CORNELL UNIVERSITY MATHEMATICS DEPARTMENT SENIOR THESIS

The Weak Order and Flag h-Vector Inequalities

A THESIS PRESENTED IN PARTIAL FULFILLMENT
OF CRITERIA FOR HONORS IN MATHEMATICS

Timothy D. DeVries

May 2005

BACHELOR OF ARTS, CORNELL UNIVERSITY

THESIS ADVISOR(S)

Edward Swartz
Department of Mathematics

Acknowledgements. I would like to thank my thesis advisor Edward Swartz, without whose counsel and direction this paper would not exist. I would also like to thank professor Louis Billera, whose courses in combinatorics triggered my initial interest in the subject.

Finally, I'd like to thank my friends and family (especially you, Dad). Without their understanding and support - emotional, monetary and otherwise - I would never have made it through Cornell.

THE WEAK ORDER AND FLAG h-VECTOR INEQUALITIES

TIMOTHY D. DEVRIES, CORNELL UNIVERSITY

Abstract. The following paper examines properties of the weak order on the groups S_{r+1} and B_{r+1} as motivated by the search for a combinatorial proof of known h-vector inequalities. Several general results are developed with respect to descent set domination, and an explanation of the computer programs used to generate a number of these results is presented. While this paper primarily focuses on h- and flag h-vector inequalities of the order complex of geometric lattices, such inequalities for the order complexes of distributive and supersolvable lattices are also explored. Finally, several courses of action for further study are considered.

1. Preliminaries

We begin with some preliminary terminology (see [NS] for further details). A *simplicial complex* Δ is a nonempty collection of subsets of a finite set V such that the following two properties hold:

(1) For all $F \in \Delta$, $G \subseteq F \Rightarrow G \in \Delta$

(2) For all $v \in V$, $\{v\} \in \Delta$.

The subsets of V in Δ are called the *faces,* and for a given face $F \in \Delta$ the *dimension* of F is defined to be $|F| - 1$. The *dimension* of Δ is then defined to be the largest dimension of any face of Δ. The faces of Δ which are also maximal subsets of Δ are called *facets.* If all the facets of Δ are of the same dimension, then Δ is said to be *pure.* Below is an example of a well-known simplicial complex.

Example 1.1. Given a connected graph G, a simplicial complex Δ_G is defined as follows: Let V be the set of edges of G whose removal does not disconnect G. Then define Δ_G so that $F \subseteq V$ implies $F \in \Delta_G$ if and only if the removal of the edges in F from G leaves a connected graph. It is simple to check that Δ_G satisfies the given properties of a simplicial complex.

Two objects associated with any simplicial complex are the f- and h-vectors. For a simplical complex Δ of dimension $d-1$, the f-vector is defined to be $(f_{-1}, f_0, \ldots, f_{d-1})$, where f_i counts the number of i-dimensional faces of Δ. The h-vector $(h_0, h_1, \ldots, h_d)$ is in turn defined in terms of the f-vector so that

Date: April 29, 2005.

Key words and phrases. h-vector, flag h-vector, weak order, matching, permutations.

the following equality holds:

$$\sum_{i=0}^{d} f_{i-1}(x-1)^{d-i} = \sum_{i=0}^{d} h_i x^{d-i}$$

(in certain classes of simplicial complexes these h_i's also have a general combinatorial interpretation; for instance, in what are known as shellable complexes). Calculating the coefficient of x^{d-i} (for $i \in \{0, 1, \ldots, d\}$) by binomial expansion yields the following formula for the elements of the h-vector:

$$h_i = \sum_{j=0}^{i} (-1)^{i-j} \binom{d-j}{d-i} f_{j-1}. \tag{1.1}$$

It is clear from the above equations that studying the f-vector is equivalent to studying the h-vector. In what follows, the h-vector will be the object of primary interest.

The specific simplicial complexes primarily investigated in this paper are the order complexes of geometric lattices, which we now present. First, let P be a finite poset. P is said to be *graded* if all maximal chains in P are of the same length. This length is then defined to be the *rank* of such a P. Given any element $x \in P$, the rank of x (denoted $\rho(x)$) is defined to be the length of the longest chain $x_0 < x_1 < \ldots < x_{\rho(x)} = x$ in P. A *lattice* L is a poset such that for all $x, y \in L$, x and y have a least upper bound (or *join,* denoted $x \vee y$) and a greatest lower bound (or *meet,* denoted $x \wedge y$). Consequently, a finite lattice L has a unique minimal element (denoted by $\hat{0}$) and a unique maximal element (denoted by $\hat{1}$). Those elements $x \in L$ that *cover* $\hat{0}$, i.e. such that $\hat{0} < y < x$ for no $y \in L$, are called the *atoms* of L. L is said to be *atomic* if every element in L can be written as the join of atoms. Finally, a *geometric lattice* L may be defined as a graded atomic lattice whose rank function satisfies the following condition:

$$\rho(x \vee y) + \rho(x \wedge y) \leq \rho(x) + \rho(y) \quad \text{for all } x, y \in L.$$

As an example of a geometric lattice, consider the set of subspaces of a finite vector space. It is not difficult to verify that such a set does indeed satisfy all the conditions of a geometric lattice.

Now let L be a geometric lattice of rank $r+1$. From L one can construct a simplicial complex $\Delta(L)$, known as the *order complex* of L and having dimension $r-1$, as follows: Let $\hat{0}, \hat{1}$ denote the minimal and maximal elements of L, respectively. Then the set V is defined to be the set $L - \{\hat{0}, \hat{1}\}$, and the elements of $\Delta(L)$ are all the chains of $L - \{\hat{0}, \hat{1}\}$. Note that the order complex may be defined for an arbitrary poset P.

The main motivation of this paper can now be stated. We wish to investigate the h-vector of the order complex of geometric lattices, seeking inequalities which relate the elements of the h-vector. Specifically, we are interested in a theorem proved in [NS], which asserts:

Theorem 1.2. *For L a rank $r+1$ geometric lattice, the h-vector of $\Delta(L)$ satisfies:*

$$h_i \leq h_{i+1} \quad \text{for } i < \lfloor \frac{r}{2} \rfloor, \text{ and}$$
$$h_i \leq h_{r-i} \quad \text{for } i \leq \lfloor \frac{r}{2} \rfloor.$$

The proof of this theorem involves the fact that the order complex of a geometric lattice has what is known as a convex ear decomposition. In this paper we are seeking a more geometric, or combinatorial (and hence more intuitive) proof of this fact.

2. Initial Work

The foundation upon which this paper's investigation is built is due to Nyman and Swartz's paper entitled "Inequalities for the h- and flag h-vectors of geometric lattices." This paper marks in part a slight extension of the original investigation contained therein. This section will present and develop several fundamental results from Nyman and Swartz's work.

First note that oftentimes in studying a mathematical object, it is best to define a seemingly more complex object in an attempt to study the original object's properties. With respect to the f- and h-vectors, these new objects are the *flag* f- and h-vectors. The flag f-vector is a refinement of the f-vector. In the case of the order complex of a lattice L of rank $r+1$, it is defined in the following way. Given $S \subseteq [r] = \{1, \ldots, r\}$, define the subposet L_S of L by:

$$L_S = \{x \in L : \rho(x) \in S, x = \hat{0} \text{ or } x = \hat{1}\}$$

Then define an integer $f_S(L)$ by:

$$f_S(L) = \text{number of maximal chains of } L_S.$$

The collection of all such $f_S(L)$ is called the flag f-vector of L. Cleary by definition, we obtain:

$$f_{i-1}(\Delta(L)) = \sum_{|S|=i} f_S(L) \quad \text{for } i \in \{0, 1, \ldots, r\}$$

justifying the above claim that the flag f-vector is a refinement of the original f-vector. The elements of the flag h-vector are in turn defined in terms of the elements of the flag f-vector. Specifically for the order complex $\Delta(L)$ of a lattice of rank $r+1$, define:

$$h_S(L) = \sum_{T \subseteq S} (-1)^{|S|-|T|} f_T(L), \quad \text{for } S \subseteq [r].$$

Then the flag h-vector is the set of all such $h_S(L)$.

Proposition 2.1. *Let L be a lattice of rank $r+1$. Then on the order complex of L:*

$$h_i(\Delta(L)) = \sum_{|S|=i} h_S(L) \quad \text{for } i \in \{0, 1, \ldots, r\},$$

i.e. the flag h-vector is a refinement of the h-vector.

Proof. For $S \subseteq [r]$, $h_S(L) = \sum_{T \subseteq S} (-1)^{|S|-|T|} f_T(L)$. This implies:

$$\begin{aligned}
\sum_{|S|=i} h_S(L) &= \sum_{|S|=i} \sum_{T \subseteq S} (-1)^{i-|T|} f_T(L) \\
&= \sum_{j=0}^{i} \sum_{|T|=j} \sum_{S:|S|=i, S \supseteq T} (-1)^{i-j} f_T(L) \\
&= \sum_{j=0}^{i} \sum_{|T|=j} (-1)^{i-j} \binom{r-j}{i-j} f_T(L) \\
&= \sum_{j=0}^{i} (-1)^{i-j} \binom{r-j}{i-j} \sum_{|T|=j} f_T(L) \\
&= \sum_{j=0}^{i} (-1)^{i-j} \binom{r-j}{i-j} f_{j-1}(\Delta(L))
\end{aligned}$$

which is equal to $h_i(\Delta(L))$ by formula 1.1. □

As the above proposition shows, studying the flag h-vector may give insight into the behavior of the original h-vector.

The flag h-vector of the order complex of a geometric lattice L can also be given a combinatorial interpretation, but presenting this interpretation requires the development of some further terminology. Given a geometric lattice L, label the atoms of L by $\{1, 2, \ldots, n\}$. Then for any edge (x, y) in the Hasse diagram of L, label (x, y) by $\lambda(x, y) = j$ where j is the minimal atom such that $x \vee j = y$. As noted in [NS], this labeling induces a lexicographic ordering on the facets of $\Delta(L)$, where each facet F is given a distinct labeling $\lambda(F)$ as an ordered tuple of the edges of which it is comprised (from $\hat{0}$ to $\hat{1}$). This ordering is called the *minimal labeling* of the facets of L. There are two theorems regarding such labelings presented in [NS] which are provided below without proof (see [NS] for further details). One is the so-called Switching Lemma:

Lemma 2.2. (Switching Lemma) *Let $(b_1, \ldots, b_i, b_{i+1}, \ldots, b_{r+1})$ be a minimal labeling of a facet of $\Delta(L)$. If $b_i < b_{i+1}$, then $(b_1, \ldots, b_{i+1}, b_i, \ldots, b_{r+1})$ is also a minimal labeling of a facet of $\Delta(L)$.*

The other gives a combinatorial interpretation of the flag h-vector:

Proposition 2.3. *For L a geometric lattice of rank $r+1$ and $S \subseteq [r]$, we have:*

$$h_S(L) = \left|\{\hat{0} = x_0 < x_1 < \ldots < x_{r+1} = \hat{1} : \{i : \lambda(x_{i-1}, x_i) > \lambda(x_i, x_{i+1})\} = S\}\right|.$$

That is, $h_S(L)$ counts the number of maximal chains of L with labels having the descent set S. The proposition above actually applies to the flag h-vector of the order complex of any lattice whose facets have been endowed with what is known as an EL-labeling. For a definition of EL-labelings and a proof of

why these labelings lead to the above combinatorial interpretation of the flag h-vector, see [St] (note that in this source, EL-labelings are referred to as R-labelings).

In order to study this flag h-vector, Nyman and Swartz turned to the permutation group S_{r+1}. Specifically, they investigated a partial ordering $\leq_w$ on S_{r+1} known as *the weak ordering,* (sometimes called the *weak Bruhat ordering*) which is defined as follows: Given $\pi_1 = a_1 a_2 \dots a_{r+1}$, $\pi_2 = b_1 b_2 \dots b_{r+1} \in S_{r+1}$ (where $a_i = \pi_1(i)$ and $b_i = \pi_2(i)$), we say that $\pi_1 \leq_w \pi_2$ if and only if π_2 can be obtained from π_1 by successively transposing adjacent elements a_i and a_{i+1} such that $a_i < a_{i+1}$.

A foundational result upon which this research is based comes from examining the weak ordering on the permutation group S_{r+1} given the above-listed properties of the minimal labeling of a geometric lattice L. First, given $S \subseteq [r]$, define $D(S) = \{\pi = a_1 a_2 \dots a_{r+1} \in S_{r+1} : a_i > a_{i+1} \Leftrightarrow i \in S\}$, i.e. $D(S)$ is the set of all permutations in S_{r+1} whose descent set is S. Then for $T, S \subseteq [r]$, we say S *dominates* T if there exists an injection $\phi : D(T) \to D(S)$ such that $\pi \leq_w \phi(\pi)$ for all $\pi \in D(T)$. Given this terminology, the following theorem falls easily out of the combinatorial interpretation of h_S, the Switching Lemma and the definition of the weak ordering:

Theorem 2.4. *Let L be a geometric lattice of order $r+1$ and let $\Delta(L)$ be the corresponding order complex. Then for $S, T \subseteq [r]$, S dominates T implies $h_T(L) \leq h_S(L)$.*

The bulk of the results in this paper have been obtained in trying to use the above theorem to provide a combinatorial proof of Theorem 1.2. The following section turns to the results obtained thus far.

