A COMPUTATIONAL APPROACH TO THE ENUMERATION OF G-PARKING FUNCTIONS

TYLER ADEN AND BRIAN BENSON

Abstract. Given a connected graph G, we prove an algorithm to efficiently list all maximal G-parking functions. Since many combinatorial bijections between G-parking functions and spanning trees have been established in the literature, enumeration of spanning trees of G and G-parking functions can be viewed as related problems. As an application, we implement our algorithm into Python to consider the enumeration of G-parking functions on the hypercube Q_n in order to consider a simple combinatorial proof of the formula for the number of spanning trees of Q_n, which Stanley refers to as not being known. Our algorithm allows us to consider an experimental approach to this problem.

1. Introduction

We should write this later.

Let $\mathcal{P}(G,q)$ be the set of G-parking functions $f : V \to \mathbf{Z}_{\geq -1}$ where q is the unique vertex in V such that $f(q) = -1$.

For a G-parking function f, the set $\mathrm{dom}(f)$ is defined to be

$$\mathrm{dom}(f) = \big\{ g \in \mathcal{P}(G,q) : g(v) \leq f(v) \text{ for all } v \in V \big\}.$$

Note that the parking function $\vec{0} \in \mathcal{P}(G,q)$, which defined to be $\vec{0}(q) = -1$ and $\vec{0}(v) = 0$ for all $v \in V \setminus \{q\}$, is in $\mathrm{dom}(f)$ for every $f \in \mathcal{P}(G,q)$.

Given a collection of maximal G-parking functions f_i with $i = 1, 2, \ldots, k$, the sets of intersections of the form $\cap_{i=1}^{k} \mathrm{dom}(f_i)$ have not been studied to our knowledge.

2. An algorithm for finding all maximal G-parking functions

We will write the connected graph G as $G = (V,E)$, where V is the set of vertices which we have enumerate and denote by $v_1, v_2, \ldots, v_n$ and E is a set of triples of the form (v_i, v_j, k). Here, we have that $v_i, v_j \in V$ and $k \in \mathbf{N}$ denotes the number of edges between v_i and v_j. We will assume that the input of the algorithm is the adjacency matrix, denote it $\mathcal{A}$, of the graph G. Let A_{ij} correspond to the entry in the i-th row and j-th column of $\mathcal{A}$. Recall that, by the definition of the adjacency matrix, the entry $A_{ij} = k$ where k denotes the number of edges between v_i and v_j.

We will write G-parking functions $f : V \to \mathbf{Z}_{\geq -1}$ as $1 \times n$ arrays of the form $[f(v_1), f(v_2), \ldots, f(v_n)]$ where we use commas to more easily distinguish the entries of the array.

To simplify the algorithm, we will always assume that $v_1 = q$, where q is the root or source vertex in the definition of a G-parking function. By a subgraph S of G, we mean that $S = (V_S, E_S)$ such that $V_S \subseteq V$ with $E_S \subset E$ such that $(v_i, v_j, k) \in E_S$ if and only if $v_i \in V_S$ and $v_j \in V_S$ and $(v_i, v_j, k) \in E$. The algorithm can be thought of as a breath first search of maximal G-parking functions where level k in the search corresponds to finding a maximal G-parking function on each connected subgraph having $k+1$ vertices and always including the vertex $q = v_1$. As a result, we will begin the algorithm with the $1 \times n$ array representing our G-parking function where the first entry is -1 and the remaining $n-1$ entries are empty. With respect to the previously mentioned subgraph correspondence, any empty entry is thought of as having its corresponding vertex removed from the subgraph.

We write the initial array as $F = \big[\underbrace{-1, \_\_, \_\_, \ldots, \_\_}_{n \text{ entries}}\big]$, which can be interpreted as the only parking function on the graph $(\{q\}, \emptyset)$. Each step in the algorithm will fill one empty entry.

(1) For $i = 2, 3, \ldots, n$, if $A_{1i} > 0$, fill in the $i-th$ entry of the array F with $A_{1i} - 1$ to create F_i. If $A_{1i} = 0$, the vertices v_1 and v_i do not share an edge and so do not form a connected subgraph, so there is no vertex corresponding to F_i in the search tree.

