results [7].

Theorem 3.4. *Let $\mathcal{P}$ be a G-parking function and $T = \phi_{\text{multi}}(\mathcal{P})$ the corresponding tree. Then*

$$\deg \mathcal{P} + \kappa(G,T) + \sum_{e \in T} \omega(e) = g, \tag{2}$$

where $g = |E| - |V| + 1$ is the genus of the graph G.

Proof. To prove Theorem 3.4 we count the number of edges. The genus

g is equal to $|E| - |V| + 1$, so we can rewrite (2) in the following way

$$\deg \mathcal{P} + \sum_{(v_i, v_j)_{v_l}} \mathrm{wt}(v_i v_l) + \sum_{e \in T} \omega(e) + |V| - 1 = |E|.$$

The number of burnt edges equals $|V| - 1$ since the spanning tree is on $|V|$ vertices.

The number of deleted edges equals $\deg \mathcal{P}$ since we delete one edge for each sand grain.

The sum over all the labellings of the edges in the spanning tree is the number of edges which were not deleted, because another edge between the same two end vertices was burnt. This is true since the DFS-multiburning algorithm always goes to the edge with the highest index.

The κ-statistic is the sum over all the inversions. This represents the number of edges which were not deleted because the DFS-multiburning algorithm always goes to the vertex with the highest index. Let (v_i, v_j) be an inversion and v_l be v_j's parent. Then, when we burnt the vertex v_j, we did not burn the edges between v_i and v_l because $j > i$. In case we come back to v_l at one point to run the DFS-multiburning algorithm, we no longer consider the edges between v_l and v_i because v_i is already burnt. So we do not delete or burn these edges. Thus, each such inversion corresponds to $\mathrm{wt}(v_i v_l)$ edges. We use the same reasoning to see that for every undeleted weighted edge, there is an inversion in the tree.

These are the four types of edges — burnt, deleted, undeleted with a burnt

edge between its end vertices and undeleted without a burnt edge between its end vertices. Therefore, when we sum them, we obtain the total number of edges. □

Hence, for a parking function $\mathcal{P}$ of any graph G the DFS-multiburning algorithm gives a corresponding spanning tree $\phi_{\text{multi}}(\mathcal{P})$ of G. Given the graph G and the corresponding spanning tree, we can also find the degree of a parking function $\mathcal{P}$ using equation (2).

We can also express Theorem 3.4 in terms of polynomials. To do so, we need to define a few notations. The codegree of a G-parking function $\mathcal{P}$ is $\text{codeg}(\mathcal{P}) = g - \deg(\mathcal{P})$. We also use the notation $[a]_q = 1 + q + q^2 + \cdots + q^{a-1}$.

Corollary 3.5. *For a given multigraph G the following equality holds true*

$$\sum_{\mathcal{P} \in \text{PF}(G)} q^{\text{codeg}(\mathcal{P})} = \sum_{T \in \text{SPT}(G^*)} \left(q^{\kappa(G,T)} \sum_{e \in T} [\text{wt}_G(e)]_q \right),$$

where G^* has the same vertex and edge set as G but the weight of all its edges equals 1.

Proof. The sum over the spanning trees of G^* is simply an easier way to write down the spanning trees of G. By definition the codegree is the degree subtracted from the genus. The rest follows directly from Theorem 3.4. □

4 DFS-Vector Burning Algorithm

After finding a connection between G-parking functions and spanning trees our goal is to examine the $\mathbf{x}$-parking functions. Here we present a bijection between the set of $\mathbf{x}$-parking functions for a given nonnegative vector $\mathbf{x}$ and the set of specific type of spanning trees related to the *rooted planar trees*.

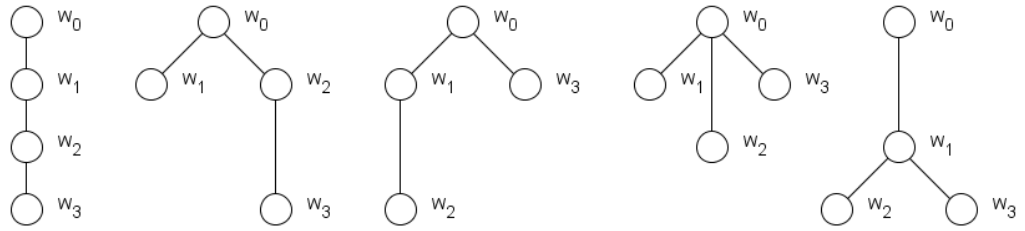

Figure 4: Rooted planar trees on 4 vertices

Definition 4.1. *Rooted planar trees are trees on the vertex set* $\{w_0, w_1, \ldots, w_n\}$ *with root* w_0 *such that there do not exist vertices* w_h, w_i, w_j *and* w_k *with* $h < i < j < k$ *such that both* $w_h w_k$ *and* $w_i w_j$ *are edges, and such that vertices in every path from the root are in an increasing order.*

One can think about the rooted planar trees as trees embedded in the plane. The method of labelling the vertices may seem strange at first but it represents the way we choose the vertices for the DFS-burning algorithm. Let denote the set of rooted planar trees on n vertices by RTP(n). In Figure 4 we present the rooted planar trees on 4 vertices with w-labelling.