3. Results concerning the weak order in S_{r+1}

The first result is an extension of a proposition given in [NS]. First, let $S_1, \dots, S_n$, $T_1, \dots, T_m \subseteq [r]$. We say that $S_1, \dots, S_n$ *dominates* $T_1, \dots, T_m$ if there exists an injection

$$\phi : D(T_1) \cup \dots \cup D(T_m) \to D(S_1) \cup \dots \cup D(S_n)$$

such that $\pi \leq_w \phi(\pi)$ for all $\pi \in D(T_1) \cup \dots \cup D(T_m)$. It is easy to see that a proof completely analogous to that of Theorem 2.4 implies that if $S_1, \dots, S_n$ *dominates* $T_1, \dots, T_m$, then $h_{T_1} + \dots + h_{T_m} \leq h_{S_1} + \dots + h_{S_n}$.

Proposition 3.1. (Subset Principle) *If $S_1, \dots, S_n$* dominates *$T_1, \dots, T_m$, then for each $i \in [m]$ there exists $j \in [n]$ such that $T_i \subseteq S_j$.*

Proof. Let $S_1, \dots, S_n, T_1, \dots, T_m \subseteq [r]$ as above, and let $i \in [m]$. Say

$$T_i = \{c_1, c_2, \dots, c_l\}, \quad \text{with } c_1 < c_2 < \dots < c_l.$$

Let $\pi \in S_{r+1}$ be the permutation with descent set T_i such that:

$$\begin{aligned}\{\pi(1), \pi(2), \ldots, \pi(c_1)\} &= \{r+2-c_1, \ldots, r+1\} \\ \{\pi(c_1+1), \pi(c_1+2), \ldots, \pi(c_2)\} &= \{r+2-c_2, \ldots, r+1-c_1\} \\ &\vdots \\ \{\pi(c_l+1), \pi(c_l+2), \ldots, \pi(r+1)\} &= \{1, \ldots, r+1-c_l\}\end{aligned}$$

It is clear by the definition of $\leq_w$ that for such a π, $\pi \leq_w \pi'$ implies that the descent set of π' must contain $\{c_1, c_2, \ldots, c_l\}$ as a subset. Hence:

$$\begin{aligned}\phi(\pi) \in D(S_1) \cup \ldots \cup D(S_n) &\Rightarrow \phi(\pi) \in D(S_j) \text{ for some } j \\ &\Rightarrow S_j \supseteq \{c_1, c_2, \ldots, c_l\} = T_i.\end{aligned}$$

□

The motivation for seeking such conditions for domination is that it allows us to relate the elements of the flag h-vector of the order complex of a geometric lattice by inequalities. This in turn may reveal inequalities regarding the standard h-vector. If, for instance, one could find a distinct $S \subseteq [r]$ of cardinality $i+1$ for each $T \subseteq [r]$ of cardinality i (where $i < \lfloor \frac{r}{2} \rfloor$) such that S dominates T, then by Theorem 2.4 this would imply

$$h_i = \sum_{|T|=i} h_T \leq \sum_{|S|=i+1} h_S = h_{i+1}$$

thus proving one of the inequalities of Theorem 1.2. As was discovered early on, however, such a result is not possible. For instance, for the descent set $T = \{2, 4\} \subseteq [5]$, there exists no descent set $S \subseteq [5]$ such that $|S| = 3$ and such that S dominates T. The reason for this is simple to state: the only subset $S \subseteq [5]$ with $|S| = 3$ such that $|D(T)| \leq |D(S)|$ is $S = \{1, 3, 5\}$. But as $T \not\subseteq S$, S can not dominate T (by the Subset Principle).

This fact does not destroy all hope of obtaining useful results from this method. Note that one only needs to prove something somewhat weaker than the above, failed attempt. Specifically, one would be able to prove the first inequality from Theorem 1.2 if for $i < \lfloor \frac{r}{2} \rfloor$ one could prove that the collection of all i-subsets of $[r]$ is dominated by the collection of all $(i+1)$-subsets of $[r]$. The failure above simply demonstrates the fact that an injection corresponding to such a domination can not always be built up from the injections corresponding to individual sets of cardinality $i+1$ dominating individual sets of cardinality i. So the next logical question is this: by additionally looking at *pairs* of i-subsets dominated by pairs of $(i+1)$-subsets, can the corresponding injections be extended to a single injection demonstrating that the collection of all i-subsets of $[r]$ is dominated by the collection of all $(i+1)$-subsets of $[r]$ (for $i < \lfloor \frac{r}{2} \rfloor$)?

As the following theorem will show, the answer is an unfortunate "No."

Theorem 3.2. *Let $T_1 = \{2, 4, \ldots, 2n\} \subseteq [2n+1]$ for some $n \geq 2$. Then there exist no $T_2, S_1, S_2 \subseteq [2n+1]$ such that:*

$$(1)\, |T_2| = n, |S_1| = |S_2| = n+1$$
$$(2)\, S_1, S_2 \text{ dominates } T_1, T_2$$

and such that the injection of the corresponding domination can be extended to an injection demonstrating that the collection of all n-subsets of $[2n+1]$ is dominated by the set of all $(n+1)$-subsets of $[2n+1]$.

Proof. Assume that there *do* exist such sets $T_2, S_1, S_2 \subseteq [2n+1]$.

First, note that by symmetry it is easy to see that the number of n-subsets of $[2n+1]$ is equal to the number of $(n+1)$-subsets of $[2n+1]$. Therefore if the injection $\phi : D(T_1) \cup D(T_2) \to D(S_1) \cup D(S_2)$ (guaranteed by the fact that S_1, S_2 dominates T_1, T_2) can really be extended as assumed, then we must have:

$$|D(T_1) \cup D(T_2)| = |D(S_1) \cup D(S_2)|\,.$$

Due to the results of Niven in [Ni]:

$$|D(S)| \leq |D(T_1)| \quad \text{for all } S \subseteq [2n+1], |S| = n+1, \text{ and}$$
$$|D(S)| = |D(T_1)| \Leftrightarrow S = \{1, 3, \ldots, 2n+1\}$$

which, when coupled with the fact that $|D(T_1) \cup D(T_2)| = |D(S_1) \cup D(S_2)|$, implies w.l.o.g. that:

$$S_1 = \{1, 3, \ldots, 2n+1\}.$$

But by the Subset Principle, $T_1 \not\subseteq S_1$ implies that $T_1 \subseteq S_2$, and hence:

$$S_2 = \{1, 2, 4, \ldots, 2n\}, \{2, 3, 4, \ldots, 2n\}, \ldots, \text{ or } \{2, 4, \ldots, 2n, 2n+1\}$$

There are two cases:

Case I: $S_2 = \{1, 2, 4, \ldots, 2n\}, \{2, 3, 4, \ldots, 2n\}, \ldots, \text{ or } \{2, 4, \ldots, 2n-1, 2n\}$

Let A denote the set of permutations $\pi \in S_{2n+2}$ with descent set T_1 such that:

$$\{\pi(1), \pi(2), \ldots, \pi(2n)\} = \{3, 4, \ldots, 2n+2\}$$
$$\{\pi(2n+1), \pi(2n+2)\} = \{1, 2\}.$$

Clearly for any $\pi \in A$ and $\pi' \in S_{2n+2}$ such that $\pi \leq_w \pi'$, the descent set of π' contains $\{2n\}$. Thus for all $\pi \in A$, $\phi(\pi) \subseteq D(S_2)$ and hence ϕ gives us an injection from A to $D(S_2)$.

Next note that for $\pi \in A$, $\pi \in D(T_1)$ implies that $\pi(2n+1) = 1$ and $\pi(2n+2) = 2$. And hence for $\phi(\pi) \in D(S_2)$:

$$(\phi(\pi))(2n+1) = 1, \quad (\phi(\pi))(2n+2) = 2.$$

So if B is defined to be the set of permutations $\pi \in S_{2n+2}$ with descent set S_2 such that

$$\{\pi(1), \pi(2), \ldots, \pi(2n)\} = \{3, 4, \ldots, 2n+2\}$$
$$\{\pi(2n+1), \pi(2n+2)\} = \{1, 2\}.$$

then we see that ϕ gives an injection from A to B. But clearly there exist obvious bijections between A and $D(T_1 - \{2n\})$, $T_1 - \{2n\} \subseteq [2n-1]$, and between B and $D(S_2 - \{2n\})$, $S_2 - \{2n\} \subseteq [2n-1]$. But again due to Niven's result from [Ni]:

$$|D(T_1 - \{2n\})| > |D(S_2 - \{2n\})|$$
$$\Rightarrow |A| > |B|$$

which contradicts the fact that $\phi|_A : A \to B$ is an injection.

Case II: $S_2 = \{2, 4, \ldots, 2n, 2n+1\}$.

Similar to Case I, let A denote the set of all $\pi \in S_{2n+2}$ with descent set T_1 such that

$$\{\pi(1), \pi(2)\} = \{2n+1, 2n+2\}$$
$$\{\pi(3), \pi(4), \ldots, \pi(2n+2)\} = \{1, 2, \ldots, 2n\}$$

and B to be the set of all $\pi \in S_{2n+2}$ with descent set S_2 such that

$$\{\pi(1), \pi(2)\} = \{2n+1, 2n+2\}$$
$$\{\pi(3), \pi(4), \ldots, \pi(2n+2)\} = \{1, 2, \ldots, 2n\}.$$

A similar argument to that employed in Case I shows that $|A| > |B|$ while $\phi|_A : A \to B$ is an injection, which is a contradiction.

Hence the original assumption was incorrect, and there can be no such sets $T_2, S_1, S_2 \subseteq [2n+1]$. □

Despite this discouraging theorem, we shall see in Section 6 that appealing to triples of descent sets dominated by triples of descent sets can sometimes yield more useful results.

The remainder of this section turns to what can be said regarding the weak order on S_{r+1}. In Sections 7 and 8 we shall see these results have applications not just to the order complex of geometric lattices, but also to a host of other objects. We begin by presenting the following lemma, used to prove Theorem 3.4 below.

Lemma 3.3. *Let $j, r+1 \in \mathbb{N}$ with $j \leq \lfloor \frac{r}{2} \rfloor$. Let X be any collection of j-subsets of $[r+1]$ such that $|X| \leq \binom{r+1}{j} - 1$, and define*

$$Y = \{V \subseteq [r+1] : |V| = r+1-j \text{ and } U \subseteq V \text{ for some } U \in X\}.$$

Then $|Y| \geq |X| + 1$.

Proof. The elements of $X \cup Y$ can be viewed as the vertices of a directed, bipartite graph G defined so that there is an edge from $U \in X$ to an element $V \in Y$ if and only if $U \subseteq V$.

We begin by counting the number of edges of G. For each $U \in X$, U is a subset of exactly $\binom{r+1-j}{r+1-2j} = \binom{r+1-j}{j}$ $(r+1-j)$-subsets of $[r+1]$. Each such $(r+1-j)$-subset is by definition an element of Y, and hence there are exactly $\binom{r+1-j}{j}$ edges from U to elements of Y. Thus there are exactly $|X|\binom{r+1-j}{j}$ edges exiting the vertex set X in G.

Clearly this means that there must be $|X|\binom{r+1-j}{j}$ edges entering the vertex set Y in G, by definition of the graph G. Now for each $V \in Y$, V is a superset of exactly $\binom{r+1-j}{j}$ j-subsets of $[r+1]$. Each such subset may or may not be an element of X, which shows that there are at most $\binom{r+1-j}{j}$ edges from the vertex set X to the vertex V. Thus the number of edges entering the vertex set Y is at most $|Y|\binom{r+1-j}{j}$, i.e.

$$|X|\binom{r+1-j}{j} \leq |Y|\binom{r+1-j}{j}$$
$$\Rightarrow |X| \leq |Y|$$

where equality holds only if each j-subset of each element of Y is an element of X. Thus to complete the proof that $|Y| \geq |X|+1$ (i.e. $|Y| > |X|$), we need only exhibit a single element of Y having a j-subset V such that $V \notin X$.

As $|X| \leq \binom{r+1}{j} - 1$, we can choose some j-subset α of $[r+1]$ such that $\alpha \notin X$. Now let $\beta = \{b_1, b_2, \ldots, b_{r+1-j}\}$ be any $(r+1-j)$-subset of $[r+1]$ such that $\alpha \subseteq \beta$. Finally, let $\beta' = \{a_1, a_2, \ldots, a_{r+1-j}\}$ be any element of Y. Define a sequence $\beta_0, \beta_1, \ldots, \beta_{r+1-j}$ of $(r+1-j)$-subsets of $[r+1]$ as follows:

$$\begin{aligned}
\beta_0 = \beta' &= \{a_1, a_2, \ldots, a_{r+1-j}\} \\
\beta_1 &= \{b_1, a_2, \ldots, a_{r+1-j}\} \\
\beta_2 &= \{b_1, b_2, \ldots, a_{r+1-j}\} \\
&\vdots \\
\beta_{r+1-j} = \beta &= \{b_1, b_2, \ldots, b_{r+1-j}\}
\end{aligned}$$

Note that $\beta_0 \in Y$ and that β_{r+1-j} has a j-subset V $(= \alpha)$ such that $V \notin X$.

Now let k be the smallest index such that β_k has a j-subset V such that $V \notin X$. This is well defined as the index $r+1-j$ satisfies this property. In order to complete the proof, all that needs to be shown is that β_k is an element of Y.

