G-PARKING FUNCTIONS AND TREE INVERSIONS

DAVID PERKINSON, QIAOYU YANG, AND KUAI YU

ABSTRACT. A depth-first search version of Dhar’s burning algorithm is used to give a bijection between the parking functions of a graph and labeled spanning trees, relating the degree of the parking function with the number of inversions of the spanning tree. Specializing to the complete graph answers a problem posed by R. Stanley.

1. Introduction

Let $G = (V, E)$ be a connected simple graph with vertex set $V = \{0, \ldots, n\}$ and edge set $E$. Fix a root vertex $r \in V$ and let $\text{SPT}(G)$ denote the set of spanning trees of $G$ rooted at $r$. We think of each element of $\text{SPT}(G)$ as a directed graph in which all paths lead away from the root. If $i, j \in V$ and $i$ lies on the unique path from $r$ to $j$ in the rooted spanning tree $T$, then $i$ is an ancestor of $j$ and $j$ is a descendant of $i$ in $T$. If, in addition, there are no vertices between $i$ and $j$ on the path from the root, then $i$ is the parent of its child $j$, and $(i, j)$ is a directed edge of $T$.

Definition 1. An inversion of $T \in \text{SPT}(G)$ is a pair of vertices $(i, j)$, such that $i$ is an ancestor of $j$ in $T$ and $i > j$. It is a $\kappa$-inversion if, in addition, $i$ is not the root and $i$’s parent is adjacent to $j$ in $G$. The number of $\kappa$-inversions of $T$ is the tree’s $\kappa$-number, denoted $\kappa(G, T)$.

Definition 2. A parking function for $G$ (with respect to the vertex $r$) is a function

$$\mathcal{P}: V \setminus \{r\} \to \mathbb{N}$$

such that for every nonempty set $S \subseteq V \setminus \{r\}$, there exists $i \in S$ such that $\mathcal{P}(i) < \text{deg}_{S^c}(i)$, where $\text{deg}_{S^c}(i)$ is the number of edges $\{i, j\}$ of $G$ with $j \notin S$ (including the possibility $j = r$). The degree of a parking function $\mathcal{P}$ is

$$\text{deg} \mathcal{P} := \sum_{i \in V \setminus \{r\}} \mathcal{P}(i).$$

The set of parking functions for $G$ is denoted by $\text{PF}(G)$.

In this work, we introduce the DFS-burning algorithm. It is a melding of depth-first search with Dhar’s burning algorithm from the Abelian sandpile model, assigning a spanning tree to each parking function of $G$. Our main result is:

Theorem 3. The DFS-burning algorithm (Algorithm 7) gives a bijection $\phi: \text{PF}(G) \to \text{SPT}(G)$ such that

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

where $g := |E| - |V| + 1$ is the genus of $G$. The inverse to $\phi$ is given by Algorithm 8.

The reader is encouraged to refer to Figure 1 for an example.
1.1. Background. Parking functions were originally defined in a study of hashing techniques in computer science, [8], phrased in terms of a problem involving preferences of drivers for parking spaces. Implicit in that definition is the restriction to complete graphs. Parking functions for general graphs were introduced by Postnikov and Shapiro in [20], where they are called $G$-parking functions. Essentially equivalent notions have appeared in a variety of guises: in the Riemann-Roch theory for graphs, parking functions are known as reduced divisors, [1]; in the context of chip-firing games, they are known as superstable configurations, [12]. Superstables are directly related to the recurrent configurations for Dhar’s Abelian sandpile model on the graph, [12, Thm. 4.4], [7], and thus to the set of critical configurations in Biggs’ dollar game, [4], and to Lorenzini’s group of components, [15], [16]. Parking functions in the case of a complete graph have appeared in the theory of symmetric functions, [11], [19], and hyperplane arrangements, [22] (for the latter, see [13] for an extension to more general graphs).

