Algebra Colloquium **19** : 2 (2012) 263–282

Algebra Colloquium

Algorithms for Twisted Involutions in Weyl Groups*

R. Haas
Department of Mathematics and Statistics, Smith College
Northampton, MA 01063, USA
E-mail: rhaas@smith.edu

A.G. Helminck
Department of Mathematics, North Carolina State University
Raleigh, N.C. 27695-8205, USA
E-mail: loek@math.ncsu.edu

Abstract. Let (W, Σ) be a finite Coxeter system, and θ an involution such that $\theta(\Delta) = \Delta$, where Δ is a basis for the root system Φ associated with W, and $\mathcal{I}_\theta = \{w \in W \mid \theta(w) = w^{-1}\}$ the set of θ-twisted involutions in W. The elements of $\mathcal{I}_\theta$ can be characterized by sequences in Σ which induce an ordering called the Richardson-Spinger Bruhat poset. The main algorithm of this paper computes this poset. Algorithms for finding conjugacy classes, the closure of an element and special cases are also given. A basic analysis of the complexity of the main algorithm and its variations is discussed, as well experience with implementation.

2000 Mathematics Subject Classification: 68W30, 20G15, 20G20, 22E15, 22E46, 43A85

Keywords: symmetric spaces, symbolic computation

1 Introduction

Twisted involutions are of fundamental importance in the study of reductive symmetric spaces and more general symmetric k-varieties and their representations. Real reductive symmetric spaces have been studied for almost a hundred years and they are key in many areas of mathematics and physics. They can be defined as the homogeneous spaces G/H, where G is a reductive Lie group and H is an open subgroup of the fixed point group of an involution of G. Initially mathematicians mainly studied the *Riemannian symmetric spaces*, i.e., the symmetric spaces for which H is a maximal compact subgroup of G. In the last 25 years the emphasis has shifted to include the general reductive symmetric spaces. Symmetric spaces are best known for their role in representation theory and many people have studied the representations associated with these real reductive symmetric spaces (see for example [5, 10, 11, 18, 26, 27, 32]).

*The first author is partially supported by DMS-0721661, while the second author is partially supported by N.S.F. Grant DMS-0532140 and NSA grant H98230-06-1-0098.

Similar spaces over other fields occur throughout mathematics. For any field k of characteristic not 2, a *symmetric k-variety* is defined as the homogeneous space G_k/H_k, where G is a reductive algebraic group defined over k and $H = G^\sigma$ the fixed point group of a k-involution σ of G. Here we have used the notation K_k for the set of k-rational points of an affine algebraic group K defined over k. The symmetric k-varieties over algebraically closed fields are also called symmetric varieties. Symmetric k-varieties over base fields other than the real numbers occur in many problems in representation theory (see [2, 33, 34]), geometry (see [1, 8, 9]), singularity theory (see [22, 24]), the study of character sheaves (see for example [13, 23]) and at the study of cohomology of arithmetic subgroups (see [31]). The p-adic symmetric spaces are also of importance for the study of automorphic forms.

Symmetric spaces are similar in structure to Lie groups, but with additional complications. In the last two decades much of the algebraic and combinatorial structure of Lie groups and their representations has been implemented in several excellent computer algebra packages, including LiE, GAP4, Chevie, Magma, Maple and Coxeter. Most of the structure for Lie groups was developed long ago, before the advent of symbolic computation. An excellent description of the structure can be found in Bourbaki [4].

For symmetric spaces there are almost no implemented algorithms. Indeed, to date there are very few algorithms at all. This is due not only to the higher level of complication mentioned above but also to the fact that most of the structure of these symmetric spaces has been proved using non-constructive analytic methods. Over the last 20 years A.G. Helminck gave algebraic proofs for much of this structure, extending at the same time the results to general fields. Using this algebraic formulation he gave the first algorithms for computations in symmetric spaces (see [19, 20]). These algorithms computed the orbits of a minimal parabolic k-subgroup acting on a symmetric space. A description of these orbits and their closures also determines some of the fine structure of these symmetric spaces and is in fact the essential building blocks in the structure of these symmetric spaces.