If $k = 0$, then by definition $\beta_0 = \beta' \in Y$. So assume $k \neq 0$. Note the following fact: if every j-subset of β_i is an element of X for some $i \in \{0, 1, \ldots, r-j\}$, then $\beta_{i+1} \in Y$. This is because any j-subset U of $\beta_{i+1} - \{b_{i+1}\}$ is a j-subset of β_i, and hence is an element of X (note that such a set U exists by the fact that $j \leq \lfloor \frac{r}{2} \rfloor$). So β_{i+1} is an $(r+1-j)$-subset of $[r+1]$ such that there exists a $U \subseteq \beta_{i+1}$ with $U \in X$, i.e. β_{i+1} is an element of Y. And so as every j-subset of β_{k-1} is an element of X by the definition of k, this demonstrates that $\beta_k \in Y$ and completes the proof. □

The above lemma can be used to prove the following theorem concerning descent set dominance. Note that the following theorem is a specific case of a more general conjecture to be presented in Section 6.

Theorem 3.4. *Let $r+1 \in \mathbb{N}$ and $j \in [r]$. Let $\{j\} \circ \beta \subseteq [r]$ be defined by*

$$\{j\} \circ \beta = \{i \in [r] : r - i + 1 \neq j\}.$$

Then if $\{j\} \subseteq \{j\} \circ \beta$ (equivalently if $j \neq \frac{r+1}{2}$), $\{j\}$ is dominated by $\{j\} \circ \beta$.

Proof. The theorem is first proved for the case $j \leq \lfloor \frac{r}{2} \rfloor$.

The first claim is that the set $D(\{j\})$ is in bijective correspondence with the set $X = \{U \subseteq [r+1] : |U| = j, U \neq [j]\}$. To see why this is so, define the function f on $D(\{j\})$ by

$$f(a_1 a_2 \dots a_{r+1}) = \{a_1, a_2, \dots, a_j\}$$

The claim is that f is a bijection from X to $D(\{j\})$. First note that for any $\pi = a_1 a_2 \dots a_{r+1} \in D(\{j\})$, the fact that $a_j > a_{j+1}$ implies that $\{a_1, a_2, \dots, a_j\} \neq [j]$. Hence $f(\pi) \in X$, i.e.

$$f : D(\{j\}) \to X.$$

To show that f is surjective, let $U \in X$. Define $\pi \in S_{r+1}$ by $\pi = a_1 a_2 \dots a_{r+1}$, where

$$\{a_1, a_2, \dots, a_j\} = U \text{ with } a_1 < a_2 < \dots < a_j, \text{ and}$$
$$\{a_{j+1}, \dots, a_{r+1}\} = [r+1] - U \text{ with } a_{j+1} < \dots < a_{r+1}.$$

As $U = \{a_1, a_2, \dots, a_j\} \neq [j]$, we must have that $a_j > a_{j+1}$. Hence $D(\pi) = \{j\}$, and thus $\pi \in D(\{j\})$. As $f(\pi) = \{a_1, a_2, \dots, a_j\} = U$, this demonstrates that f is a surjective map.

To show that f is injective, assume that $f(\pi_1) = f(\pi_2)$ for $\pi_1, \pi_2 \in D(\{j\})$. Denote $\pi_1 = a_1 a_2 \dots a_{r+1}$ and $\pi_2 = b_1 b_2 \dots b_{r+1}$. Then the condition $f(\pi_1) = f(\pi_2)$ gives

$$\{a_1, \dots, a_j\} = \{b_1, \dots, b_j\}, \text{ and}$$
$$\{a_{j+1}, \dots, a_{r+1}\} = [r+1] - f(\pi_1) = [r+1] - f(\pi_2) = \{b_{j+1}, \dots, b_{r+1}\}$$

But the conditions that $a_1 < \dots < a_j$, $a_{j+1} < \dots < a_{r+1}$, $b_1 < \dots < b_j$ and $b_{j+1} < \dots < b_{r+1}$ imply that

$$a_i = b_i \text{ for all } i \in \{1, \dots, j\} \cup \{j+1, \dots, r+1\}$$

i.e. $\pi_1 = \pi_2$. Hence f is injective and thus bijective, as desired.

The second claim is that the set $D(\{j\} \circ \beta)$ is in bijective correspondence with the set $Y = \{V \subseteq [r+1] : |V| = r+1-j, U \neq \{j+1, \dots, r+1\}\}$. The function g defined on $D(\{j\} \circ \beta)$ by

$$g(a_1 a_2 \dots a_{r+1}) = \{a_1, a_2, \dots, a_{r+1-j}\}$$

yields the appropriate bijection. This follows by reasoning similar to the above, recognizing that $D(\{j\} \circ \beta)$ is the set of all permutations in S_{r+1} whose ascent set (rather than descent set) is $\{n-j\}$.

The final claim is that for $\pi_1 \in D(\{j\})$ and $\pi_2 \in D(\{j\} \circ \beta)$, $f(\pi_1) \subseteq g(\pi_2)$ implies that $\pi_1 \leq_w \pi_2$. To see why this is so, let $\pi_1 = a_1 a_2 \dots a_{r+1} \in D(\{j\})$, $\pi_2 = b_1 b_2 \dots b_{r+1} \in D(\{j\} \circ \beta)$ and assume that $f(\pi_1) = \{a_1, \dots, a_j\}$ is a subset of $g(\pi_2) = \{b_1, \dots, b_{r+1-j}\}$. As $a_1 < \dots < a_j$, repeated application of the switching algorithm described earlier yields:

$$\pi_1 \leq_w \pi_1' := a_j a_{j-1} \dots a_1 a_{j+1} a_{j+2} \dots a_{r+1}.$$

As $\{a_1, \ldots, a_j\} \subseteq \{b_1, \ldots, b_{r+1-j}\}$, we can write

$$a_1 = b_{i_1}, a_2 = b_{i_2}, \ldots, a_j = b_{i_j}$$

for some indices $i_1, \ldots, i_j$. Hence $\pi_1' = b_{i_j} b_{i_{j-1}} \ldots b_{i_1} a_{j+1} a_{j+2} \ldots a_{r+1}$. As $b_{i_j} > \ldots > b_{i_1}$, $b_1 > \ldots > b_j$ and $a_{j+1} < \ldots < a_{r+1}$, further application of the switching algorithm yields:

$$\pi_1' \leq_w \pi_1'' := b_1 b_2 \ldots b_{r+1-j} b_{r+1} b_r \ldots b_{r+2-j}$$

where here π_1'' was obtained from π_1' by "switching" all those elements belonging to both $\{a_{j+1}, \ldots, a_{r+1}\}$ and $\{b_1, \ldots, b_{r+1-j}\} - \{a_1, \ldots, a_j\}$ to the left as far as possible. Finally, as $b_{r+1} < \ldots < b_{r+2-j}$, further application of the switching algorithm yields:

$$\pi_1'' \leq_w b_1 b_2 \ldots b_{r+1-j} b_{r+2-j} b_{r+3-j} \ldots b_{r+1} = \pi_2.$$

Thus $\pi_1 \leq_w \pi_2$, as desired.

In order to prove the theorem, it must be shown that there exists a matching from $D(\{j\})$ to $D(\{j\} \circ \beta)$ on the graph G whose vertex set is $D(\{j\}) \cup D(\{j\} \circ \beta)$ and whose edges are defined by: $\pi_1 \in D(\{j\})$ is connected to $\pi_2 \in D(\{j\} \circ \beta)$ if and only if $\pi_1 \leq_w \pi_2$. By what has been shown thus far, such a matching is guaranteed if there exists a matching from X to Y on the graph G' whose vertex set is $X \cup Y$ and whose edges are defined by: $U \in X$ is connected to $V \in Y$ if and only if $U \subseteq V$. Thus we turn to proving the existence of such a matching in G'.

Let $X_1 \subseteq X$ and define

$$\Gamma(X_1) = \{V \in Y : \text{ there exists } U \in X_1 \text{ such that } U \subseteq V\},$$

that is, $\Gamma(X_1)$ is the set of neighbors of the vertex set X_1 in G'. By definition, X_1 is a collection of j-subsets of $[r+1]$ such that $|X_1| \leq |X| = \binom{r+1}{j} - 1$. If we define

$$Y_1 = \{V \subseteq [r+1] : |V| = r + 1 - j \text{ and } U \subseteq V \text{ for some } U \in X\},$$

then by Lemma 3.3 we have that $|Y_1| \geq |X_1| + 1$. But clearly $\Gamma(X_1) = Y_1 \cap Y$, and hence $\Gamma(X_1) = Y_1$ or $\Gamma(X_1) = Y_1 - \{\{j+1, \ldots, r+1\}\}$. So we have:

$$|\Gamma(X_1)| \geq |Y_1| - 1 \geq |X_1|.$$

As $|\Gamma(X_1)| \geq |X_1|$ for all $X_1 \subseteq X$, Hall's Marriage Theorem guarantees the existence of a matching from X to Y in G'. So by what was stated above, this proves the theorem for $j \leq \lfloor \frac{r}{2} \rfloor$.

For $j \geq \lfloor \frac{r+2}{2} \rfloor$, the proof is similar. In this case, one can show that $D(\{j\})$ is in bijective correspondence with the set Y under the function f' defined by

$$f'(a_1 a_2 \ldots a_{r+1}) = \{a_{j+1}, a_{j+2}, \ldots, a_{r+1}\},$$

and that $D(\{j\} \circ \beta)$ is in bijective correspondence with the set X under the function g' defined by

$$g'(a_1 a_2 \ldots a_{r+1}) = \{a_{r+2-j}, a_{r+3-j}, \ldots, a_{r+1}\}.$$

Then in a manner similar to the above, it can be shown that for $\pi_1 \in D(\{j\})$ and $\pi_2 \in D(\{j\} \circ \beta)$, $f'(\pi_1) \subseteq g'(\pi_2)$ implies that $\pi_1 \leq_w \pi_2$. The rest of the proof proceeds analogously. □

A corollary to the above theorem is the following application to the h-vector of the order complex of geometric lattices.

Corollary 3.5. *Let L be a geometric lattice of order $r+1$, r even, and let $\Delta(L)$ be the corresponding order complex. Then $h_1(\Delta(L)) \leq h_{r-1}(\Delta(L))$.*

Proof. As r is even, $\{j\} \subseteq \{j\} \circ \beta$ for all $j \in [r]$. Then by the previous theorem:

$$\{j\} \text{ is dominated by } \{j\} \circ \beta \text{ for all } j \in [r],$$

which implies by Theorem 2.4 that

$$h_{\{j\}}(L) \leq h_{\{j\}\circ\beta}(L) \text{ for all } j \in [r].$$

Finally, as $h_1(\Delta(L)) = \sum_{j\in[r]} h_{\{j\}}(L)$ and as $h_{r-1}(\Delta(L)) = \sum_{j\in[r]} h_{\{j\}\circ\beta}(L)$ (by Proposition 2.1), this implies that

$$h_1(\Delta(L)) \leq h_{r-1}(\Delta(L))$$

□

This section concludes with another theorem concerning subset dominance. Note that this theorem is another special case of the conjecture referred to in presenting Theorem 3.4.

Theorem 3.6. *Let $S = \{1, 2, \ldots, j\}$ be a subset of $[r]$ and let $S \circ \beta \subseteq [r]$ be defined by*

$$S \circ \beta = \{i \in [r] : r - i + 1 \notin T\}.$$

Note that $S \circ \beta = \{1, 2, \ldots, r-j\}$. Then if $S \subseteq S \circ \beta$ (equivalently, if $j \leq \lfloor \frac{r}{2} \rfloor$), $S \circ \beta$ dominates S.

Proof. The proof of this theorem is quite similar to that of Theorem 3.4, and as such will simply be outlined below.

It can be shown that $D(S)$ is in bijective correspondence with the set of j-subsets of $[r+1] - \{1\}$ under the function f defined by

$$f(a_1 a_2 \ldots a_{r+1}) = \{a_1, a_2, \ldots, a_j\} \text{ for } a_1 a_2 \ldots a_{r+1} \in D(S).$$

It can also be shown that $D(S \circ \beta)$ is in bijective correspondence with the set of $(r-j)$-subsets of $[r+1] - \{1\}$ under the function g defined by

$$f(a_1 a_2 \ldots a_{r+1}) = \{a_1, a_2, \ldots, a_{r-j}\} \text{ for } a_1 a_2 \ldots a_{r+1} \in D(S).$$

Then just as in Theorem 3.4, it can be shown that $f(\pi_1) \subseteq g(\pi_2)$ implies that $\pi_1 \leq_w \pi_2$ for $\pi_1 \in D(S), \pi_2 \in D(S \circ \beta)$. So in order to find an injection $\phi : D(S) \to D(S \circ \beta)$ such that $\pi \leq_w \phi(\pi)$ for all $\pi \in D(S)$, it suffices to find an injection ϕ' from the set of all j-subsets of $[r+1] - \{1\}$ to the set of all $(r-j)$-subsets of $[r+1] - \{1\}$ such that $U \subseteq \phi'(U)$ for all such j-subsets U. But as can easily be shown by Hall's Marriage Theorem, one can always find a bijection from the set of j-subsets to the set of $(r-j)$-subsets of an r element

set such that each j-subset is a subset of its image under the bijection. Thus the injection ϕ' exists, completing the proof. □

4. Results concerning the weak order in B_{r+1}

As has been shown, the permutation group S_{r+1} endowed with the weak order is an interesting object to investigate as it is intimately tied to the h-vector of geometric lattices. As we shall see in Sections 7 and 8, the properties of this ordering on S_{r+1} are also linked to inequalities of the h-vector of distributive and supersolvable lattices. In fact, due to the fundamental structure of distributive lattices (as revealed in section 7), the weak order on S_{r+1} can even been seen as linked to certain properties of arbitrary finite posets. Because of the importance of the weak order, this section takes the time to develop and investigate the properties of the weak order on another group: B_{r+1}.