The Tutte polynomial for a simple graph $G$ is

$$T(G, x, y) = \sum_{A \subseteq E} (x - 1)^{c(A) - c(E)} (y - 1)^{c(A) + |A| - |V|}$$

where $c(A)$ is the number of connected components of the subgraph of $G$ with vertex set $V$ and edge set $A$. (For our purposes, we assume $G$ is connected, so $c(E) = 1$.) Translating the work of Merino, [18], into the language of parking functions, $T(1, y) = \sum_{i=0}^{y} a_i y^i$ where $y - a_i$ is the number of parking functions of $G$ of degree $d$. (Hence, $y^d T(1, 1/y)$ is the generating function for the parking functions by degree.) On the other hand, Gessel, [9], shows that $a_i$ is the number of spanning trees of $G$ with $\kappa$-number $i$. Theorem 3 may be regarded as an explanation of the coincidence.

In earlier work, Kreweras, [14], had already noticed that for a complete graph the number of trees with inversion number $a$ equals the number of parking functions of degree $g - a$. Stanley, [22], presents this result and poses the problem of finding a corresponding explicit bijection for the complete graph, $K_n$, that does not depend on recursing through bijections for $K_i$ for $i \leq n$, [22, Chapter 6, Exercise 4]. This problem was the motivation for our work and is generalized and solved by Theorem 3. Note that although the algorithms of Theorem 3 use recursion, they recurse only through the vertices of a fixed graph. The runtime is $O(|V| + |E|)$, as it is for the usual depth-first search of a graph (and it would be a standard exercise to rewrite the algorithm using a stack and avoiding recursion without changing the runtime).

We were influenced by [2], which gives an exposition of the work of Cori and Le Borne in [6]. They describe an algorithm that gives a bijection between parking functions and spanning trees in which a parking function of degree $d$ is assigned a tree with external activity $g - d$. We are indebted to Farbod Shokrieh for explaining this work to us at the American Institute of Mathematics workshop on Generalizations of chip-firing and the critical group, July 2013. For previous work on the correspondence between parking functions and the external activity of associated spanning trees, see [3].

While preparing this manuscript we became aware of the work of Shin, [21], later subsumed in a paper by de Oliveira and Las Vergnas, [10], in which Stanley’s problem had previously been solved. Roughly, depth-first search is used to give a bijection between permutations and trees with no inversions. On the complete graph, permutations may be thought of as maximal-degree parking functions. To extend the bijection to arbitrary parking functions on the complete graph, a procedure is given for relabeling. The advantages of the bijection of Theorem 3 are (i) it applies to arbitrary (simple, connected, labeled) graphs, not only to complete graphs, and (ii) the algorithms providing the bijection and its inverse are less complicated, using little more than depth-first search,
avoiding re-writing rules. We note that restricting our bijection to the case of complete graphs gives a substantially different bijection from that of [10] or [21].

1.2. Organization. In Section 2, we describe and verify the algorithms providing the bijection, then prove Theorem 3. We end the section with a discussion of threshold graphs and prove Proposition 9, which may be of independent interest. Proposition 9 says that for spanning trees of (suitably labeled) threshold graphs, every inversion is a \( \kappa \)-inversion. Section 3 presents the DFS-algorithm and its “inverse”.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure1.png}
\caption{The graph \( G \) with its 11 spanning trees and their corresponding \( \kappa \)-numbers and \( G \)-parking functions (as provided by the DFS-burning algorithm). The root vertex is \( r = 0 \). Each parking function \( \mathcal{P} \) is written as vector with \( i \)-th component \( \mathcal{P}(i) \).}
\end{figure}

Acknowledgements. The authors would like to thank Jim Fix and Farbod Shokrieh for helpful conversations. We thank Collin Perkinson for his comments.

2. Proof of main theorem

The DFS-burning algorithm is presented as Algorithm 1 in Section 3. The idea is to imagine that a fire is started at the root vertex, \( r \), of \( G \) and spreads according to a depth-first rule along the edges until all vertices are burnt. A nonnegative function \( \mathcal{P} \) is thought of as an allocation of drops of water to each non-root vertex. Suppose fire travels along an edge \( e \) to a vertex \( v \). If there are still drops of water on \( v \), one drop will be used to “dampen” \( e \), thus protecting \( v \) from the
flame, and the search backtracks. If there is no remaining water at \( v \), then \( e \) is marked, \( v \) is burnt, and the fire proceeds from \( v \) along an edge to the largest unburnt neighboring vertex. In the end, either (i) all vertices are burnt, \( \mathcal{P} \) is a parking function, and the collection of marked edges forms a spanning tree, \( T_{\mathcal{P}} \), (and the number of dampened edges is \( \deg \mathcal{P} \)) or (ii) the nonempty set of unburnt vertices, \( S \), serves as a certificate that \( \mathcal{P} \) is not a \( G \)-parking function: \( \mathcal{P}(j) \geq \deg_{S_{c}}(j) \) for each \( j \in S \).