While the algorithms in [19, 20] were important as the first algorithms to compute some of the structure of these symmetric spaces, these algorithms were quite inefficient and allowed one to compute only small cases. Since the orbits of a minimal parabolic k-subgroup P acting on the symmetric k-variety G_k/H_k play a fundamental role in the study of representations associated with these symmetric k-varieties, we started to look for other ways to characterize these orbits, hoping both to compute them more efficiently and to reveal more of their structure. The traditional ways to characterize these orbits are as P_k-orbits acting on the symmetric k-variety G_k/H_k by θ-twisted conjugation (i.e., if $x, g \in G_k$ then define $g * x := gx\theta(g)^{-1}$), or as the number of H_k-orbits acting on the flag variety G_k/P_k by conjugation or also as the set $P_k\backslash G_k/H_k$ of (P_k, H_k)-double cosets in G_k. The last is the same as the set of $P_k \times H_k$-orbits on G_k. Using these approaches these orbits were characterized for k algebraically closed by Springer [30], for $k = \mathbb{R}$ characterizations were given by Matsuki [25] and Rossmann [29] and for general fields these orbits were characterized by Helminck and Wang [21]. While these characterizations work very well for theoretical purposes, they did not lead to efficient algorithms. For computational

considerations it is better to take a different approach. One can first consider the set of (P, H)-double cosets in G. Then the (P_k, H_k)-double cosets in G_k can be characterized by the (P, H)-double cosets in G defined over k plus an additional (field dependent) invariant describing the decomposition of a (P, H)-double coset into (P_k, H_k)-double cosets. The (P, H)-double cosets in G can be characterized by combinatorial data, which are highly suited for computation, as follows. Let $A \subset P$ be a θ-stable maximal k-split torus of G, N the normalizer of A in G_k, Z the centralizer of A in G_k, let $W = N/Z$ be the Weyl group of A in G_k, and let $\mathcal{I}_\theta = \{w \in W \mid \theta(w) = w^{-1}\}$ be the set of *θ-twisted* involutions in W. We will also call these twisted involutions when it is clear which involution θ we are using. The (P, H)-double cosets in G defined over k can be characterized by the pairs $(\mathcal{O}, w)$, where $\mathcal{O}$ is a closed orbit and $w \in \mathcal{I}_\theta \subset W$ (see [14]). In order to classify the $P \times H$ orbits on G one needs to determine both the closed orbits and the twisted involutions that occur. In many cases there is a unique closed orbit and then the $P \times H$ orbits on G are completely characterized by the twisted involutions in W.

The twisted involutions also play an essential role in the study of the orbit closures. The closure of a $P \times H$ orbit on G decomposes as the union of other $P \times H$ orbits on G and these orbit closures are of fundamental importance in the study of Harish-Chandra modules (see [34]). There is a natural order $\succ$ on the set of $P \times H$ orbits on G, called the *Bruhat order*, which is defined as follows. If $\mathcal{O}_1, \mathcal{O}_2$ are orbits, then $\mathcal{O}_1 \succ \mathcal{O}_2$ if and only if $\mathcal{O}_2$ is contained in the closure of the orbit $\mathcal{O}_1$. This order on the orbits induces an order on the related set of twisted involutions, which we call the *Bruhat order* on $\mathcal{I}_\theta$. In [28] Richardson and Springer gave a combinatorial description of these Bruhat orders in terms of sequences of reflections in simple roots. The geometry of these orbits and their closures induce a poset structure on the set $\mathcal{I}_\theta$. Understanding this poset structure is not only key to understanding the structure of the orbits but also of fundamental importance in the studying the representations associated with the related symmetric k-varieties. In this paper we will give algorithms to compute the poset of twisted involutions, its Bruhat order and orbit closures. The poset of twisted involutions is a combinatorial structure and can completely be defined in the setting of Coxeter groups. The combinatorial and geometric structure of the poset of twisted involutions is of interest and importance for other areas of mathematics and computer science as well. The poset is related to the Bruhat lattice for Coxeter groups, which has been studied extensively by combinatorists for decades and has a very interesting geometrical structure. There are many open questions about this poset of twisted involutions. Having algorithms to compute this poset and its structure will be useful for further studies of this poset.

In Section 2 we set the notation and recall some properties about twisted involutions. In Section 3 the main algorithm and variations for computing the poset of twisted involutions are presented. In Section 4 we discuss the computational complexity. Section 5 describes our experience with implementing the algorithms. The final section describes a combinatorial data structure and its usefulness. We thank our student Kathryn Brenneman for her excellent assistance in the implementation. The Appendix contains the implementation of the basic Algorithm 1 in the Coxeter Maple package. Code for the other algorithms is available from the authors.