We introduce a labelling of the vertices of the rooted planar trees. The vertex w_0 is now labeled (w_0, v_0). Each vertex w_i is now labeled $(w_i, v_{\pi(i)})$ for

any permutation π of the first n integers obeying *Rule 1*: If w_i and w_j have the same parent and $i < j$, then $\pi(i) > \pi(j)$. An example of this labelling is given in Figure 5.

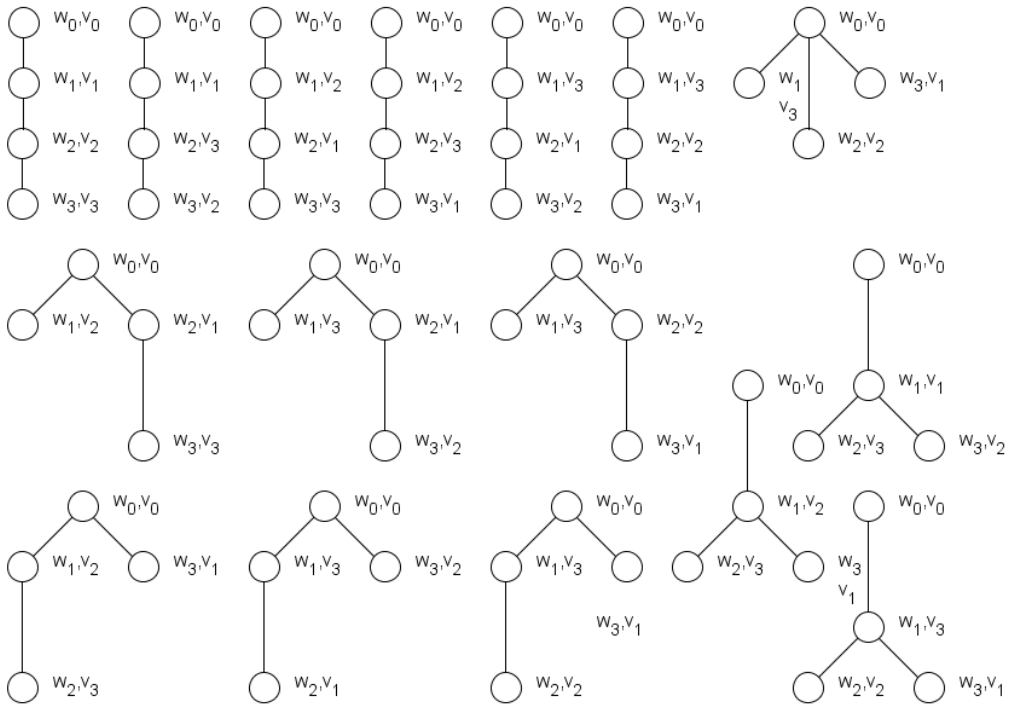

Figure 5: Rooted planar trees with (w, v)-labelling

When we consider only the first part of the pair $(w_i, v_{\pi(i)})$ for all vertices of a graph, we will refer to it as w-labelling of the graph and a w-labeled graph. When we only consider the second part of the pair $(w_i, v_{\pi(i)})$ for all the vertices of a graph, we will refer to it as v-labelling of the graph and a v-labeled graph. We can see an example of w-labelling in Figure 4 and an example of v-labelling in Figure 6.

After labelling the vertices we make multiple edges between the already

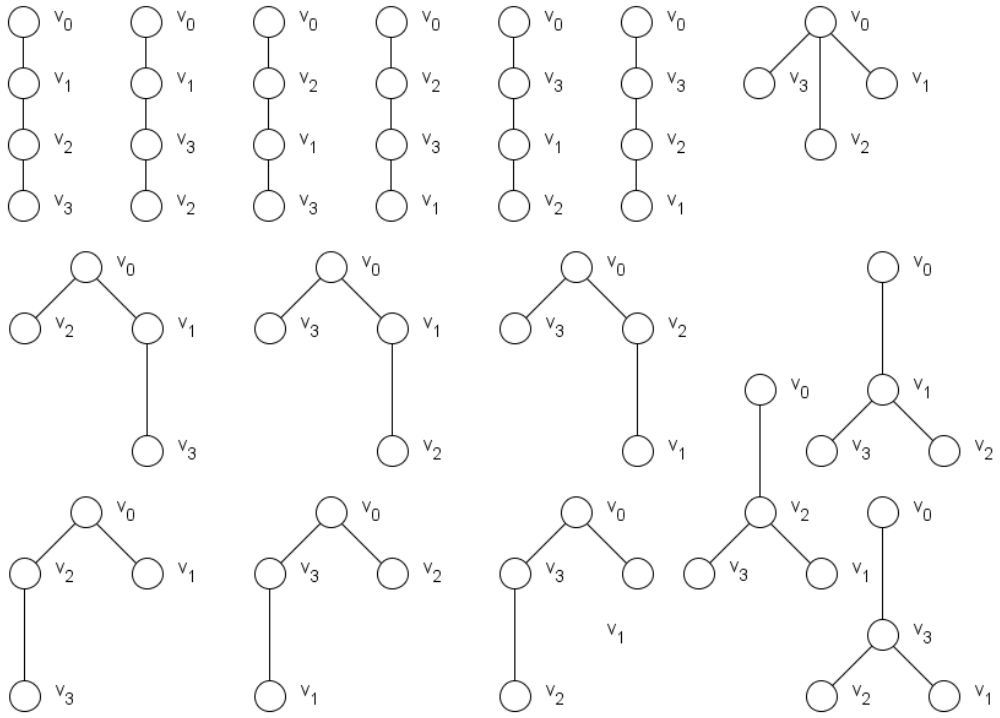

Figure 6: v-labeled rooted planar trees on 4 vertices

connected vertices. Let w_iw_j be an edge in the tree T. Then, the weight of the edge equals $x_{\min(i,j)+1}$. The edges are labeled from 0 to $x_{\min(i,j)+1} - 1$. We choose one edge of each set of multiple edges between the same vertices and thus we obtain trees with labeled edges.