**Example 4.** Figure 2 illustrates a running of the DFS-burning algorithm. Vertex \( i \) is labeled \( v_{i} \). The value of the parking function \( \mathcal{P} \) at \( v_{i} \) is the \( i \)-th component of the vector \( (0, 0, 1, 0) \). The root vertex, \( r = 0 = v_{0} \), is lit and fire spreads along the edge to the highest-numbered adjacent vertex, \( v_{2} \). Since there are no drops of water on \( v_{2} \), i.e., since \( \mathcal{P}(v_{2}) = 0 \), the vertex \( v_{2} \) is burnt and \( (v_{0}, v_{2}) \) is added to the list of tree edges. Similarly, the fire spreads from \( v_{2} \), causing \( v_{4} \) to be burnt and \( (v_{2}, v_{4}) \) to become a tree edge. The fire then attempts to spread to \( v_{3} \), but the drop of water there is used to dampen the edge \( (v_{4}, v_{3}) \). Backtracking to \( v_{2} \), the fire then spreads to the remaining vertices. Note that the number of dampened edges is \( \deg \mathcal{P} \).

The resulting spanning tree, \( \phi(\mathcal{P}) \), has inversions \( (v_{2}, v_{1}) \) and \( (v_{3}, v_{1}) \), but only \( (v_{2}, v_{1}) \) is a \( \kappa \)-inversion since the parent of \( v_{3} \) in the tree is \( v_{2} \) and \( \{v_{1}, v_{2}\} \) is not an edge in the graph. In accordance with Theorem 3, we have \( g - \deg \mathcal{P} = 2 - 1 = 1 \).

**Verification of Algorithm 1 (cf. Section 3).** The DFS-burning algorithm must terminate since the function \( \text{dfs} \) from() is called at most once per vertex, and the loop starting at line 7 then eventually considers (perhaps after backtracking from a later call to \( \text{dfs} \) from()) each adjacent vertex exactly once.

If every vertex is burnt, a collection of \( n \) edges is returned, which forms a connected subgraph of \( G \) containing all \( n + 1 \) vertices and hence is a spanning tree. To see that in this case \( \mathcal{P} \) is a
Proof of Theorem 3. Let $j$ be the first vertex of $S$ to be burnt. Just before $j$ is added to the list of burnt vertices (line 10), $\mathcal{P}(j)$ edges incident on $j$ will have been already added to the list of dampened edges. Then, just after $j$ is burnt, a new edge incident on $j$ is added to the list of tree edges. Each of these edges has the form $(i, j)$ where $i$ is a vertex burnt prior to $j$, and hence $i \notin S$. This shows that $\mathcal{P}(j) < \deg_{S'}(j)$, as required.

Now suppose that when the algorithm terminates, not every vertex is burnt. Let $S$ be the nonempty set of unburnt vertices. Take $j \in S$, and consider the set $I$ of vertices adjacent to $j$ but not in $S$. For each $i \in I$, the algorithm at some point added the edge $(i, j)$ to the set of dampened edges and decreased $\mathcal{P}(j)$ by one (while maintaining its nonnegativity). Thus, $\mathcal{P}(j) \geq \deg_{S'}(j)$ (for the original input function $\mathcal{P}$), which shows that $\mathcal{P}$ is not a $G$-parking function.

The DFS-burning algorithm provides a bijection

$$\phi: \mathbf{PF}(G) \rightarrow \mathbf{SPT}(G)$$

$$\mathcal{P} \mapsto T_{\mathcal{P}}.$$ 

It is straightforward to check that the inverse is given by Algorithm 2 starting with a spanning tree $T$, the idea is to preliminarily set $\mathcal{P} = 0$, then run the DFS-burning algorithm but adjust $\mathcal{P}$ along the way to force the output to be $T$. The resulting $\mathcal{P}$ is $\phi^{-1}(T)$.