The group B_{r+1}, known as the *signed permutation group,* is defined as follows. Each element of B_{r+1} is of the form $\pi = a_1 a_2 \dots a_{r+1}$, where

$$\{|a_1|, |a_2|, \dots, |a_{r+1}|\} = [r+1]$$
$$\text{and } a_i \in \{-r-1, -r, \dots, -1, 1, \dots, r, r+1\} \text{ for } i \in [r+1].$$

That is, each element of B_{r+1} is simply an element $\pi' = a'_1 a'_2 \dots a'_{r+1}$ of S_{r+1} for which some combination of the a'_i have been multiplied by -1. As in the case of S_{r+1}, there exists a partial ordering on the elements of B_{r+1} known as the weak ordering (sometimes called the weak Bruhat ordering). This ordering will also be denoted by the symbol $\leq_w$, although it should be clear from context which group is being referred to whenever this symbol is used. The definition of this ordering will be presented in two equivalent ways.

Let $\pi_1 = a_1 a_2 \dots a_{r+1}$, $\pi_2 = b_1 b_2 \dots b_{r+1} \in B_{r+1}$. Let $i, j \in [r+1]$ with $i < j$. Define $x(i,j) = \min\{|a_i|, |a_j|\}$, $y(i,j) = \max\{|a_i|, |a_j|\}$. Next, define two ordered pairs: $\alpha(i,j) = (a_i, a_j)$ and $\beta(i,j) = (b_{k_1}, b_{k_2})$ where k_1 is the minimum index such that $|b_{k_1}| \in \{|a_i|, |a_j|\}$ and k_2 is the maximum index such that $|b_{k_2}| \in \{|a_i|, |a_j|\}$. Finally, define a finite poset $P(i,j)$ whose Hasse diagram appears on the following page.

Then we say that $\pi_1 \leq_w \pi_2$ in B_{r+1} if $\alpha(i,j) \leq \beta(i,j)$ in the partial ordering on $P(i,j)$ for all $i, j \in [r+1]$ with $i < j$.

Example 4.1. Let $\pi_1 = -1, 2$ and $\pi_2 = -2, -1$ be elements of B_2. The only $i, j \in [2]$ such that $i < j$ are $i = 1$ and $j = 2$. Then by definition:

$$x(1,2) = \min\{|-1|, |2|\} = 1,$$
$$y(1,2) = \max\{|-1|, |2|\} = 2,$$
$$\alpha(1,2) = (-1, 2) = (-x(1,2), y(1,2)),$$
$$\beta(1,2) = (-2, 1) = (-y(1,2), -x(1,2)).$$

And as $(-x(1,2), y(1,2)) < (-y(1,2), -x(1,2))$ in $P(1,2)$, this implies that $\pi_1 \leq_w \pi_2$ in B_2.

The above definition of the weak order in B_{r+1} is hard to work with in practice (though it is well suited for use in a computer program, as will be discussed in

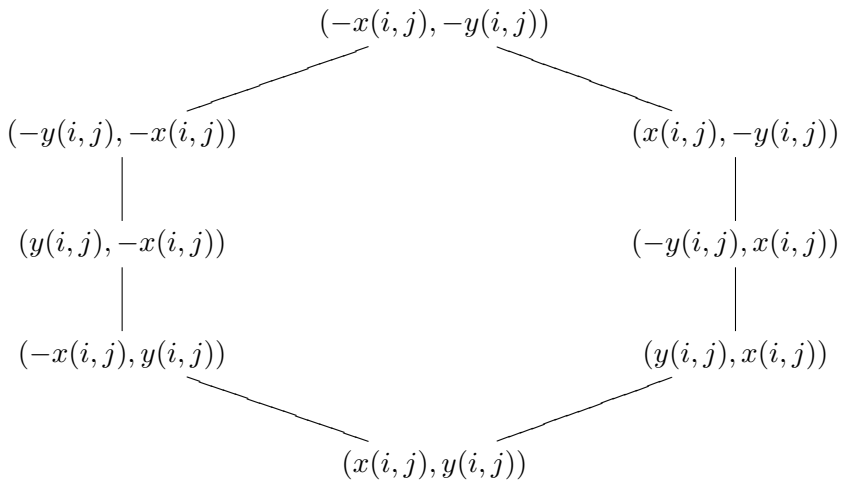

FIGURE 1. Hasse diagram for the poset $P(i,j)$.

Section 5). A more desirable definition would be something analogous to the switching procedure described earlier to define the weak order in S_{r+1}. So again, let $\pi_1 = a_1 a_2 \dots a_{r+1}$, $\pi_2 = b_1 b_2 \dots b_{r+1} \in B_{r+1}$. The claim is that $\pi_1 \leq_w \pi_2$ in B_{r+1} if and only if π_2 can be obtained from π_1 by repeated application of so-called "legal moves" of either of the following forms:

(1) Transposing a_i and a_{i+1}, if $a_i < a_{i+1}$
(2) Changing a_1 to $-a_1$, if $a_1 > 0$

(assuming that after each move, the resulting permutation is reindexed in the natural way). One direction of the equivalence of these two definitions is easy to see. Namely if π_2 can be obtained from π_1 by repeated application of the legal moves above, then clearly $\pi_1 \leq_w \pi_2$. Showing the other direction will require the following lemma.

Lemma 4.2. *Let $\pi_1 = a_1 a_2 \dots a_{r+1}$, $\pi_2 = b_1 b_2 \dots b_{r+1} \in B_{r+1}$ be two signed permutations such that $\pi_1 \leq_w \pi_2$. If $\pi_1 \neq \pi_2$, then there exits $\pi_3 \in B_{r+1}$, obtainable from π_1 by applying one of the two legal moves described above, such that*

$$\pi_1 <_w \pi_3 \leq_w \pi_2.$$

Proof. This proof makes use of the notation that was developed earlier in defining the weak order on B_{r+1}. Additionally, for $i \in [r+1]$, the index $k \in [r+1]$ such that $|b_k| = |a_i|$ will be denoted by $g(i)$. There are three possible cases:

Case I ($a_1 < 0, b_{g(1)} > 0$): Because $\pi_1 \leq_w \pi_2$, $\alpha(1,2) = (a_1, a_2) \leq \beta(1,2)$ in the partial ordering on $P(1,2)$. By examination of $P(1,2)$, it is evident that $(a_1, a_2) \leq \beta(1,2)$ in $P(1,2)$ and $a_1 < 0$ implies that $b_{g(1)} < 0$. This is a contradiction, and hence this case does not occur.

Case II ($a_1 > 0, b_{g(1)} < 0$): Because $\pi_1 \leq_w \pi_2$, $\alpha(1,n) = (a_1, a_n) \leq \beta(1,n)$ in $P(1,n)$ for all $n \in \{2,3,\dots,r+1\}$. By examination of $P(1,n)$, it is evident that if

$(a_1, a_n) \leq \beta(1,n)$ in $P(1,n)$ with $a_1 > 0$ and $b_{g(1)} < 0$, then $(-a_1, a_n) \leq \beta(1,n)$ in $P(1,n)$. Hence if we define $\pi_3 \in B_{r+1}$ by $\pi_3 = (-a_1)a_2 \dots a_{r+1}$, this shows that $\pi_3 \leq_w \pi_2$. But as π_3 is obtained from π_1 by one of the legal moves described earlier, $\pi_1 \leq_w \pi_3$. And as $\pi_1 \neq \pi_3$, $\pi_1 <_w \pi_3$ and the conclusion of the lemma follows in this case.

Case III ($a_1 = b_{g(1)}$): There are several things which may happen in this case. First, it might be that $a_i > a_{i+1}$ for all $i \in [r]$. Assume that this is the case. As $\pi_1 \leq_w \pi_2, \alpha(n, n+1) = (a_n, a_{n+1}) \leq \beta(n, n+1)$ in $P(n, n+1)$ for all $n \in [r]$. By examination of $P(n, n+1)$, it is evident that if $(a_n, a_{n+1}) \leq \beta(n, n+1)$ in $P(n, n+1)$ with $a_n > a_{n+1}$ and $a_n = b_{g(n)}$, then $g(n) < g(n+1)$ and $a_{n+1} = b_{g(n+1)}$. As $a_1 = b_{g(1)}$, induction on n yields that $g(1) < g(2) < \ldots < g(r+1)$ and $a_i = b_{g(i)}$ for $i \in [r+1]$. But this implies that $g(i) = i$ for all $i \in [r+1]$, and hence $a_i = b_i$ for $i \in [r+1]$. In other words, $\pi_1 = \pi_2$.

Second, it might be true that for all i such that $a_i < a_{i+1}$, $g(i) < g(i+1)$ and $a_{i+1} = b_{g(i+1)}$. Assume that this is the case, and let $n \in [r]$. If $a_n > a_{n+1}$ and $a_n = b_{g(n)}$, then by the reasoning of the preceding paragraph it follows that $g(n) < g(n+1)$ and $a_{n+1} = b_{g(n+1)}$. If $a_n < a_{n+1}$, then by assumption $g(n) < g(n+1)$ and $a_{n+1} = b_{g(n+1)}$. As $a_1 = b_{g(1)}$, induction on n implies that $g(1) < g(2) < \ldots < g(r+1)$ and $a_i = b_{g(i)}$ for $i \in [r+1]$. As before, this implies that $\pi_1 = \pi_2$.

Finally, it might be that there is an index j such that $a_j < a_{j+1}$ and either $g(j) > g(j+1)$ or $a_{j+1} \neq b_{g(j+1)}$. If this is the case, let j denote the minimum index for which this occurs. Then by the reasoning presented in the preceding two paragraphs, it can by shown that $a_i = b_{g(i)}$ for all $i \in [j]$. Now as $\pi_1 \leq_w \pi_2$, $\alpha(j, j+1) = (a_j, a_{j+1}) \leq \beta(j, j+1)$ in $P(j, j+1)$. But examination of $P(j, j+1)$ shows that as $a_j = b_{g(j)}$ and as either $g(j) > g(j+1)$ or $a_{j+1} \neq b_{g(j+1)}$, (a_{j+1}, a_j) is also less than or equal to $\beta(j, j+1)$ in $P(j, j+1)$. Hence if we define $\pi_3 \in B_{r+1}$ by $\pi_3 = a_1 a_2 \dots a_{j+1} a_j \dots a_{r+1}$, this shows that $\pi_3 \leq_w \pi_2$. As $a_j < a_{j+1}$, π_3 is obtained from π_1 by an application of one of the legal moves described above and so $\pi_1 \leq_w \pi_3$. As $\pi_1 \neq \pi_3$, $\pi_1 <_w \pi_3$ and the conclusion of the lemma follows in this case. □

Using this lemma, the following theorem may be proved.

Theorem 4.3. *Let $\pi_1, \pi_2 \in B_{r+1}$. Then $\pi_1 \leq_w \pi_2$ if and only if π_2 can by obtained from π_1 by repeated application of the legal moves described earlier.*

Proof. As mentioned before, one direction of this proof is trivial to show from the definition of the weak ordering on B_{r+1}. So it suffices to prove that if $\pi_1 \leq_w \pi_2$, then π_2 can be obtained from π_1 by repeated application of the two legal moves.

If $\pi_1 = \pi_2$, then there is nothing to show. If $\pi_1 <_w \pi_2$, then by Lemma 4.2 we can apply a legal move to π_1 to obtain a $\pi_3 \in B_{r+1}$ such that $\pi_1 <_w \pi_3 \leq_w \pi_2$. If $\pi_3 = \pi_2$, then we are done. If not, then we can apply Lemma 4.2 to π_3 and π_2 to obtain a new signed permutation larger than π_3 and less than or equal to π_2. We continue in this manner, repeatedly applying the results of Lemma 4.2 to each new permutation. If at any point we obtain the permutation π_2, then

we have shown that the permutation π_2 can be obtained from π_1 by repeated application of the two legal moves described above, and hence we have proved the theorem.

What if the permutation π_2 is never obtained? Then by Lemma 4.2 we can continue this process indefinitely to produce a chain in B_{r+1} of arbitary length. As B_{r+1} is a finite set, this is a contradiction. Thus π_2 must be obtainable from π_1 through repeated application of the legal moves. □

And so the two provided definitions of the weak order on B_{r+1} really are equivalent (note that in the literature, the latter definition is the one usually presented). As with S_{r+1}, the more algorithmic definition of the weak ordering will prove itself more useful in establishing facts about B_{r+1}.

One such fact is that the Subset Principle holds in B_{r+1} as well as in S_{r+1}. In order to prove this, however, it must first be established what is meant by descent sets and domination in B_{r+1}.