Definition 4.2. *The* $\mathbf{x}$*-graph* $G_{\mathbf{x}} = (V_{\mathbf{x}}, E_{\mathbf{x}})$ *has the vertex set* $V_{\mathbf{x}} = \{w_0, w_1, \ldots, w_n\}$ *and edges* w_iw_j *with weight* $x_{\min(i,j)+1}$.

The spanning trees of the graph $G_{\mathbf{x}}$ which are labeled according to the rules for the w and v labellings and the trees with labeled edges are essentially the same concepts. We denote the set of such labeled spanning trees of $G_{\mathbf{x}}$

by $\mathrm{LST}(G_{\mathbf{x}})$.

We prove that there is a bijection between $\mathrm{LST}(G_{\mathbf{x}})$, the set of labeled spanning trees of $G_{\mathbf{x}}$ and the set of $\mathbf{x}$-parking functions.

But first we need to introduce our algorithm for $\mathbf{x}$-parking functions — the DFS-vector burning algorithm.

DFS-vector burning algorithm

for an $\mathbf{x}$-parking function $\alpha = (a_1, a_2, \ldots, a_n)$, where $\mathbf{x} = (x_1, \ldots, x_n)$

1: We start with a set of vertices $V = \{v_0, v_1, \ldots, v_n\}$ and no edges between them.

2: We assign a_i grains of sand to the vertex v_i. The vertex v_0 is the sink.

3: We start burning vertices from the sink v_0. We label it w_0.

4: We are currently at vertex v_i.

5: If v_i is the s^{th} burnt vertex we label this vertex w_s.

6: We construct edges towards each unburnt vertex with weight x_{s+1}.

7: If v_i is connected to an unburnt vertex we choose the unburnt vertex with the highest index connected to v_i and the edge with the highest number connecting the two vertices. Let these be v_j and $(v_iv_j)_l$.

8: If v_j has a positive number of grains we delete the edge $(v_iv_j)_l$ from the graph and remove 1 grain of sand from v_j. We go back to step 2 for v_i.

9: If v_j has no grains of sand we burn the edge $(v_iv_j)_l$ and the vertex

v_j. Then, we go to step 2 for vertex v_j.

10: If v_i is not connected to an unburnt vertex, then we go to step 2 for the parent of v_i.

11: We stop when we cannot burn any more vertices.

The main difference between the DFS-multiburning algorithm and the DFS-vector burning algorithm is that we simultaneously construct the graph and its spanning tree. We also have another type of labelling which corresponds to the order in which vertices are burnt.

We can see the DFS-vector burning algorithm working in Figure 7. At each step we create edges or burn or delete the already created edges of the graph. The vertex with the star is the sink v_0 and the numbers inside each nonsink vertex are the numbers of sand grains. The orange vertices and edges are already burnt. The dashed edges are already deleted.

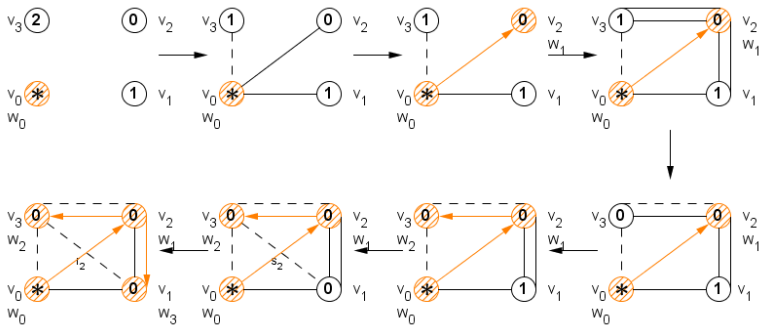

Figure 7: DFS-vector burning algorithm for $\mathbf{x} = (1, 2, 1)$

When we consider the w-labelling we have only one graph we examine

and a well-defined set of its spanning trees. However, we run the DFS-vector burning algorithm with respect to the v-labelling and hence the results about the degree and the κ-statistic are not true for the w-labelling. When we consider the v-labelling we obtain a wide variety of multigraphs and only a part of their spanning trees correspond to the $\mathbf{x}$-parking functions. However, since we run the DFS-vector burning algorithm with respect to the v-labelling, the results proven for the G-parking functions still hold for the v-labeled graph we obtain. Let G_α denote the v-labeled graph obtained by the algorithm for an $\mathbf{x}$-parking function α.