We now proceed to a proof of our main result, Theorem 6.

Definition 5. The depth-first search tree (DFS-tree) of $G$, denoted $\text{DFS}(G)$, is the output of the DFS-burning algorithm with input $\mathcal{P} = 0$.

Lemma 6. Let $g = |E| - |V| + 1 = |E| - n$ be the genus of $G$. Then $\kappa(G, \text{DFS}(G)) = g$.

Proof. Suppose $e = \{i', j\}$ is an edge of $G$ but not an edge of $\text{DFS}(G)$. Without loss of generality, assume $i'$ is added to the list of burnt vertices before $j$ in the execution of the DFS-burning algorithm used to create $\text{DFS}(G)$. Since $\{i', j\}$ is not in $\text{DFS}(G)$, when the algorithm burns $j$, it has not yet backtracked to $i'$. Hence, there is a path in $\text{DFS}(G)$ from $i'$ to $j$ of edges directed away from the root. If $i$ is the child of $i'$ in this path, then $(i, j)$ is a $\kappa$-inversion for $\text{DFS}(G)$. In this way, we get a bijection between edges of $G$ that are not edges of $\text{DFS}(G)$ and $\kappa$-inversions of $\text{DFS}(G)$. The result follows.

Lemma 7. Suppose $H$ is a connected graph obtained by deleting an edge of $\text{DFS}(G)$ from $G$. The $\kappa$-inversions of $\text{DFS}(H)$ as a subgraph of $G$ are the same as those as a subgraph of $H$ (assuming the same root for both $G$ and $H$). Hence,

$$\kappa(G, \text{DFS}(H)) = \kappa(H, \text{DFS}(H)).$$

Proof. Let $T = \text{DFS}(H)$. Trivially, $\kappa$-inversions of $T$ in $H$ are $\kappa$-inversions for $T$ in $G$. We prove the opposite inclusion by contradiction. Suppose $(i, j)$ is a $\kappa$-inversion for $T$ in $G$ but not in $H$. In other words, letting $i'$ be the parent of $i$ in $T$, the edge $e = \{i', j\}$ is in $G$ but not in $H$, hence, $e$ is the edge of $\text{DFS}(G)$ deleted to obtain $H$.

Since $G$ and $H$ differ only in the edge $e$, the depth-first searches of both $G$ and $H$ are the same up to the point at which the vertex $i'$ of $e$ is reached. Next, since $i > j$, the depth-first search of $G$ travels from $i'$ to $i$, i.e., $(i', i)$ is an edge of $\text{DFS}(G)$. Subsequently, the path from $i$ to $j$ in $T$ must also be eventually added to $\text{DFS}(G)$. Hence, $e$ cannot be in $\text{DFS}(G)$, which is a contradiction.

Proof of Theorem 6. Let $\mathcal{P} \in \mathbf{PF}(G)$. It remains to be shown that $\kappa(G, \phi(\mathcal{P})) = g - \deg \mathcal{P}$.

Let $D = \{e_1, \ldots, e_t\}$ be the dampened edges resulting from applying the DFS-burning algorithm to $\mathcal{P}$. We assume that these edges are listed in the order they were found by the algorithm, and
note that \( k = \deg \mathcal{P} \). Define \( G_0 := G \), and for \( 0 \leq \ell \leq k \), let \( G_\ell \) be the graph obtained from \( G_{\ell-1} \) by removing edge \( e_\ell \). Each \( G_\ell \) contains \( \phi(\mathcal{P}) \) and is consequently connected. Further, for \( G_k \), obtained from \( G \) by removing all the dampened edges, we have \( \text{DFS}(G_k) = \phi(\mathcal{P}) \).