So given $S \subseteq \{0, 1, \ldots, r\}$, define

$$D_\sigma(S) = \{\pi = a_1 \ldots a_{r+1} \in B_{r+1} : a_i > a_{i+1} \Leftrightarrow i \in S, \text{ where } a_0 := 0\}.$$

That is, $D_\sigma(S)$ is the set of all signed permutations in B_{r+1} whose descent set is S (where a signed permutation $\pi = a_1 a_2 \ldots a_{r+1}$ is said to have a descent in the 0^{th} place if a_1 is negative). Note that the subscript σ is meant to indicate that what is being referred to is a collection of *signed* permutations, distinguishing this current object from the analogue for standard permutations ($D()$, presented earlier). Then analogous to the definition presented for standard permutations, we say that for $S_1, \ldots, S_n, T_1, \ldots, T_m \subseteq \{0, \ldots, r\}$, $S_1, \ldots, S_n$ *dominates* $T_1, \ldots, T_m$ in B_{r+1} if there exists an injection

$$\phi : D_\sigma(T_1) \cup \ldots \cup D_\sigma(T_m) \to D_\sigma(S_1) \cup \ldots \cup D_\sigma(S_n)$$

such that $\pi \leq_w \phi(\pi)$ in B_{r+1} for all $\pi \in D_\sigma(T_1) \cup \ldots \cup D_\sigma(T_m)$.

Theorem 4.4. (Subset Principle, B_{r+1}) *Let $S_1, \ldots, S_n, T_1, \ldots, T_m \subseteq \{0, \ldots, r\}$, and assume that $S_1, \ldots, S_n$ dominates $T_1, \ldots, T_m$ in B_{r+1}. Then for each $i \in [m]$ there exists $j \in [n]$ such that $T_i \subseteq S_j$.*

Proof. Let $S_1, \ldots, S_n, T_1, \ldots, T_m$ be as above, let $i \in [m]$, and let ϕ denote the implied injection. Say

$$T_i = \{c_1, c_2, \ldots, c_l\} \text{ with } c_1 < c_2 < \ldots < c_l.$$

There are two cases:

Case 1: ($c_1 = 0$) Then let $\pi \in B_{r+1}$ be the (unique) signed permutation with descent set T_i such that

$$\begin{aligned}
\{\pi(1), \pi(2), \ldots, \pi(c_1)\} &= \{-1, -2, \ldots, -c_1\} \\
\{\pi(c_1 + 1), \pi(c_1 + 2), \ldots, \pi(c_2)\} &= \{-c_1 - 1, -c_1 - 2, \ldots, -c_2\} \\
&\vdots \\
\{\pi(c_l + 1), \pi(c_l + 2), \ldots, \pi(r + 1)\} &= \{-c_l - 1, -c_l - 2, \ldots, -r - 1\}
\end{aligned}$$

Now let $\pi' \in B_{r+1}$ be any signed permutation such that $\pi \leq_w \pi'$ in B_{r+1}. It is clear by the algorithmic definition of the weak order in B_{r+1} presented earlier that:

$$\begin{gathered}\{\pi'(1), \pi'(2), \ldots, \pi'(c_1)\} = \{-1, -2, \ldots, -c_1\} \\ \{\pi'(c_1+1), \pi'(c_1+2), \ldots, \pi'(c_2)\} = \{-c_1-1, -c_1-2, \ldots, -c_2\} \\ \vdots \\ \{\pi'(c_l+1), \pi'(c_l+2), \ldots, \pi'(r+1)\} = \{-c_l-1, -c_l-2, \ldots, -r-1\}\end{gathered}$$

and hence $D_\sigma(\pi') \supseteq \{c_1, c_2, \ldots, c_l\} = T_i$. As $\phi(\pi) \in D(S_j)$ for some j with $\pi \leq_w \phi(\pi)$, this demonstrates that

$$T_i = \{c_1, c_2, \ldots, c_l\} \subseteq S_j.$$

Case 2: ($c_1 \neq 0$) Then let $\pi \in B_{r+1}$ be the (unique) signed permutation with descent set T_i such that

$$\begin{gathered}\{\pi(1), \pi(2), \ldots, \pi(c_1)\} = \{1, 2, \ldots, c_1\} \\ \{\pi(c_1+1), \pi(c_1+2), \ldots, \pi(c_2)\} = \{-c_1-1, -c_1-2, \ldots, -c_2\} \\ \vdots \\ \{\pi(c_l+1), \pi(c_l+2), \ldots, \pi(r+1)\} = \{-c_l-1, -c_l-2, \ldots, -r-1\}\end{gathered}$$

Now let $\pi' \in B_{r+1}$ be any signed permutation such that $\pi \leq_w \pi'$ in B_{r+1}. Similar to the above, it is clear by the algorithmic definition of the weak order in B_{r+1} that:

$$\begin{gathered}\{\pi'(1), \pi'(2), \ldots, \pi'(c_1)\} \subseteq \{\pm 1, \pm 2, \ldots, \pm c_1\} \\ \{\pi'(c_1+1), \pi'(c_1+2), \ldots, \pi'(c_2)\} = \{-c_1-1, -c_1-2, \ldots, -c_2\} \\ \vdots \\ \{\pi'(c_l+1), \pi'(c_l+2), \ldots, \pi'(r+1)\} = \{-c_l-1, -c_l-2, \ldots, -r-1\}\end{gathered}$$

and hence $D_\sigma(\pi') \supseteq \{c_1, c_2, \ldots, c_l\} = T_i$. As $\phi(\pi) \in D(S_j)$ for some j with $\pi \leq_w \phi(\pi)$, this demonstrates that

$$T_i = \{c_1, c_2, \ldots, c_l\} \subseteq S_j.$$

□

5. Computer Based Approach

As has been shown thus far, injections preserving the weak order in S_{r+1} and B_{r+1} are interesting objects. In order to experiment with these injections, it was necessary to write computer programs to search for them. The following section presents some background information describing how the resultant programs work and what features are currently implemented. The full source code for these programs is contained in the appendices of this paper.

The programs, called *weak-Sn* and *weak-Bn,* were developed in the scripting language Python. The program *weak-Sn* was specifically designed to seek weak order respecting bijections in S_{r+1}, while the program *weak-Bn* was designed to handle the same task on B_{r+1}. Both programs are composed primarily of three distinct components:

Component 1: The Matching Algorithm. A core element of each program is the implementation of a matching algorithm for bipartite graphs. Given a bipartite graph G with vertex set $V = X \cup Y$, the matching algorithm employs a method designed by Hopcroft and Karp ("the most efficient sequential algorithm known" [BL]) to return an edge set of G corresponding to a maximum matching on G. The code for this component was freely available on the web, and can be attributed to the author David Eppstein [Ep].

Component 2: The Weak Ordering Component. The function of this component is to discover the actual injections sought after in this study. The way in which *weak-Sn* handles this task is as follows. This component of the program is given two basic inputs: a set of descent sets $T_1, \ldots, T_m$ and a set of descent sets $S_1, \ldots, S_n$. The program then creates a bipartite graph G whose vertex set is $V = D(T_1) \cup \ldots \cup D(T_m) \cup D(S_1) \cup \ldots \cup D(S_n)$, where there is an edge between $\pi_1 \in D(T_1) \cup \ldots \cup D(T_m)$ and $\pi_2 \in D(S_1) \cup \ldots \cup D(S_n)$ if and only if $\pi_1 \leq_w \pi_2$. After the graph G is generated, the matching algorithm is called to find a maximum matching on G. It is clear that if this matching has cardinality equal to $|D(T_1) \cup \ldots \cup D(T_m)|$, then this matching corresponds to an injection ϕ from $D(T_1) \cup \ldots \cup D(T_m)$ to $D(S_1) \cup \ldots \cup D(S_n)$ such that $\pi \leq_w \phi(\pi)$ for all $\pi \in D(T_1) \cup \ldots \cup D(T_m)$. It is also clear that if the cardinality of the matching is less than $|D(T_1) \cup \ldots \cup D(T_m)|$, then there is no such matching. Hence this component is used to determine whether or not a given set of subsets $S_1, \ldots, S_n$ dominates another set of subsets $T_1, \ldots, T_m$. For the program *weak-Bn* the method is analogous (simply replace each D above with D_σ). The way in which the weak order for B_{r+1} is implemented in *weak-Bn* is by using the "poset definition" presented earlier.

Component 3: The Interface. The last major component of each program is the interface by which all of the program's functions can be accessed. The interface of *weak-Sn* splits its capabilities into two classes: manual and automatic. Manual mode gives the user full control to examine whether or not one collection of sets dominates another. Automatic mode gives the user the ability to run some more complex, batch tasks. Currently there are two options: the user can see if all subsets of $[r]$ of cardinality i are dominated by all subsets of $[r]$ of cardinality $r - i$, or the user can run through all tuples of descent sets of different sizes to seek one tuple which dominates another. The interface of *weak-Bn* exhibits similar capabilities (as *weak-Bn* was built up from *weak-Sn*), however *weak-Bn* has the added capability of being able to check if all i-subsets of $\{0, 1, \ldots, r\}$ are dominated by all $(i+1)$-subsets of $\{0, 1, \ldots, r\}$ for $i < \lfloor \frac{r}{2} \rfloor$. Undoubtedly the interface of each program is likely to evolve as more is done to investigate the work at hand.

6. Results Generated by the Programs

The following section outlines the most noteworthy results that have been verified/discovered with the help of *weak-Sn* and *weak-Bn.*

First, at the end of Section 3, it was noted that some interesting results may be obtained by examining *triples* of descent sets which dominate triples of descent sets. Namely, this allows us to give a combinatorial flag h-vector proof of the fact that $h_2 \leq h_3$ in all geometric lattices L of rank greater than or equal to 6. Specifically, it has been verified by *weak-Sn* that:

$$\begin{aligned}
\{1,3\},\{1,5\},\{2,4\} &\text{ is dominated by } \{1,2,4\},\{1,3,5\},\{2,3,4\} \text{ in } S_6\\
\{2,3\},\{2,5\},\{3,5\} &\text{ is dominated by } \{1,2,5\},\{2,3,5\},\{2,4,5\} \text{ in } S_6\\
\{1,4\},\{3,4\} &\text{ is dominated by } \{1,3,4\},\{1,4,5\} \text{ in } S_6\\
\{1,2\} &\text{ is dominated by } \{1,2,3\} \text{ in } S_6\\
\{4,5\} &\text{ is dominated by } \{3,4,5\} \text{ in } S_6
\end{aligned}$$

which shows that the collection of all subsets of $[5]$ of cardinality 2 is dominated by the collection of all subsets of $[5]$ of cardinality 3, proving the result stated above.

In examining descent sets on B_{r+1}, many of the same difficulties arise as in the case of S_{r+1}. For instance, using *weak-Bn* it was verified that the descent set $\{1\}$ is not dominated by any 2-subset of $\{0,1,2\}$ in B_3. Thus a proof that the collection of all 1-subsets of $\{0,1,2\}$ is dominated by the collection of all 2-subsets of $\{0,1,2\}$ in B_3 can not be obtained simply by examining single descent sets. It was also verified using *weak-Bn* that this result can not be obtained by moving to doubles of descent sets either. Even so, some results may be obtained by examining doubles of descent sets in other groups. For instance, using *weak-Bn,* it has been shown that:

$$\begin{aligned}
\{0\},\{1\} &\text{ is dominated by } \{0,2,3\},\{1,2,3\} \text{ in } B_4\\
\{2\},\{3\} &\text{ is dominated by } \{0,1,2\},\{0,1,3\} \text{ in } B_4
\end{aligned}$$

which shows that the collection of all 1-subsets of $\{0,1,2,3\}$ is dominated by the collection of all 2-subsets of $\{0,1,2,3\}$ in B_4.

weak-Sn and *weak-Bn* were also used to examine the following three conjectures concerning descent set domination. The first is a conjecture attributed to Stanley:

Conjecture 6.1. *In S_{r+1}: The collection of all i-subsets of $[r]$ is dominated by the collection of all $(r-i)$-subsets of $[r]$, for $i \leq \lfloor \frac{r}{2} \rfloor$. In B_{r+1}: The collection of all i-subsets of $\{0,1,\ldots,r\}$ is dominated by the collection of all $(r+1-i)$-subsets of $\{0,1,\ldots,r\}$, for $i \leq \lfloor \frac{r+1}{2} \rfloor$.*

Using *weak-Sn,* the above conjecture was verified in S_{r+1} for all $r \leq 8$. Using *weak-Bn,* the above conjecture was verified in B_{r+1} for all $r \leq 5$.

Another conjecture is the following:

Conjecture 6.2. *In S_{r+1}: The collection of all i-subsets of $[r]$ is dominated by the collection of all $(i+1)$-subsets of $[r]$, for $i < \lfloor \frac{r}{2} \rfloor$. In B_{r+1}: The collection of all i-subsets of $\{0, 1, \ldots, r\}$ is dominated by the collection of all $(i+1)$-subsets of $\{0, 1, \ldots, r\}$, for $i < \lfloor \frac{r+1}{2} \rfloor$.*

The above conjecture went largely unstudied in S_{r+1}, but was proved by *weak-Bn* to be true in B_{r+1} for all $r \leq 5$.