Lemma 4.3. *For an $\mathbf{x}$-parking function $\alpha = (a_1, \ldots, a_n)$ and the v-labeled graph G_α the function $\mathcal{P} : \widetilde{V} \to \mathbb{Z}$ given by $\mathcal{P}(v_i) = a_i$ for all nonsink vertices v_i of G is a G-parking function.*

Proof. As in Theorem 3.2 it is easy to see that we obtain a tree.

Suppose that the last vertex the algorithm burns is w_t (we still consider only the v-labelling but use the w-labelling so that we can more easily refer to the t^{th} burn vertex). This means that all edges are either burnt or deleted. The number of all edges connecting a burnt vertex to the unburnt vertices is $\sum_{i=0}^{t} x_{i+1}$. But if all these edges are deleted, this means that the values of α which correspond to the unburnt vertices are all greater or equal to $\sum_{i=0}^{t} x_{i+1}$. However, the inequalities for the $\mathbf{x}$-parking functions state that there must be $t+1$ values for which $a_i < \sum_{i=0}^{t} x_{i+1}$ — a contradiction. Therefore, all the vertices are burnt.

Now suppose that we do not obtain a superstable configuration. This means that there is a set S with l vertices that can be fired. Let v_s be the first vertex from S to be burnt using the DFS-vector burning algorithm. Then the number of sandgrains on it is $a_s = \deg_{G/S}(v_s) - 1$ because we delete a_s edges and burn 1. Therefore, the set S cannot be fired — a contradiction. □

Lemma 4.3 gives us the chance to consider the $\mathbf{x}$-parking functions as G-parking functions. Therefore, the statement of Theorem 3.4 holds for the $\mathbf{x}$-parking functions as well.

We also present the inverse of the DFS-vector burning algorithm.

Inverse of the DFS-vector burning algorithm for an $\mathbf{x}$-parking function

---

1: We start with a labeled spanning tree of $G_{\mathbf{x}}$.

2: We run the inverse of the DFS-multiburning algorithm with respect to the v-labelling.

Theorem 4.4. *The DFS-vector burning algorithm induces a bijection* $\Phi : PF(\mathbf{x}) \to LST(G_{\mathbf{x}})$ *between the set of* $\mathbf{x}$*-parking functions and the set of labeled spanning trees of* $G_{\mathbf{x}}$.

Proof. We need to prove that the function Φ is surjective, injective and well-defined.

From Lemma 4.3 and the proof of Theorem 3.2 it follows that the codomain of Φ includes only spanning trees of $G_{\mathbf{x}}$.

Since the DFS-vector burning algorithm first goes to the vertex with the highest index and the w-labelling corresponds to the order in which the vertices are burnt, the condition is satisfied. Therefore, the function is well-defined.

From Lemma 4.3 and the proof of Theorem 3.2 it follows that the function is injective as well.

Suppose Φ is not surjective. Then, when we run the inverse of the DFS-vector burning algorithm on a labeled spanning tree we obtain a sequence of numbers which is not an $\mathbf{x}$-parking function. The inequalities for the $\mathbf{x}$-parking functions imply that there are l numbers is the sequence we obtained, which are greater or equal to $\sum_{i=1}^{l} x_i$. Now we run the DFS-vector burning algorithm for the sequence we obtained. The condition means that there is a set S of l vertices having at least $\sum_{i=1}^{l} x_i$ sand grains on each of them. The number of edges between the vertices in G/S and every vertex in S is exaclty $\sum_{i=1}^{l} x_i$, which means that these vertices will not be burnt. However we started with a spanning tree — a contradiction. □

Theorem 4.5. *For a given nonnegative vector* $\mathbf{x}$ *the following equality holds true*

$$\sum_{\alpha \in \mathrm{PF}(\mathbf{x})} q^{\mathrm{codeg}(\alpha)} = \sum_{T \in \mathrm{RPT}(n+1)} \left(\left(\sum_{e \in T} [\mathrm{wt}(e)]_q \right) \left(\sum_{l(G^*_{\mathbf{x}})} q^{\kappa(T, l(G^*_{\mathbf{x}}))} \right) \right), \quad (3)$$

where the last sum is over all the v-labellings $l(G^*_{\mathbf{x}})$ of the graph $G_{\mathbf{x}}$.

Proof. Lemma 4.3 allows us to use Corollary 3.5. According to it for a

given v-labelling $l(G^*_{\mathbf{x}})$ of a given rooted planar tree we have

$$\sum_{\alpha} q^{\mathrm{codeg}(\alpha)} = q^{\kappa(T,l(G^*_{\mathbf{x}}))} \sum_{e \in T} [\mathrm{wt}(e)]_q$$

for all α which correspond to the given spanning tree of $l(G^*_{\mathbf{x}})$ with the given v-labelling. We notice that in Corollary 3.5 the sum is over *all* of the parking functions which correspond to all of the spanning trees of G^*. However, as seen from the proof of Theorem 3.4 we have equality for the sum, because we have equality for every element of the sum.

Now we sum over all the labellings of the graph $G_{\mathbf{x}}$. We obtain

$$\sum_{\alpha} q^{\mathrm{codeg}(\alpha)} = \sum_{l(G^*_{\mathbf{x}})} \left(q^{\kappa(T,l(G^*_{\mathbf{x}}))} \sum_{e \in T} [\mathrm{wt}(e)]_q \right) = \left(\sum_{e \in T} [\mathrm{wt}(e)]_q \right) \left(\sum_{l(G^*_{\mathbf{x}})} q^{\kappa(T,l(G^*_{\mathbf{x}}))} \right)$$

for all α which correspond to the given spanning tree. Now we sum over all the rooted planar trees and obtain the desired result. □

Using (3) we also count the number of $\mathbf{x}$-parking functions.