(2) For $k \in \{2, 3, \ldots, n\}$, the k-th step is defined as follows. Each vertex of the search tree is indexed by a $(k-1)$-tuple of the form $(i_1, i_2, \ldots, i_{k-1})$ with $i_j \in \{2, 3, \ldots, n\}$ and $i_\alpha \neq i_\beta$ for $\alpha \neq \beta$. For each $i_k \in \{2, 3, \ldots, n\} \setminus \{i_1, i_2, \ldots, i_{k-1}\}$, if

$$A_{1k} + \sum_{j=1}^{k-1} A_{i_j i_k} > 0 \text{ and } \big(A_{i_{k-1} i_k} \neq 0 \text{ or } i_{k-1} < i_k\big), \tag{2.1}$$

(where the parentheses are to distinguish the top level connective) then fill in the i_k-th entry in $F_{i_1 i_2 \cdots i_{k-1}}$ with $A_{1k} + \sum_{j=1}^{k-1} A_{i_j i_k} - 1$ to create $F_{i_1 i_2 \cdots i_{k-1} i_k}$. Otherwise, there is no vertex corresponding to $F_{i_1 i_2 \cdots i_{k-1} i_k}$ in the search tree.

Note that whenever $A_{i_{k-1} i_k} = A_{i_k i_{k-1}} = 0$, burning vertex i_{k-1} directly before burning vertex i_k will result in the same maximal G-parking function in the search regardless of which level of the search these are in. This is the reason for specifying that $A_{i_{k-1} i_k} \neq 0$ or $i_{k-1} < i_k$ in addition to $A_{1k} + \sum_{j=1}^{k-1} A_{i_j i_k} > 0$ in the statement (2.1).

The final leaves of the search tree are indexed by $(k-1)$-tuples which correspond to the burn order of the vertices in the Dhar extended algorithm.

3. An Application of Maximal G-Parking Functions

We think that the study of maximal G-parking functions might have practical applications. Consider the following simple, location model for minimizing costs to prevent infectious disease transmission. In the model, the vertices represent locations (such as cities) and the edges represent direct travel routes (such as airline routes) between locations. The choice of a vertex q in the G-parking function represents the source of an outbreak and a function $f : V \setminus \{q\} \to \mathbf{Z}_{\geq 0}$ represents the allocation of resources (the amount of money spent) to fight the spread of the disease beginning at the vertex q. When a vertex is burned in Dhar's algorithm, it represents the spread of the disease to that vertex or location. The G-parking functions represent the "disaster outcomes" where the disease spread to all of the vertices or locations in the model. A maximal G-parking function represents a resource allocation plan where all vertices have been infected, but allocating an additional unit of resources at any vertex will stop the transmission to that vertex. Furthermore, an additional unit of resources allocated to the neighbors of q will stop the

What about "loading" the neighbors of q? If a miscalculation is made in terms of resources providing all of the resources to neighboring vertices of q does not provide any additonal protection against transmission for vertices more than distance 1 from q. Can we introduce probability of transmission into the model? Can we generalize G-parking functions to real numbers?

4. Questions To Pursue

(1) For a Q_n-parking function, we can associate an ordered n-tuple $(x_1, x_2, \ldots, x_{n-1})$ such that $x_k = |\{v \in V : f(v) = k\}|$. For each such n-tuple, how many maximal Q_n-parking functions are associated to this triple?
(2) Look at intersections of $\operatorname{dom}(f)$ where f are maximal G-parking functions.

5. Python Code

Insert Code Here.

6. A Catalog of Q_n-Parking Functions For $n = 3, 4, 5$.

6.1. **List of Q_3-Parking Functions.**

6.2. **List of Q_4-Parking Functions.**

6.3. **List of Q_5-Parking Functions.** *Place holder.*

Department of Mathematics, Kansas State University, Manhattan, KS 66506

E-mail address: tkaden@ksu.edu

E-mail address: babenson@ksu.edu