Note that by Theorem 2.4, a proof of Conjectures 6.1 and 6.2 would yield a combinatorial proof of Theorem 1.2. Additionally, a proof of Conjecture 6.1 would yield a proof of Conjecture 6.2. To see why this is true for S_{r+1}, note that if the collection of all i-subsets of $[r]$ is dominated by the collection of all $(r-i)$-subsets of $[r]$ for some r and $i \leq \lfloor \frac{r}{2} \rfloor$, then it is not hard to see that the collection of all i-subsets of $[r+1]$ is dominated by the collection of all $(r-i)$-subsets of $[r+1]$. Taking i to be such that $i+1 = r-i$, this shows that a proof of Conjecture 6.1 would imply that the collection of all i-subsets of $[r]$ is dominated by the collection of all $(i+1)$-subsets of $[r]$ for all $r \geq 2i$. Similar reasoning applies for B_{r+1}.

The third conjecture was formulated in [NS]:

Conjecture 6.3. *Let $T \subseteq [r]$ and let $T \circ \beta \subseteq [r]$ be defined by*

$$T \circ \beta = \{i \in [r] : r - i + 1 \notin T\}.$$

Then if $T \subseteq T \circ \beta$, $T \circ \beta$ dominates T in S_{r+1}.

In [NS] it was mentioned that the above conjecture was verified by computer for $r \leq 8$. Using *weak-Sn,* this has now been verified for $r = 9$. Note that Theorems 3.4 and 3.6 above represent special cases of the preceding conjecture.

The above three conjectures represent likely directions in which one can search for the injections which will ultimately reveal flag h-vector inequalities. By following these conjectures and implimenting both the Subset Principles and the techniques employed in the proof of Theorems 3.2, 3.4 and 3.6, it is hoped that general principles regarding these interesting injections will unfold.

7. Applications to Finite Distributive Lattices

The preceding sections of this paper have dealt mostly with investigating properties of the weak ordering on S_{r+1} and B_{r+1}, motivated by what these connections imply about inequalities relating to the h-vector of the order complex of geometric lattices. In this section and in the one to follow it will be shown that these applications represent only the beginning, and that study of this order relation has implications for other classes of simplicial complexes.

The first simplicial complex that we examine is the order complex of what is known as a distributive lattice. A lattice L is said to be a *distributive lattice* if the following two distributive laws hold for all $x, y, z \in L$:

$$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$

Some interesting things can be said about the underlying structure of all finite distributive lattices, but some further terminology must be developed in order to understand these statements.

Let P be a poset. Define an *order ideal* of P to be a subset $I \subseteq P$ such that if $x \in I$, then $y \in I$ for all $y \leq_P x$. Note that a new poset can be formed from P by taking the set of all order ideals of P ordered by inclusion. This poset is denoted by $J(P)$.

Theorem 7.1. (Birkhoff) *Let L be a finite distributive lattice. Then there exists a unique (up to isomorphism) finite poset P such that $L \cong J(P)$ (where here $\cong$ is used to denote equivalence by lattice isomorphism).*

For a proof of the above theorem, known as the fundamental theorem of finite distributive lattices, see [St].

For any finite distributive lattice L, one defines the order complex of L (denoted by $\Delta(L)$) just as was done for geometric lattices. Earlier it was noted that the structure of geometric lattices is such that the flag h-vector of their order complexes has a nice combinatorial interpretation. As we shall soon see, the structure of finite distributive lattices (as revealed by the above theorem) is such that the flag h-vectors of their order complexes may also be given a nice combinatorial interpretation.

So let L be a finite distributive lattice, and let P be the corresponding finite poset such that $L \cong J(P)$. Label the elements of P by $[r+1]$ in such a way that P is a *natural ordering,* that is if $i \leq_P j$, then $i \leq j$ in the usual ordering on $[r+1]$. Then for any edge (I, I') in the Hasse diagram of $J(P)$ (hence L), define a label on the edge (I, I') by $\lambda(I, I') = j$, where $j \in [r+1]$ is the unique element of the set $I' - I$. Note that j is well defined as the fact that I' covers I implies that $I \subseteq I'$ and that $|I' - I| = 1$. This edge labeling introduces a lexicographic ordering on the facets of $\Delta(L)$, where each such F is given a label $\lambda(F)$ as an ordered tuple of the edges of which it is comprised (from $\hat{0}$ to $\hat{1}$). As shown in [St], this is actually an EL-labeling. Hence we have the same combinatorial interpretation of the flag h-vector of $\Delta(L)$ as afforded in Proposition 2.3.

To see what the weak order on S_{r+1} can say about the flag h-vector of the order complex of a distributive lattice, first note the following lemma (an analogue of the Switching Lemma presented earlier).

Lemma 7.2. (Switching Lemma for Distributive Lattices) *Let L be a finite distributive lattice of rank $r+1$ and let $(b_1, \ldots, b_i, b_{i+1}, \ldots, b_{r+1})$ be the labeling of a facet of $\Delta(L)$. If $b_i > b_{i+1}$ in the natural ordering on $[r+1]$, then $(b_1, \ldots, b_{i+1}, b_i, \ldots, b_{r+1})$ is also the labeling of a facet of $\Delta(L)$.*

We briefly note that this lemma is essentially the "reverse" of the Switching Lemma presented earlier. The implications of this fact will be felt in Theorem 7.3 below.

Proof. By definition, $(b_1, \ldots, b_i, b_{i+1}, \ldots, b_{r+1})$ is the labeling of the maximal chain

$$\hat{0} < I_1 < I_2 < \ldots < I_r < \hat{1} \text{ in } J(P) \cong L$$

where $I_i = \{b_1, b_2, \ldots, b_i\}$ for $i \in [r]$. Now it is clear by the definition of the labeling that $(b_1, \ldots, b_{i+1}, b_i, \ldots, b_{r+1})$ is the labeling of a facet of $\Delta(L)$ if and only if the following is a facet of $\Delta(L)$:

$$\hat{0} < I_1 < I_2 < \ldots < I_{i-1} < I' < I_{i+1} < \ldots < I_r < \hat{1}$$

where $I' = \{b_1, b_2, \ldots, b_{i-1}, b_{i+1}\}$. This is because the labeling of a maximal chain of $J(P)$ uniquely determines the chain.

So in order to prove the theorem, it must be shown that the above "chain" truly is a chain of $J(P)$, and it is clear that this is true only if I' is an element of $J(P)$. We prove this fact by contradiction.

Assume that $I' \subseteq [r+1]$ is *not* an order ideal of P, i.e. there exists some $j \in I'$ and some $j' \in P$ such that $j' \leq_P j$ but $j' \notin I'$. As

$$I_{i-1} = \{b_1, b_2, \ldots, b_{i-1}\} = I' - \{b_{i+1}\}$$

is an order ideal of P, $j = b_{i+1}$. Also, as

$$I_{i+1} = \{b_1, b_2, \ldots, b_{i+1}\}$$

is an order ideal of P, $j' \leq_P j$ implies that $j' \in I_{i+1}$. So $j' \in I_{i+1} - I'$, i.e. $j' = b_i$. But as $b_i > b_{i+1}$ in the natural ordering on $[r+1]$, it can not be true that $b_i = j' \leq_P j = b_{i+1}$ (as the partial ordering on P respects the natural ordering on $[r+1]$). This is a contradiction.

Thus I' is an order ideal of P, completing the proof. □

Using this lemma, the following theorem can be proved.

Theorem 7.3. *Let L be a distributive lattice of rank $r+1$ and let $\Delta(L)$ be the corresponding order complex. Let $S, T \subseteq [r]$ and assume that S dominates T. Then if $|T| = j$ and $|S| = r - j$ for some $j \leq \lfloor \frac{r}{2} \rfloor$,*

$$h_T(L) \geq h_S(L).$$

And if $|T| = j$ and $|S| = j+1$ for some $j < \lfloor \frac{r}{2} \rfloor$,

$$h_{S^C}(L) \geq h_{T^C}(L)$$

where $S^C = [r] - S$ and $T^C = [r] - T$.

Proof. To prove the first claim, assume that $|T| = j$ and $|S| = r - j$ for $j \leq \lfloor \frac{r}{2} \rfloor$.

By assumption, there exists an injection $\phi : D(T) \to D(S)$ which respects the weak order on S_{r+1}. But as $S, T \subseteq [r]$ and $|S| = r - |T|$, $|D(T)| = |D(S)|$. Hence ϕ is in fact a bijection. Then the inverse function ϕ^{-1} is an injection from $D(S)$ to $D(T)$ which *reverses* the weak order on S_{r+1}. By the switching lemma for distributive lattices and the definition of the weak order on S_{r+1}, ϕ^{-1} induces an injection from those facets of $\Delta(L)$ whose labels have descent set S to those whose labels have descent set T. Hence by the combinatorial interpretation of the flag h-vector of $\Delta(L)$, this yields $h_S(L) \leq h_T(L)$.

To prove the second part of the theorem, assume that $|T| = j$ and $|S| = j+1$ for $j < \lfloor \frac{r}{2} \rfloor$. As before, let $\phi : D(T) \to D(S)$ be the implied injection. We wish to define a function ϕ_2 on $D(T^C)$, but to do so it is necessary to develop some preliminary notation.

Let $\pi = a_1 a_2 \dots a_{r+1} \in S_{r+1}$, and define the permutation $\pi^R \in S_{r+1}$ by $\pi^R = a_{r+1} a_r \dots a_1$. Note that if $\pi \in D(T)$ then $\pi^R \in D(T^C)$ and vice versa (similarly if we replace T with S or any other descent set). Then define the function $\phi_2 : D(T^C) \to D(S^C)$ by

$$\phi_2(\pi) = (\phi(\pi^R))^R \text{ for any } \pi \in T^C.$$

The fact that $\pi_1^R = \pi_2^R$ if and only if $\pi_1 = \pi_2$ coupled with the fact that ϕ is an injection implies that ϕ_2 is likewise an injection. As one can also check, $\pi_1 \leq_w \pi_2$ for $\pi_1, \pi_2 \in S_{r+1}$ implies that $\pi_1^R \geq_w \pi_2^R$. Thus for any $\pi \in T^C$:

$$\begin{aligned} \pi^R &\leq_w \phi(\pi^R) \text{ (by definition of } \phi) \\ \Rightarrow \pi = (\pi^R)^R &\geq_w (\phi(\pi^R))^R = \phi_2(\pi) \end{aligned}$$

So ϕ_2 is an injection from $D(T^C)$ to $D(S^C)$ which reverses the weak order on S_{r+1}. As before, this yields $h_{T^C}(L) \leq h_{S^C}(L)$. □

It was shown in [BF] that for L a distributive lattice of rank $r+1$, the following inequalities hold

$$\begin{aligned} h_j(\Delta(L)) \geq h_{r-j}(\Delta(L)) \text{ for } j \leq \lfloor \frac{r}{2} \rfloor \\ \text{and } h_{r-j-1}(\Delta(L)) \geq h_{r-j}(\Delta(L)) \text{ for } j < \lfloor \frac{r}{2} \rfloor, \end{aligned}$$

but just as with Theorem 1.2 the proof is not combinatorial in nature. A proof of Conjectures 6.1 and 6.2 in S_{r+1}, along with the result the preceding theorem, would provide the necessary material for a combinatorial proof of these inequalities.

As a final note, recall that the fundamental theorem of finite distributive lattices indicates a structural link between each finite poset and an associated distributive lattice. Thus the above inequalities (and hence the weak order on S_{r+1}) are actually linked to the structure of arbitrary finite posets.

8. Applications to Supersolvable Lattices

As a further application of the results of this paper, a link is noted between the weak order in S_{r+1} and what are known as supersolvable lattices. A finite lattice L is said to be *supersolvable* if there exists a maximal chain C on L such that the sublattice generated by C and any other chain of L is a distributive lattice. In this section we will be interested only in those supersolvable lattices L whose Möbius function μ satisfies $\mu(x, y) \neq 0$ for all $x <_L y$ in L.

In a paper yet to be published, Jay Schweig proved the following proposition about such supersolvable lattices.

Proposition 8.1. (Jay Schweig) *Let L be a finite supersolvable lattice of rank $r+1$ whose Möbius function satisfies the above restriction. Then there exists an EL-labeling on the edges of L and a partition of the maximum chains of L such that the following property holds: if C is a maximal chain of L labeled by*

$$(b_1, b_2, \dots, b_i, b_{i+1}, \dots, b_{r+1}) \text{ with } b_i < b_{i+1}$$

then there exists a maximal chain C' of L in the same block as C whose labeling is obtained by transposing b_i and b_{i+1} in the labeling of C. That is, the labeling of C' is given by

$$(b_1, b_2, \ldots, b_{i+1}, b_i, \ldots, b_{r+1}).$$

From the above proposition, the following theorem follows.

Theorem 8.2. *Let L be a finite supersolvable lattice of rank $r+1$ whose Möbius function satisfies the above restriction, and let $\Delta(L)$ denote the order complex of L. Let $S, T \subseteq [r]$ and assume that S dominates T. Then $h_T(L) \leq h_S(L)$.*

Proof. By assumption, there exists an injection $\phi : D(T) \to D(S)$ respecting the weak order on S_{r+1}. But by Proposition 8.1 and the definition of the weak order, it follows that for any chain $C \in \Delta(L)$ whose labeling $\lambda(C)$ has descent set T, there exists a chain $C' \in \Delta(L)$ whose labeling is $\phi(\lambda(C))$ (where here we make use of the natural correspondence between maximum chain labelings and permutations of $[r+1]$). Hence there exists a chain C' whose labeling has descent set S. Thus by the combinatorial interpretation of the flag h-vector of the order complex of an EL-labeled lattice, this shows that $h_T(L) \leq h_S(L)$. □

As has also been shown by Jay Schweig, the following inequalities hold for L such a supersolvable lattice (with rank $r+1$):

$$h_j(\Delta(L)) \leq h_{r-j}(\Delta(L)) \text{ for } j \leq \lfloor \frac{r}{2} \rfloor$$
$$\text{and } h_j(\Delta(L)) \leq h_{j+1}(\Delta(L)) \text{ for } j < \lfloor \frac{r}{2} \rfloor,$$

but again the proof of this fact is not combinatorial in nature. A proof of Conjectures 6.1 and 6.2 in S_{r+1}, along with the result the Theorem 8.2, would provide the basis for a combinatorial proof of these inequalities.