Corollary 4.6. *The number of* $\mathbf{x}$*-parking functions for* $\mathbf{x} = (x_1, \ldots, x_n)$ *expressed in terms of* w*-labeled rooted planar trees is*

$$n! \sum_{T \in RPT(n+1)} \left(\prod_{i=0}^{n-1} \left(\frac{x_{i+1}^{ddeg_T(w_i)}}{ddeg_T(w_i)!} \right) \right),$$

where the downdegree function $\mathrm{ddeg}_T(w_i)$ *gives the number of edges not connecting* w_i *to its parent in* T*.*

Proof. We input $q = 1$ in Theorem 4.5 and obtain

$$\#\mathbf{x}\text{-parking functions} = \sum_{T \in \mathrm{RPT}(n+1)} \left(\prod_{e \in T} \Big(\mathrm{wt}(e) \Big) \Big(\#\ v\text{-labellings of } G_{\mathbf{x}} \Big) \right).$$

For a given tree T the weight of its edges can be expressed in terms of the elements of $\mathbf{x}$. For a vertex w_i the number of edges with weight x_{i+1} is equal to the downdegree $\mathrm{ddeg}(w_i)$. Therefore, we obtain the formula

$$\#\mathbf{x}\text{-parking functions} = \sum_{T \in \mathrm{RPT}(n+1)} \left(\prod_{i=0}^{n-1} \Big(x_{i+1}^{\mathrm{ddeg}_T(w_i)} \Big) \Big(\#\ v\text{-labellings of } G_{\mathbf{x}} \Big) \right). \tag{4}$$

If we do not consider Rule 1 the number of v-labellings for a given tree T is the number of permutations which is $n!$. If w_i has $\mathrm{ddeg}(w_i)$ children, they can only be labeled in one way due to Rule 1. This means the number of ways is reduced $\mathrm{ddeg}(w_i)!$ times for each w_i. We substitute this in (4) and obtain

$$\#\mathbf{x}\text{-parking functions} = \sum_{T \in \mathrm{RPT}(n+1)} \left(\prod_{i=0}^{n-1} \Big(x_{i+1}^{\mathrm{ddeg}_T(w_i)} \Big) \prod_{i=0}^{n-1} \Big(\mathrm{ddeg}_T(w_i)! \Big) \right),$$

which is the desired result. □

Pitman and Stanley [14] derive a different formula for the number of $\mathbf{x}$-parking functions dependent on the classical parking functions and the values of x_i.

Theorem 4.7. *(Pitman and Stanley [14]) The number of* $\mathbf{x}$*-parking functions*

for a nonnegative vector $\mathbf{x} = (x_1, \ldots, x_n)$ *is*

$$\sum_{(a_1,\ldots,a_n)} x_{a_1+1} \ldots x_{a_n+1}$$

summed over all classical parking functions $(a_1, \ldots, a_n)$.

We prove that Corollary 4.6 and Theorem 4.7 give the same results. Before we present our proof of Theorem 4.7 we define two functions.

Definition 4.8. *For a* w*-labelling of a tree the parent function* $p_w : V \to \mathbb{Z}$ *for a nonroot vertex* v_i *is the index of its parent in the* w*-labelling.*

Definition 4.9. *The function* $p : \mathrm{LST}(G_{(1,\ldots,1)}) \to \mathbb{Z}^n$ *is given by the following: when the parent function gives values* $p_w(v_i) = a_i$ *for* $i = 1, \ldots, n$*, the function* p *is given by* $p(T) = (a_1, \ldots, a_n)$.

The following Lemma is crucial for the proof of Theorem 4.7.

Lemma 4.10. *The function* $p : \mathrm{LST}(G_{(1,\ldots,1)}) \to \mathbb{Z}^n$ *gives a bijection between the set of labeled rooted planar trees on* $n+1$ *vertices without weighted edges and the set of parking functions of length* n.

Proof. It is clear that p is injective. We first prove that p is well-defined. The vertex w_1 is connected to w_0. Therefore, $p_w(v_{\pi(1)}) = 0$. The vertex w_2 is connected to either w_0 or w_1. Therefore, $p_w(v_{\pi(2)}) \leq 1$. The vertex w_3 is connected to one of w_0, w_1, or w_2. Therefore, $p_w(v_{\pi(3)}) \leq 2$. We continue by induction to show that $p_w(v_{\pi(n)}) \leq n - 1$. The function is well-defined.