We now show that \( e_\ell \in \text{DFS}(G_{\ell-1}) \) for \( 0 \leq \ell < k \), from which the result follows by Lemma 6 and repeated application of Lemma 7. The key idea is that starting with any connected simple graph and any nonnegative function on the graph’s non-root vertices, the first dampened edge created by the DFS-burning algorithm is an edge in the graph’s depth-first search tree. For example, \( e_1 \) is an edge of \( \text{DFS}(G) = \text{DFS}(G_0) \). Define \( \mathcal{P}_0 = \mathcal{P} \), and for \( 1 \leq \ell < k \), define \( \mathcal{P}_\ell : V \setminus \{r\} \to \mathbb{N} \) by

\[
\mathcal{P}_\ell(j) = \begin{cases} 
\mathcal{P}_{\ell-1}(j) - 1 & \text{if } e_\ell = (i, j), \\
\mathcal{P}_{\ell-1}(j) & \text{otherwise}.
\end{cases}
\]

Then \( e_{\ell+1} \) is in \( \text{DFS}(G_\ell) \) since it is the first dampened edge when the algorithm is run with input \( G_\ell \) and \( \mathcal{P}_\ell \) (the full sequence of dampened edges being \( e_{\ell+1}, \ldots, e_k \)).

\[\square\]

2.1. Special case. Threshold graphs are a family of graphs, including the complete graphs, introduced by Chvatal and Hammer [5]. For a comprehensive study, see [17]. We show that if properly labeled, there is no distinction between inversions and \( \kappa \)-inversions for their spanning trees.

**Definition 8.** A graph is a threshold graph if it can be constructed from a graph with one vertex and no edges by repeatedly carrying out the following two steps:

- Add a dominating vertex: a vertex that is connected to every other existing vertex.
- Add an isolated vertex: a vertex that is not connected to any other existing vertex.

A threshold graph with more than one vertex is connected if and only if the last-added vertex is dominating. Each threshold graph is uniquely defined by its build sequence: a string starting with the symbol * (for the initial vertex) followed by any string consisting of the letters \( d \) (for the addition of a dominating vertex) and \( i \) (for the addition of an isolated vertex). Thus, \(*iddid\) describes the threshold graph pictured in Figure 3 formed from a single vertex by adding, in order, an isolated vertex, two dominating vertices, an isolated vertex, then a final dominating vertex. Omitting * and reading left-to-right, group the consecutive sequences consisting entirely of a single

![Figure 3](image-url)

**Figure 3.** The threshold graph with build sequence \(*iddid\). The vertices are labeled \( \alpha, \beta, \gamma, \delta, \epsilon, \zeta \), in the order of the build sequence.

letter (either \( d \) or \( i \)) into blocks, then include * in the first block. Thus, the sequence of blocks for \(*iddid\) is \([*i], [dd], [i], [d]\).
We say that a threshold graph is labeled by reverse degree sequence if its vertices are labeled by 0, ..., n in such a way that \( \text{deg}(i) \geq \text{deg}(j) \) for each pair of vertices \( i < j \). If there is more than one vertex with the same degree, the labeling is not unique. For example, for the graph in Figure 3,

(i) \( \zeta \) must be labeled 0,
(ii) \( \gamma, \delta \) must be labeled 1, 2 in either order,
(iii) \( \alpha, \beta \) must be labeled 3, 4 in either order, and
(iv) \( \epsilon \) must be labeled 5.

**Proposition 9.** Let \( G \) be a connected threshold graph labeled by reverse degree sequence, and let \( T \) be a spanning tree of \( G \). Then every inversion of \( T \) is a \( \kappa \)-inversion.

**Proof.** First note that two vertices have the same degree in \( G \) if and only if they belong to the same block of the build sequence for \( G \). The degree of any vertex in a \( d \)-block is greater than the degree of any vertex in an \( i \)-block. Also, block-by-block, the degrees for vertices in successive \( d \)-blocks increase from left-to-right, and the degrees for \( i \)-blocks decrease.

Let \((i, j)\) be an inversion of \( T \), and let \( i' \) be the parent of \( i \). We must show that \( \{i', j\} \in E \), where \( E \) denotes the set of edges of \( G \). Since \( i > j \) and \( G \) is labeled by reverse degree sequence, \( \text{deg}(i) \leq \text{deg}(j) \). If \( \text{deg}(i) = \text{deg}(j) \), then \( i \) and \( j \) belong to the same block in the build sequence for \( G \). In that case, not counting each other, \( i \) and \( j \) have the same neighbors. Hence, \( \{i', j\} \in E \). Otherwise, \( \text{deg}(i) < \text{deg}(j) \), and the result follows using the build sequence and considering cases:

- If \( i \) and \( j \) are both dominating, then \( \text{deg}(i) < \text{deg}(j) \) implies \( j \) follows \( i \) in the build sequence, and thus \( i' \) must be adjacent to \( j \).
- If \( i \) is isolated and \( j \) is dominating, then \( i' \) must be dominating. So \( \{i', j\} \in E \) since every pair of dominating vertices are adjacent in \( G \).
- If both \( i \) and \( j \) are isolated, then \( j \) precedes \( i \) in the build sequence and \( i' \) is a dominating vertex added after \( i \). Hence, \( \{i', j\} \in E \).
- It is not possible for \( i \) to be dominating and \( j \) to be isolated since \( \text{deg}(i) < \text{deg}(j) \).

\( \square \)

**Remark 10.** In light of Proposition 9 for a threshold graph, one may replace \( \kappa(G, \phi(P)) \) in Theorem 3 by the number of inversions of \( \phi(P) \). Specializing to the case of the complete graph gives a solution to a problem posed by Stanley [22, Chapter 6, Exercise 4].
3. Algorithms

In the following algorithms, it is assumed that $G$ is a simple connected graph with vertices $V = \{0, \ldots, n\}$ and fixed root vertex $r \in V$.

**Algorithm 1 DFS-burning algorithm.**

1: function $\text{dfs\_burn}(P)$
2: \hspace{1em} burnt\_vertices = \{r\}
3: \hspace{1em} dampened\_edges = \{\}
4: \hspace{1em} tree\_edges = \{\}
5: \hspace{1em} $\text{dfs\_from}(r)$
6: function $\text{dfs\_from}(i)$
7: \hspace{2em} foreach $j$ adjacent to $i$ in $G$, from largest numerical value to smallest do
8: \hspace{3em} if $j \notin \text{burnt\_vertices}$ then
9: \hspace{4em} if $P(j) = 0$ then
10: \hspace{5em} append $j$ to burnt\_vertices
11: \hspace{5em} append $(i, j)$ to tree\_edges
12: \hspace{4em} $\text{dfs\_from}(j)$
13: \hspace{3em} else
14: \hspace{4em} $P(j) = P(j) - 1$
15: \hspace{4em} append $(i, j)$ to dampened\_edges

Algorithm 1 Let $P : V \setminus \{r\} \rightarrow \mathbb{N}$. After running $\text{dfs\_burn}(P)$, if $\text{burnt\_vertices} = V$, then $P$ is a $G$-parking function (with respect to the given root vertex, $r$) and $\text{tree\_edges}$ forms a spanning tree of $G$. In this way, $\text{dfs\_burn}$ provides a function $\phi : \text{PF}(G) \rightarrow \text{SPT}(G)$ from $G$-parking functions to spanning trees of $G$.

If $\text{burnt\_vertices} \neq V$, the nonempty set $S := V \setminus \text{burnt\_vertices}$ has the property that $P(j) \geq \deg_{S^c}(j)$ for all $j \in S$, certifying that $P$ is not a $G$-parking function.
Algorithm 2 Tree to $G$-parking function algorithm.

1: function $\text{TREE\_TO\_PARKING\_FUNCTION}(T)$
2: \hspace{1em} burnt_vertices = \{r\}
3: \hspace{1em} $P = 0$ // (the 0-function on the non-root vertices of $G$)
4: \hspace{1em} $\text{TREE\_FROM}(r)$

5: function $\text{TREE\_FROM}(i)$
6: \hspace{1em} foreach $j$ adjacent to $i$ in $G$, from largest numerical value to smallest do
7: \hspace{2em} if $(i, j)$ is an edge of $T$ then
8: \hspace{3em} append $j$ to burnt_vertices
9: \hspace{3em} $\text{TREE\_FROM}(j)$
10: \hspace{2em} else
11: \hspace{3em} $P(j) = P(j) + 1$

Algorithm 2 Let $T$ be a spanning tree rooted at $r$ with edges directed away from the root. After running $\text{TREE\_TO\_PARKING\_FUNCTION}(T)$, the function $P$ is a $G$-parking function. In this way, $\text{TREE\_TO\_PARKING\_FUNCTION}$ provides the inverse to $\phi : PF(G) \to SPT(G)$ given by DFS-burning algorithm.
References

[1] Matthew Baker and Serguei Norine. Riemann-Roch and Abel-Jacobi theory on a finite graph. *Adv. Math.*, 215(2):766–788, 2007.