9. Future Research Directions

There are many directions in which research on this problem can progress. We end this paper with a list of the more tenable choices:

(1) A useful step in the continuation of the research begun in [NS] and continued here would be to enhance the code of *weak-Sn* and *weak-Bn,* including implementing new automated tasks in each program's interface, optimizing the code for speed, making the code more readable, etc. As more properties of weak order preserving injections are discovered, they can be implemented into the computer programs in order to boost efficiency.

(2) Another step would be to adapt *weak-Sn* and *weak-Bn* to handle all finite Coxeter groups. In particular, it would probably be a rather simple task to adapt these programs to study the group D_{r+1} of even-signed permutations (the subgroup of B_{r+1} consisting of signed permutations having an even number of negative signs). The weak order on D_{r+1} is

harder to define, but could be examined by computer just as with S_{r+1} and B_{r+1}.

(3) One could also examine the weak order on finite products of A_{r+1} and B_{r+1}, defined in the obvious way, to see what can be concluded from what is already known of A_{r+1} and B_{r+1}. What can be said about subset domination on products of A_{r+1} and B_{r+1} given what is known about the sets individually?

(4) A significant amount has been said about the implications of the weak order on the h-vector of various simplicial complexes, but there are still many implications which were not studied in this paper. For instance, one could investigate the link between the permutation group S_{r+1} and the f- and h-vectors of the face lattice of zonotopes (as discussed in [BER]).

Appendix A. Code for *weak-Sn*

What follows is the Python code for the program *weak-Sn.* Because the layout of tabs and carriage returns is so important to the structure of a Python program, we make the following two notes concerning the manner in which this code was typset. First, all tabs have been represented by a series of three spaces. Additionally, the symbol ">>>" has been used to indicate that what follows is a continuation of the previous line of text.

```
## WEAK-SN ##

import sys

def automated1(pBar):
# Function designed to handle the automated task of finding a
# matching from all those permutations having descent set of size
# j to all those permutations having descent set of size
# n - 1 - j.
   n = input('Enter the number of elements which we will be permu
>>>ting: ')
   print "We attempt to find a matching from descent sets having"
   print "size j into descent sets having size %i - j" % (n-1)
   j = input('Enter your choice for j: ')
   L1 = generateList("choose1",n-1,j,[])
   L2 = generateList("choose1",n-1,n-1-j,[])
   seekMatching(L1,L2,n,False,pBar)


def automated2(pBar):
# Function designed to handle the automated task of finding a
# matching from all those permutations in some given number of
# descent sets of a certain size to all those pemutations in the
```

```
# same number of descent sets of a different size.
   n = input('Enter the number of elements which we will be permu
>>>ting: ')
   g = input('Enter how many sets you would like to be in each de
>>>scent list collection: ')
   dSize1 = input('Enter the size of the descent sets which you w
>>>ould like to match: ')
   L1 = generateList("choose1",n-1,dSize1,[])
   M1 = generateList("choose0",len(L1),g,[])
   numForce = input('How many descent sets would you like to forc
>>>e to be checked every time? ')
   for i in range(numForce):
      forceSet = input("Please enter such a descent set (#%i): "
>>>% (i+1))
      dex = descentIndex(L1,forceSet)
      M1 = forceDescSet(M1,dex)
   dSize2 = input('Enter the size of the descent sets which you a
>>>re matching into: ')
   L2 = generateList("choose1",n-1,dSize2,[])
   M2 = generateList("choose0",len(L2),g,[])
   numForce = input('How many descent sets would you like to forc
>>>e to be checked every time? ')
   for i in range(numForce):
      forceSet = input("Please enter such a descent set (#%i): "
>>>% (i+1))
      dex = descentIndex(L2,forceSet)
      M2 = forceDescSet(M2,dex)
   dSetsList1,dSetsList2 = [],[]
   for i in M1:
      for ind1 in i:
         dSetsList1 = dSetsList1 + [L1[ind1]]
      for j in M2:
         for ind2 in j:
            dSetsList2 = dSetsList2 + [L2[ind2]]
         seekMatching(dSetsList1,dSetsList2,n,True,pBar)
         dSetsList2=[]
      dSetsList1=[]

def bipartiteMatch(graph):
# Hopcroft-Karp bipartite max-cardinality matching and max
# independent set
# David Eppstein, UC Irvine, 27 Apr 2002
   '''Find maximum cardinality matching of a bipartite graph
   (U,V,E). The input format is a dictionary mapping members of U
   to a list of their neighbors in V.  The output is a triple
```

```
(M,A,B) where M is a dictionary mapping members of V to their
matches in U, A is the part of the maximum independent set in
U, and B is the part of the MIS in V. The same object may
occur in both U and V, and is treated as two distinct vertices
if this happens.'''

# initialize greedy matching (redundant, but faster than full
# search)
matching = {}
for u in graph:
   for v in graph[u]:
      if v not in matching:
         matching[v] = u
         break

while 1:
   # structure residual graph into layers
   # pred[u] gives the neighbor in the previous layer for u in
   # U
   # preds[v] gives a list of neighbors in the previous layer
   # for v in V
   # unmatched gives a list of unmatched vertices in final
   # layer of V, and is also used as a flag value for pred[u]
   # when u is in the first layer
   preds = {}
   unmatched = []
   pred = dict([(u,unmatched) for u in graph])
   for v in matching:
      del pred[matching[v]]
   layer = list(pred)

   # repeatedly extend layering structure by another pair of
   # layers
   while layer and not unmatched:
      newLayer = {}
      for u in layer:
         for v in graph[u]:
            if v not in preds:
               newLayer.setdefault(v,[]).append(u)
      layer = []
      for v in newLayer:
         preds[v] = newLayer[v]
         if v in matching:
            layer.append(matching[v])
            pred[matching[v]] = v
```

```
            else:
               unmatched.append(v)

      # did we finish layering without finding any alternating
      # paths?
      if not unmatched:
         unlayered = {}
         for u in graph:
            for v in graph[u]:
               if v not in preds:
                  unlayered[v] = None
         return (matching,list(pred),list(unlayered))

      # recursively search backward through layers to find
      # alternating paths
      # recursion returns true if found path, false otherwise
      def recurse(v):
         if v in preds:
            L = preds[v]
            del preds[v]
            for u in L:
               if u in pred:
                  pu = pred[u]
                  del pred[u]
                  if pu is unmatched or recurse(pu):
                     matching[v] = u
                     return 1
         return 0

      for v in unmatched: recurse(v)

def descentIndex(descList,descSet):
# For a list of descent sets descList, this function returns the
# index at which the descent set descSet is located.
   n,index = len(descList),-1
   for i in range(n):
      if sContains(descList[i],descSet) and sContains(descSet,des
>>>cList[i]):
         index = i
   return index

def descentPerms(prefix, descSet, permSize):
# Recursive function meant to generate all the permutations
# having the given descent set (descSet), for permutations on
# permSize elements.
```

```
   L=[]
   n = len(prefix)
   if n == permSize:
      L = L + [prefix]
   elif n in descSet:
      if prefix[n-1] != 1:
         for i in range(1,prefix[n-1]):
            if (i in prefix) == False:
               L = L + descentPerms(prefix + [i],descSet,permSize
>>>)
   else:
      if prefix[n-1] != permSize:
         for i in range(prefix[n-1]+1,permSize+1):
            if (i in prefix) == False:
               L = L + descentPerms(prefix + [i],descSet,permSize
>>>)
   return L

def forceDescSet(indexList,descIndex):
# This function takes a list of sets of indices (indexList), and
# returns a modified list in which all sets of indices not
# including the index descIndex have been removed.
   L = []
   for i in indexList:
      if sContains([descIndex],i):
         L = L + [i]
   return L

def genChoose(prefix, n, i):
# Recusive function meant to generate all the subsets of [n]
# having cardinality i, where [n] = {1,2,...,n} or [n] =
# {0,1,...,n-1}.
   L = []
   size =len(prefix)
   if size == i:
      L = [prefix]
   else:
      for j in range(prefix[size-1]+1,n+1):
         L = L + genChoose(prefix+[j],n,i)
   return L

def generateList(listType,size,k,descSet):
# This function is meant to interface with the functions
# genChoose and descentPerms.  We can think of this function as
# taking care of the base case for each of these recursive
```

```
# functions.
   L = []
   if listType == "permutations":
      for i in range(size):
         L = L + descentPerms([i+1],descSet,size)
   elif listType == "choose0":
      for i in range(size):
         L = L + genChoose([i],size-1,k)
   elif listType == "choose1":
      for i in range(1,size+1):
         L = L + genChoose([i],size,k)
   return L

def genGraph(invList1,invList2,pBarVal):
# Generates the actual graph between lists of permutations, given
# the inversion sets for each permutation.
   edgeSet={}
   if pBarVal:
      counter = 0
      print "  Generating Graph (progress indicated below)"
      print "  0%--------25%--------50%--------75%------100%"
      print "  ",
   for i in range(len(invList1)):
      for j in range(len(invList2)):
         if sContains(invList1[i],invList2[j]):
            if edgeSet.has_key(j):
               edgeSet[j].append(i)
            else:
               edgeSet[j]=[i]
      if pBarVal:
         count2 = (45*(i+1))/(len(invList1))
         if count2 != counter:
            for k in range(count2-counter):
               sys.stdout.write(".")
            counter = count2
   if pBarVal:
      print ""
   return edgeSet

def invSet(permutation):
# Calculates the inversion set of the given permutation.
   L=[]
   for i in range(len(permutation)):
      for j in range(i+1,len(permutation)):
         if permutation[i] > permutation[j]: L.append((permutatio
```

```
>>>n[i],permutation[j]))
   return L

def sContains(list1,list2):
# Boolean function to test whether the list list1 is contained in
# the list list2.
   contain = True
   for i in list1:
      if (i in list2) == False:
         contain = False
         break
   return contain

def seekMatching(dSetsList1,dSetsList2,permSize,printTitle,pBar):
# Function that coordinates the entire task of finding a matching
# given the descent sets of the permutations under investigation
# and the size of the permutation group to be studied.
   if printTitle:
      print "Matching",
      for i in dSetsList1:
         print i,
      print "-->",
      for i in dSetsList2:
         print i,
      print ""
   leftList,rightList=[],[]
   invLeft,invRight=[],[]
   for i in dSetsList1:
      leftList = leftList + generateList("permutations",permSize,
>>>0,i)
   for i in dSetsList2:
      rightList = rightList + generateList("permutations",permSiz
>>>e,0,i)
   print "(a) The lists have been computed."
   if pBar[0]:
      counter = 0
      print "  Generating Inversions (progress indicated below)"
      print "  0%--------25%--------50%--------75%------100%"
      print "  ",
   for i in range(len(leftList)):
      invLeft = invLeft + [invSet(leftList[i])]
      if pBar[0]:
         count2 = (22*(i+1))/(len(leftList))
         if count2 != counter:
            for k in range(count2-counter):
```

```
               sys.stdout.write(".")
            counter = count2
   if pBar[0]:
      counter = 0
   for i in range(len(rightList)):
      invRight = invRight + [invSet(rightList[i])]
      if pBar[0]:
         count2 = (23*(i+1))/(len(rightList))
         if count2 != counter:
            for k in range(count2-counter):
               sys.stdout.write(".")
            counter = count2
   if pBar[0]:
      print ""
   print "(b) The inversion sets have been computed."
   D = genGraph(invLeft,invRight,pBar[1])
   print "(c) The graph has been generated."
   match = bipartiteMatch(D)[0]
   print "(d) A maximal matching has been computed."
   if len(match.keys()) == len(leftList):
      print "*** A complete matching was found (on %i vertices)!
>>>***" % len(leftList)
      outChoice = raw_input('Would you to write this matching to
>>>a file (enter "yes" if so)? ')
      if outChoice in ('YES','yes','Y','y'):
         filename = raw_input('Please enter a file name: ')
         writeMatch(filename,match,leftList,rightList)
   else:
      print "  A complete matching was not found."
      print "  Only %i out of %i of the 'left' permutations were
>>>matched." % (len(match.keys()),len(leftList))

def setBools():
# Function that sets certain boolean values based on settings in
# a configuration file ('weakorder.ini').  The settings in this
# file determine whether to display a progress bar while the
# inversion sets are being generated (if invBar is True) and
# whether to display a progress bar while the graph is being
# generated (if grapBar is true).
   invBar,grapBar = False,False
   iniVals = open('weakorder.ini', 'r')
   S = iniVals.readline()
   while S[0] == ':':
      S = iniVals.readline()
   if S[:8] == "invBar=1":
```

```
      invBar = True
   S = iniVals.readline()
   while S[0] == ':':
      S = iniVals.readline()
   if S[:9] == "grapBar=1":
      grapBar = True
   return invBar,grapBar

def writeMatch(filename, matching,list1,list2):
# Function designed to write the matching (injection) whenever a
# full matching exists.
   matchOutput = open(filename, 'w')
   for i in matching.keys():
      matchOutput.write("%s - %s\n" % (list1[i],list2[matching[i]
>>>]))
   return None

# What follows is the main body of the program, whose primary
# purpose is to present an interface to the user.
pBar = setBools()
print "Please select from one of the following menu options:"
print "/------------------------------."
print "| (a) Automated Matching Tasks |"
print "| (m) Manual Matching Tasks    |"
print ".------------------------------/"
menuChoice = raw_input("Enter your choice: ")
if menuChoice in ('a','A'):
   print "Please choose among the following tasks:"
   print "/-----------------------------------------------------
>>>------."
   print "| (1) Matching descent sets of size j into those of siz
>>>e d-j |"
   print "| (2) Matching collections of descent sets
>>>      |"
   print ".-----------------------------------------------------
>>>------/"
   menu2Choice = input("Enter your choice: ")
   if menu2Choice == 1:
      automated1(pBar)
   elif menu2Choice == 2:
      automated2(pBar)
   else:
      print "Invalid Input"
elif menuChoice in ('m','M'):
   L1,L2 = [],[]
```

```
   n = input('Enter the number of elements which we will be permu
>>>ting: ')
   m = input('Enter the number of descent sets on each side of th
>>>e graph: ')
   print 'For the permutations to be matched:'
   for i in range(m):
      dSet = input('Enter descent set #%i (as a list, i.e. [2,3])
>>>: ' % (i+1))
      L1 = L1 + [dSet]
   print 'For the permutations we will match into:'
   for i in range(m):
      dSet = input('Enter descent set #%i (as a list, i.e. [2,3])
>>>: ' % (i+1))
      L2 = L2 + [dSet]
   seekMatching(L1,L2,n,True,pBar)
else:
   print "Invalid Input"
```