It is clear that given a parking function, we can construct a tree corresponding to it. We start from w_0. We construct all the vertices for which $p_w(v_i) = 0$. Since they have the same parent, there is only one possible w-labelling due to Rule 1. Then we choose one of the vertices w_k we obtained and construct all the vertices for which $p_w(v_i) = k$. We continue by induction. Hence, p is surjective as well. □

Corollary 4.11. *The bijection* $p : \mathrm{LST}(G_{(1,\ldots,1)}) \to \mathbb{Z}^n$ *gives the relation* $\mathrm{ddeg}(w_{i-1}) = m(a_i)$ *for all* $i = 1, \ldots, n$, *where* $m(a_i)$ *denotes the number of appearances of* a_i *in the parking function* α. □

Proof of Theorem 4.7. We consider a nonnegative vector $\mathbf{x} = (x_1, \ldots, x_n)$. We take the labeled spanning trees of $G_{\mathbf{x}}$. We put them at different classes using the following rule: if two trees are the same labeled rooted planar trees without considering the labelling of the edges we put them in the same class. To count the number of $\mathbf{x}$-parking functions we need to find the number of labeled spanning trees in each class. Because different edges between the same vertices give different trees, for each class the number of labeled spanning trees increases $\mathrm{wt}_{G_{\mathbf{x}}}(e)$ times for each edge. But as stated in the proof of Corollary 4.6 this is exactly $x_{i+1}^{\mathrm{ddeg}(w_i)}$. We need to express $\mathrm{ddeg}(w_i)$ in terms of the classical parking functions.

By Lemma 4.10 we know that each value of function p corresponds to a classical parking function. We also know that the bijection p gives the relation $\mathrm{ddeg}(w_{i-1}) = m(a_i)$ by Corollary 4.11. It is straightforward to see that an alternative way to write the number of $\mathbf{x}$-parking functions corresponding to

a classical parking function is

$$x_{a_1+1} \dots x_{a_n+1},$$

by using the relation in Corollary 4.11. When we sum over all the classical parking functions we obtain the desired result. □

5 Conclusion

We have examined the connections between vector parking functions, graph parking functions and spanning trees satisfying a certain relation between the degree of a parking function and the κ-statistic of a tree, a generalized inversion number. In the first part of the paper we extended the results of Perkinson, Yang and Yu [8] to multigraphs as we found a bijection between the set of graph parking functions and the set of spanning trees of this graph. In the second part of the paper we found a bijection between the set of vector parking functions and the set of labeled spanning trees of a specific graph. This bijection is crucial in finding a relation between the degree of a vector parking function and the κ-statistic of the trees we examine. We also found a formula in terms of rooted planar trees for the number of vector parking functions. Using this formula we proved another result about the number of vector parking functions by Pitman and Stanley [14]. In the Appendix we focused on some special cases of vector parking functions which are of

particular interest. These were the vector parking functions for the vectors $\mathbf{x} = (a, b, b, \ldots, b)$ and $\mathbf{x} = (a, \underbrace{0, \ldots, 0}_{k}, a, \underbrace{0, \ldots, 0}_{k}, a, \ldots, a, \underbrace{0, \ldots, 0}_{k-1})$.

6 Acknowledgments

I would like to thank my mentors Mr. Sam Hopkins and Dr. Pavel Etingof for guiding me during my work on this paper. I would like to express my gratitude towards my tutor Dr. John Rickert. I am also thankful for the direction Dr. Tanya Khovanova has given me. I would like to thank Dr. David Jerison. I would like express my gratitude towards Jenny Sendova, Adam Sealfon, Bennett Amodio, Kati Velcheva, Stanislav Atanasov, Kalina Petrova, Antoni Rangachev and Konstantin Delchev for the helpful discussions about the paper. I am also thankful to Sts Cyril and Methodius Foundation whose sponsorship allowed me to participate in RSI. I would also like to express my gratitude towards RSI, MIT and CEE.

References

[1] A. G. Konheim and B. Weiss. An occupancy discipline and applications. *SIAM Journal on Applied Mathematics*, 14(6):1266–1274, 1966.

[2] J. Haglund. The q, t-catalan numbers and the space of diagonal harmonics. *University Lecture Series*, 41, 2008.

[3] R. P. Stanley. An introduction to hyperplane arrangements. In *Lecture notes, IAS/Park City Mathematics Institute*. Citeseer, 2004.

[4] S. Hopkins and D. Perkinson. Bigraphical arrangements. *arXiv preprint arXiv:1212.4398*, 2012.

[5] S. Hopkins and D. Perkinson. Orientations, semiorders, arrangements, and parking functions. *arXiv preprint arXiv:1112.5421*, 2011.

[6] M. Beck, A. Berrizbeitia, M. Dairyko, C. Rodriguez, A. Ruiz, and S. Veeneman. Parking functions, shi arrangements, and mixed graphs. *arXiv preprint arXiv:1405.5587*, 2014.

[7] R. P. Stanley. Parking functions. *http://www-math.mit.edu/ rstan/trans.html*, 1999.

[8] D. Perkinson, Q. Yang, and K. Yu. G-parking functions and tree inversions. *arXiv preprint arXiv:1309.2201*, 2013.

[9] D. Perkinson, J. Perlman, and J. Wilmes. Primer for the algebraic geometry of sandpiles. Technical report, 2011.

[10] M. Baker and S. Norine. Riemann–roch and abel–jacobi theory on a finite graph. *Advances in Mathematics*, 215(2):766–788, 2007.

[11] I. M. Gessel. Enumerative applications of a decomposition for graphs and digraphs. *Discrete mathematics*, 139(1):257–271, 1995.