[2] Matthew Baker and Farbod Shokrieh. Chip-firing games, potential theory on graphs, and spanning trees. *J. Combin. Theory Ser. A*, 120(1):164–182, 2013.

[3] Janet Simpson Beissinger. On external activity and inversions in trees. *J. Combin. Theory Ser. B*, 33(1):87–92, 1982.

[4] N. L. Biggs. Chip-firing and the critical group of a graph. *J. Algebraic Combin.*, 9(1):25–45, 1999.

[5] Václav Chvátal and Peter L. Hammer. Aggregation of inequalities in integer programming. In *Studies in integer programming (Proc. Workshop, Bonn, 1975)*, pages 145–162. Ann. of Discrete Math., Vol. 1. North-Holland, Amsterdam, 1977.

[6] Robert Cori and Yvan Le Borgne. The sand-pile model and Tutte polynomials. *Adv. in Appl. Math.*, 30(1-2):44–52, 2003. Formal power series and algebraic combinatorics (Scottsdale, AZ, 2001).

[7] Deepak Dhar. Theoretical studies of self-organized criticality. *Phys. A*, 369(1):29–70, 2006.

[8] Konheim A. G. and Weiss B. An occupancy discipline and applications. *SIAM J. Applied Math.*, 14:1266–1274, 1966.

[9] Ira M. Gessel. Enumerative applications of a decomposition for graphs and digraphs. *Discrete Math.*, 139(1-3):257–271, 1995. Formal power series and algebraic combinatorics (Montreal, PQ, 1992).

[10] António Guedes de Oliveira and Michel Las Vergnas. Parking functions and labeled trees. *Sém. Lothar. Combin.*, 65:Art. B65e, 10, 2010/12.

[11] Mark D. Haiman. Conjectures on the quotient ring by diagonal invariants. *J. Algebraic Combin.*, 3(1):17–76, 1994.

[12] Alexander E. Holroyd, Lionel Levine, Karola Mészáros, Yuval Peres, James Propp, and David B. Wilson. Chip-firing and rotor-routing on directed graphs. In *In and out of equilibrium. 2*, volume 60 of *Progr. Probab.*, pages 331–364. Birkhäuser, Basel, 2008.

[13] S. Hopkins and D. Perkinson. Bigraphical Arrangements. eprint, arXiv:1212.4398, 2012.

[14] G. Kreweras. Une famille de polynômes ayant plusieurs propriétés énumératives. *Period. Math. Hungar.*, 11(4):309–320, 1980.

[15] Dino J. Lorenzini. Arithmetical graphs. *Math. Ann.*, 285(3):481–501, 1989.

[16] Dino J. Lorenzini. A finite group attached to the Laplacian of a graph. *Discrete Math.*, 91(3):277–282, 1991.

[17] N. V. R. Mahadev and U. N. Peled. *Threshold graphs and related topics*, volume 56 of *Annals of Discrete Mathematics*. North-Holland Publishing Co., Amsterdam, 1995.

[18] Criel Merino López. Chip firing and the Tutte polynomial. *Ann. Comb.*, 1(3):253–259, 1997.

[19] Jean-Christophe Novelli and Jean-Yves Thibon. Hopf algebras and dendriform structures arising from parking functions. *Fund. Math.*, 193(3):189–241, 2007.

[20] Alexander Postnikov and Boris Shapiro. Trees, parking functions, syzygies, and deformations of monomial ideals. *Trans. Amer. Math. Soc.*, 356(8):3109–3142 (electronic), 2004.

[21] Heesung Shin. A New Bijection Between Forests and Parking Functions. eprint, arXiv:0810.0427, 2008.

[22] Richard P. Stanley. An introduction to hyperplane arrangements. In *Geometric combinatorics*, volume 13 of *IAS/Park City Math. Ser.*, pages 389–496. Amer. Math. Soc., Providence, RI, 2007.

E-mail address: davidp@reed.edu

E-mail address: yangq@reed.edu

E-mail address: kyu@reed.edu

Reed College, Portland OR, 97202