Included below is a sample configuration file, called *weakorder.ini,* which is read by the program *weak-Sn.*

```
: Configuration file for WEAK-SN the program
:
: The following flag enables (1) or disables (0) the inversion
: progress bar
: (note: the inversion progress bar is not entirely accurate)
invBar=1
: The following flag enables (1) or disables (0) the graph
: progress bar
grapBar=1
```

Appendix B. Code for *weak-Bn*

This section contains the code for *weak-Bn.* The formatting conventions used in this section are just as they were in the last. Note that in *weak-Bn,* the possible descent positions for the group B_{r+1} are labeled by the numbers 1 through $r+1$. This is slightly different from the convention adopted in the body of this paper, where the descent positions for the group B_{r+1} are labeled by the numbers 0 through r.

```
## WEAK-BN ##

def bipartiteMatch(graph):
# This function is exactly as in weak-Sn, and is hence omitted.

def genChoose(prefix, n, i):
# This function is exactly as in weak-Sn, and is hence omitted.
```

```
def generateList(listType,size,k,descSet):
# This function is meant to interface with the functions
# genChoose and descentPerms.  It is analagous in function
# (although slightly different in structure) to generateList from
# weak-Sn.
   L = []
   if listType == "signed_permutations":
      L = SPdescentPerms([0],descSet,size)
      for i in L:
         del i[0]
   elif listType == "choose1":
      for i in range(1,size+1):
         L = L + genChoose([i],size,k)
   return L

def sContains(list1,list2):
# This function is exactly as in weak-Sn, and is hence omitted.

def SPBruhatLEQ(leftPerm,rightPerm,permSize):
# Function which checks whether leftPerm is less than or equal to
# rightPerm in the weak ordering on the signed permutation group.
   isLEQ = 1
   for i in range(1,permSize):
      for j in range(i+1,permSize+1):
         tempLEQ = SPLattComp(SPLatticeVal(SPFindPair(leftPerm,(i
>>>,j))),SPLatticeVal(SPFindPair(rightPerm,(i,j))))
         if tempLEQ != 1:
            isLEQ = 0
            break
      if isLEQ == 0: break
   return isLEQ

def SPdescentPerms(prefix, descSet, permSize):
# Recursive function meant to generate all the permutations
# having the given descent set (descSet), for permutations on
# permSize elements.  Specifically designed for signed
# permutations
   L=[]
   n = len(prefix)
   if n == permSize + 1:
      L = L + [prefix]
   elif n in descSet:
      if prefix[n-1] != -1*permSize:
         for i in range(-1*permSize,prefix[n-1]):
            if ((i in prefix) == False) and ((-1*i in prefix) ==
```

```
>>>False):
               L = L + SPdescentPerms(prefix + [i],descSet,permSi
>>>ze)
   else:
      if prefix[n-1] != permSize:
         for i in range(prefix[n-1]+1,permSize+1):
            if ((i in prefix) == False) and ((-1*i in prefix) ==
>>>False):
               L = L + SPdescentPerms(prefix + [i],descSet,permSi
>>>ze)
   return L

def SPFindPair(signPerm,pair):
# Given a signed permutation and a pair of positive integers,
# this function returns an ordered pair corresponding to the way
# in which this original pair of numbers appears in the signed
# permutation.
   if pair[0] in signPerm:
      iIndex = signPerm.index(pair[0])
   else:
      iIndex = signPerm.index(-1*pair[0])
   if pair[1] in signPerm:
      jIndex = signPerm.index(pair[1])
   else:
      jIndex = signPerm.index(-1*pair[1])
   if iIndex < jIndex:
      newPair = (signPerm[iIndex],signPerm[jIndex])
   else:
      newPair = (signPerm[jIndex],signPerm[iIndex])
   return newPair

def SPgenGraph(permList1,permList2,permSize):
# Generates the graph from which descent set matchings will be
# found.
   edgeSet={}
   for i in range(len(permList1)):
      for j in range(len(permList2)):
         if SPBruhatLEQ(permList1[i],permList2[j],permSize):
            if edgeSet.has_key(j):
               edgeSet[j].append(i)
            else:
               edgeSet[j]=[i]
   return edgeSet

def SPLattComp(leftVal,rightVal):
```

```
# Given two values of the signed permutation poset, this function
# returns a 1 if leftVal < rightVal in this poset and returns a 0
# otherwise.
   isLEQ = 0
   if leftVal == 0: geq = [0,1,2,3,4,5,6,7]
   elif leftVal == 1: geq = [1,3,4,6]
   elif leftVal == 2: geq = [2,5,6,7]
   elif leftVal == 3: geq = [3,4,6]
   elif leftVal == 4: geq = [4,6]
   elif leftVal == 5: geq = [5,6,7]
   elif leftVal == 6: geq = [6]
   else: geq = [6,7]
   if rightVal in geq:
      isLEQ = 1
   return isLEQ

def SPLatticeVal(pair):
# Given an ordered pair, this function returns an integer
# designating where that pair falls in the signed permutation
# poset.
   val = 0
   if abs(pair[0]) > abs(pair[1]): val = val + 1
   if abs(pair[0]) != pair[0]: val = val + 2
   if abs(pair[1]) != pair[1]: val = val + 4
   return val

def SPSeekMatching(dSetsList1,dSetsList2,permSize):
# Analogue of the function seekMatching in weak-Sn, but designed
# for signed permutations.
   if not(subsetPrin(dSetsList1,dSetsList2)):
      print "Matching fails to exist by the Subset Principle."
   else:
      leftList,rightList = [],[]
      for i in dSetsList1:
         leftList = leftList + generateList("signed_permutations"
>>>,permSize,0,i)
      for i in dSetsList2:
         rightList = rightList + generateList("signed_permutation
>>>s",permSize,0,i)
      print "(a) the lists have been computed."
      D = SPgenGraph(leftList,rightList,permSize)
      print "(b) the graph has been generated."
      match = bipartiteMatch(D)[0]
      print "(c) a maximal matching has been computed."
      if len(match.keys()) == len(leftList):
```

```
         print "*** A complete matching was found (on %i vertices
>>>)! ***" % len(leftList)
         outChoice = raw_input('Would you to write this matching
>>>to a file (enter "yes" if so)? ')
         if outChoice in ('YES','yes','Y','y'):
            filename = raw_input('Please enter a file name: ')
            writeMatch(filename,match,leftList,rightList)
      else:
         print "  A complete matching was not found."
         print "  Only %i out of %i of the 'left' permutations we
>>>re matched." % (len(match.keys()),len(leftList))

def subsetPrin(descList1,descList2):
# Function which determines whether or not the subset principle
# holds for descList1 (the left list) and descList2 (the right
# list).
   for i in descList1:
      principle = False
      for j in descList2:
         if sContains(i,j):
            principle = True
            break
      if not(principle):
         break
   return principle


def writeMatch(filename,matching,list1,list2):
# This function is exactly as in weak-Sn, and is hence omitted.

# What follows is the main body of the program, the primary
# purpose of which is to present an interface to the user.
print "Please select the task in which you wish to engage:"
print " (1) Match from j to n - j"
print " (2) Match from j into j + 1"
print " (3) Automated descent set matchings"
print " (4) Match descent sets manually"
print " (5) Check number of elements in descent set"
menu = input('>>> ')
if menu == 1:
   n = input('Enter the number of elements which we will be permu
>>>ting: ')
   print "We attempt to find a matching from descent sets having"
   print "size j into descent sets having size %i - j" % n
   j = input('Enter your choice for j: ')
```

```
   L1 = generateList("choose1",n,j,[])
   if L1 == []:
      L1 = [[]]
   L2 = generateList("choose1",n,n-j,[])
   #print L1
   #print L2
   SPSeekMatching(L1,L2,n)
elif menu == 2:
   n = input('Enter the number of elements which we will be permu
>>>ting: ')
   print "We attempt to find a matching from descent sets having"
   print "size j into descent sets having size j + 1"
   j = input('Enter your choice for j: ')
   L1 = generateList("choose1",n,j,[])
   if L1 == []:
      L1 = [[]]
   L2 = generateList("choose1",n,j+1,[])
   SPSeekMatching(L1,L2,n)
elif menu == 3:
   dSet1, dSet2 = [], []
   n = input('Enter the number of elements which we will be permu
>>>ting: ')
   m1 = input('Enter the size of the descent sets you would like
>>>to match: ')
   m2 = input('Enter the size of the descent sets you would like
>>>to match into: ')
   l = input('Enter the number of descent sets on each side: ')
   L1 = generateList("choose1",n,m1,[])
   if L1 == []:
      L1 = [[]]
   L2 = generateList("choose1",n,m2,[])
   L1CHOOSE = generateList("choose1",len(L1),l,[])
   L2CHOOSE = generateList("choose1",len(L1),l,[])
   for i in L1CHOOSE:
      for k in i:
         dSet1 = dSet1 + [L1[k-1]]
      for j in L2CHOOSE:
         for k in j:
            dSet2 = dSet2 + [L2[k-1]]
         print "Matching",
         for k in dSet1:
            print k,
         print "-->",
         for k in dSet2:
            print k,
```

```
         print ""
         SPSeekMatching(dSet1,dSet2,n)
         dSet2 = []
      dSet1 = []
elif menu == 4:
   dSet1, dSet2 = [], []
   n = input('Enter the number of elements which we will be permu
>>>ting: ')
   m1 = input('Enter the number of descent sets you would like to
>>> match: ')
   m2 = input('Enter the number of descent sets you would like to
>>> match into: ')
   for i in range(m1):
      dSet1 = dSet1 + [input('Enter the descent set (#%i) you wou
>>>ld like to match: ' % (i+1))]
   for i in range(m2):
      dSet2 = dSet2 + [input('Enter the descent set (#%i) you wou
>>>ld like to match into: ' % (i+1))]
   SPSeekMatching(dSet1,dSet2,n)
elif menu == 5:
   list = []
   n = input('Enter the number of elements which we will be permu
>>>ting: ')
   m = input('Enter the number of descent sets you would like to
>>>count: ')
   for i in range(m):
      dSet = input('Enter descent set #%i (i.e. [2,3], etc.): ' %
>>> (i+1))
      list = list + generateList("signed_permutations",n,0,dSet)
   print "There are %i signed permutations with the given descent
>>> set" % len(list)
else:
   print "menu choice invalid"
```

References

[BL] H.A. Baier Saip and C.L. Lucchesi. Matching algorithms for bipartite graphs. *Technical Report DCC-03/93.* Departmento de Cincia da Computao, Universidade Estudal de Campinas, 1993.

[BER] L.J. Billera, R. Ehrenborg and Margaret Readdy. The c-2d-Index of Oriented Matroids. *Journal of Combinatorial Theory,* 80:1 79-105, 1997.

[BF] A. Björner and J.D. Farley. Chain polynomials of distributive lattices are 75 % unimodal. 2004, arXiv:math.CO/0411610.

[Ep] D. Eppstein. Hopcroft-Karp bipartite matching. *http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/123641.*

[Ni] I. Niven. A combinatorial problem on finite sequences. *Nieuw Arch. Wisk.,* 16: 116-123, 1968.

[NS] K. Nyman and E. Swartz. Inequalities for the h-vectors and flag h-vectors of geometric lattices. *Discrete and Comput. Geom.,* 32: 533-548, 2004.

[St] R.P. Stanley. *Enumerative Combinatorics, Volume I.* Cambridge University Press, 1997.

E-mail address: tdd2@cornell.edu

Cornell University