[12] D. Chebikin and A. Postnikov. Generalized parking functions, descent numbers, and chain polytopes of ribbon posets. *Advances in Applied Mathematics*, 44(2):145–154, 2010.

[13] D. Armstrong and S.-P. Eu. Nonhomogeneous parking functions and noncrossing partitions. *the electronic journal of combinatorics*, 15(1):R146, 2008.

[14] R. P. Stanley and J. Pitman. A polytope related to empirical distributions, plane trees, parking functions, and the associahedron. *Discrete & Computational Geometry*, 27(4):603–602, 2002.

[15] D. Armstrong, N. A. Loehr, and G. S. Warrington. Rational parking functions and catalan numbers. *arXiv preprint arXiv:1403.1845*, 2014.

[16] E. Gorsky, M. Mazin, and M. Vazirani. Affine permutations and rational slope parking functions. *arXiv preprint arXiv:1403.0303*, 2014.

[17] C. H. Yan. Generalized parking functions, tree inversions, and multicolored graphs. *Advances in Applied Mathematics*, 27(2):641–670, 2001.

A Definitions

In this section we introduce definitions and basic facts used throughout the paper.

A.1 Classical Parking Functions

Definition A.1. *A classical parking function of length n is a sequence of nonnegative integers $(a_1, \ldots, a_n) \in \mathbb{N}^n$ such that its weakly increasing rearrangement $a_{i_1} \leq a_{i_2} \leq \cdots \leq a_{i_n}$ satisfies $a_{i_j} \leq j - 1$.*

We denote the set of classical parking functions of length n by $\mathrm{PF}(n)$.

When presented in this way, it seems natural to generalize parking function by changing some of the parameters in the inequalities.

A.2 Graph Parking functions

We use the definitions of Perkinson, Perlman and Wilmes [9] for the abelian sandpile model.

Definition A.2. *The Laplacian Δ of G for undirected graphs, which are the ones we examine, coincides with the standard Laplacian $L = D - A$ where D is the diagonal matrix $\mathrm{diag}(\deg(v_0), \ldots, \deg(v_n))$ and the adjacency matrix A is given by $A_{ij} = \mathrm{wt}(v_i v_j)$. The reduced Laplacian $\widetilde{\Delta}$ is obtained by deleting from the standard Laplacian the first row and column which correspond to the sink v_0.*

Definition A.3. *Vertex firing from a vertex v_i is an operation on a configuration c with a nonnegative number of sand grains which results in a new configuration $\widetilde{c} = c - \widetilde{\Delta} v_i$ with a nonnegative number number of sand grains.*

A.3 Spanning Trees

We use some basic graph theory definitions, namely *rooted spanning trees, inversions, κ-inversion, genus* and *degree* of a G-parking function.

Definition A.4. *A rooted spanning tree T of a connected, undirected graph G is a subgraph of G that includes all of the vertices and some or all of the edges of G, does not contain any cycles and one of its vertices is chosen as a root.*

Definition A.5. *An inversion of a rooted spanning tree T of the graph G is a pair of vertices (v_i, v_j), such that v_i is an ancestor of v_j in T and $i > j$. It is a κ-inversion if, in addition, v_i is not the root and v_i's parent is adjacent to v_j in G.*

We show the inversions and κ-inversions in a orange tree in Figure 8. The blue line shows that the parent of one of the vertices is connected to the other. The dashed blue line shows which two vertices should be connected so that we can have a κ-inversion.

The number of κ-inversions of T is the tree's κ-number, denoted $\kappa(G, T)$.

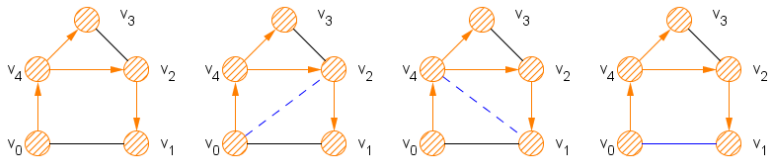

Figure 8: Inversions: (v_4, v_2), (v_2, v_1), (v_4, v_1)
κ-inversions: (v_4, v_1)

Definition A.6. *The degree of a G-parking function $\mathcal{P}$ is*

$$\deg(\mathcal{P}) = \sum_{i=1}^{n} \mathcal{P}(v_i). \tag{5}$$

Definition A.7. *The genus of a graph $G = (V, E)$ is defined as $g = |E| - |V| + 1$.*

Perkinson, Yang and Yu [8] present an algorithm which induces a bijection between the set of G-parking functions for a given simple graph G and the set of the spanning trees of G. The bijection ϕ of Perkinson, Yang and Yu induces the following relation for a G-parking function $\mathcal{P}$:

$$\kappa(G, \phi(\mathcal{P}) = g - \deg(\mathcal{P}). \tag{6}$$

B DFS-Burning Algorithm

The algorithm of Perkinson, Yang and Yu starts with a sandpile configuration on a simple graph and gives a spanning tree. At each step of the algorithm a vertex is either deleted or added to the tree. If the vertex is added to the

tree then we say that the vertex is *burnt*.

DFS-burning algorithm

1: We start burning vertices from the sink v_0.

2: We are currently at vertex v_i.

3: If v_i is connected to an unburnt vertex we choose the unburnt vertex with the highest index connected to v_i. Let that be v_j.

4: If v_j has a positive number of grains we delete the edge v_iv_j from the graph and remove 1 grain of sand from v_j. We go back to step 2 for v_i.

5: If v_j has no grains of sand we burn the edge v_iv_j and the vertex v_j. Then, we go to step 2 for vertex v_j.

6: If v_i is not connected to an unburnt vertex, then we go to step 2 for the parent of v_i.

7: We stop when we cannot burn any more vertices.

Perkinson, Yang and Yu [8] also present the inverse of this algorithm.

We can see the DFS-burning algorithm working in Figure 9. At each step we either burn or delete one of the edges of the graph. The vertex with the star is the sink v_0 and the numbers inside each nonsink vertex are the number of sand grains. The orange vertices and edges are already burnt. The dashed edges are already deleted.

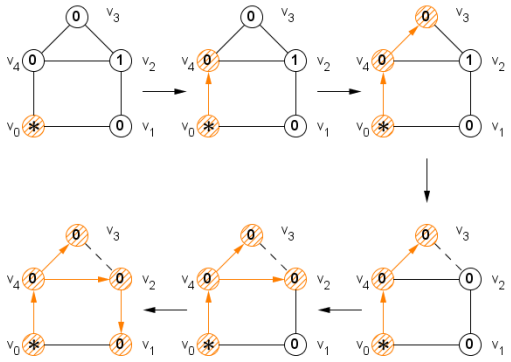

Figure 9: DFS-burning algorithm

C Special case: $\mathbf{x} = (a, b, b, \ldots, b)$

Here we present a bijection between the set of $\mathbf{x}$-parking functions for $\mathbf{x} = (a, b, b, \ldots, b)$ and the set G-parking functions for a specific graph G. The special case of $\mathbf{x}$-parking functions for $\mathbf{x} = (a, b, b, \ldots, b)$ have already been examined by Yan [17]. Her bijection is between these $\mathbf{x}$-parking functions and a special type of trees. On the other hand our bijection is between $\mathbf{x}$-parking functions and G-parking functions for a specific graph G. Our results from Section 2, allow us to expand this bijection for spanning trees of the graph G as well. By n we denote the number of entries in $\mathbf{x}$.

Let $\mathbf{x} = (a, b, b, \ldots, b)$. Then the weakly increasing arrangement of an $\mathbf{x}$-parking function must satisfy the conditions $a_1 \leq a - 1$, $a_2 \leq b + a - 1, \ldots, a_n \leq (n-1)k + a - 1$. The graph we consider is the complete graph on $n + 1$ vertices with weighted edges. Every edge with end vertex v_0 has a weight a while all the other edges have weights b. We can also think about it in terms of the DFS-vector burning algorithm. Since all of the vertices

except the sink v_0 are identical, the w-labelling is irrelevant. Therefore, the v-labeled graph $G_{\mathbf{x}}$ for $\mathbf{x} = (a, b, b, \ldots, b)$ is the same graph. For simplicity in this section to refer to the v-labeled graph as $G_{\mathbf{x}}$.

Definition C.1. *The function* $\psi_1 : \mathrm{PF}(\mathbf{x}) \to \mathrm{PF}(G_{\mathbf{x}})$ *for the vector* $\mathbf{x} = (a, b, b, \ldots, b)$ *is given by the following: for an* $\mathbf{x}$*-parking function* $\alpha = (a_1, a_2, \ldots, a_n)$ *the* G*-parking function* $\mathcal{P} = \psi_1(\alpha)$ *is given by* $\mathcal{P}(v_i) = a_i$ *for each* $i = 1, \ldots, n$.

Theorem C.2. *The function* ψ_1 *is a bijection.*

Proof. It is clear that the function is injective so we need to prove that the codomain of the function is the superstables of $G_{\mathbf{x}}$, and that it is also surjective.

We first prove that the function is well-defined. We assume the opposite, that for $\mathbf{x} = (a, b, b, \ldots, b)$ there exists an $\mathbf{x}$-parking function α such that $\psi_1(\alpha)$ is not a superstable configuration in $G_{\mathbf{x}}$, therefore not a G-parking function. Then there is a set S of l nonroot vertices which can be fired. Without loss of generality, let those vertices be $v_1, \ldots, v_l$ with number of sand grains $a_1 \geq a_2 \geq \cdots \geq a_l$. The number of sand grains on $v_1, \ldots, v_l$ can be at most $(n-1)b+a-1$, $(n-2)b+a-1, \ldots, (n-l)b+a-1$ respectively, because of the inequalities the $\mathbf{x}$-parking functions satisfy. However, the number of edges between the graph G/S and each vertex in S is $\deg_{G/S} v_i = (n-l)b+a$ for each $i = 1, \ldots, l$. This is greater than the number of sand grains on v_l, therefore, the set S cannot be fired — a contradiction. This proves that the codomain of ψ_1 is the set of superstables of $G_{\mathbf{x}}$.

The proof of surjectivity is analogous. The function ψ_1 is well-defined and is also surjective and injective. This means that it is a bijection. $\square$

Now it is sufficient to examine only the G-parking function of the graph $G_{\mathbf{x}}$ when we consider the staircase $\mathbf{x}$-parking function. We provide a formula for the number of spanning trees of $G_{\mathbf{x}}$ which is $a(a+nb)^{n-1}$.