2 Preliminaries and Recollections

Let W be a Weyl group generated by a set of reflections Σ. We will assume that Σ comes from a basis Δ for the root system associated with W, i.e., (W, Σ) is a Coxeter System. If θ is an involution of W then the set $\mathcal{I}_\theta = \{w \in W \mid \theta(w) = w^{-1}\}$ denotes the set of θ-twisted involutions in W. As shown in [14] it is sufficient to consider only those involutions θ such that $\theta(\Delta) = \Delta$, and hence $\theta(\Sigma) = \Sigma$. The Weyl group W acts on the set $\mathcal{I}_\theta$ by *twisted conjugation* which is defined as $\overline{w}a = wa\theta(w)^{-1}$ where $w \in W$ and $a \in \mathcal{I}_\theta$. This operation is sometimes denoted as $w * a$. If $s \in \Sigma$ and $a \in \mathcal{I}_\theta$ then define $s \circ a = sa$ (group multiplication) if $\overline{s}a = a$ and $s \circ a = \overline{s}a$ otherwise. A sequence $\mathbf{s} = (s_1, \ldots, s_k)$ in Σ induces a sequence in $\mathcal{I}_\theta$ defined by induction as follows: $\mathbf{a}(\mathbf{s}) = (a_0, a_1, \ldots, a_k)$, where $a_0 = e$ and $a_i = s_i \circ a_{i-1} = s_i \circ \cdots \circ s_1 \circ e$ for $i \in [1, k]$. It will be important to keep track of for which elements in $\mathbf{s}$ the $^-$ (bar) operation is used. Thus for $s \in \Sigma$, $a \in \mathcal{I}_\theta$ we will use the notation $\overline{s_i}$ if $s \circ a_{i-1} = \overline{s_i}a_{i-1}$. Define $\overline{\Sigma} := \{\overline{s} \mid s \in \Sigma\}$ and let $\mathbf{r_s} = (r_1, r_2, \ldots, r_k)$ be the sequence in $\Sigma \cup \overline{\Sigma}$ defined by $r_i = \overline{s_i}$ if $s_i \circ a_{i-1} = \overline{s_i}a_{i-1}$ in $\mathbf{a}(\mathbf{s})$ and $r_i = s_i$ otherwise. A sequence $\mathbf{r} \in \Sigma \cup \overline{\Sigma}$ is called an *admissible sequence* if it is induced from a sequence in Σ by this process. Recall $l(w)$, the *length* of w with respect to Σ, is the number of elements in a minimal expression of w as a (usual) product of basis elements. Note that for twisted involutions, this is not generally equal to the number of elements in $\Sigma \cup \overline{\Sigma}$ in a sequence determining w as an element of $\mathcal{I}_\theta$. The sequence $\mathbf{r}_s$ is called an *ascending admissible sequence* for $\mathcal{I}_\theta$ if $0 = l(a_0) < l(a_1) < \cdots < l(a_k)$. Richardson and Springer [28] showed that every element in $\mathcal{I}_\theta$ can be represented by ascending admissible sequences. An element may be represented by several ascending admissible sequences, and determining all of these will be important. We use xy to denote regular group multiplication of x and y and the conjugacy action $\overline{s}x = sxs$ by $\overline{s}x$. For a sequence $\mathbf{r} = (r_1, r_2, \ldots, r_k)$ where $r_i \in \Sigma \cup \overline{\Sigma}$ and $w \in W$, define $\mathbf{r} \cdot w := r_k \cdots r_2 r_1 w$. For example, $(\overline{s_1}, s_2, \overline{s_3}, s_4) \cdot w = s_4\overline{s_3}s_2\overline{s_1}w = s_4s_3s_2s_1ws_1s_3$.

Implicit in the definition of an ascending admissible sequence for an element $w \in \mathcal{I}_\theta$, is an associated expression for w. This expression will be reduced. All reduced expressions for w contain the same number of elements, namely $l(w)$. Hence, if $(r_1, r_2, \ldots, r_{n_1})$ is an ascending admissible sequence for w then $l(w) = 2\,|(r_1, r_2, \ldots, r_{n_1}) \cap \overline{\Sigma}| + |(r_1, r_2, \ldots, r_{n_1}) \cap \Sigma|$, where multiple occurrences of s_i or $\overline{s_i}$, are counted multiple times. While this limits the possible number of elements in an ascending admissible sequence for w something stronger is in fact true.

Proposition 1. [28] *If $(r_1, r_2, \ldots, r_{n_1})$ and $(t_1, t_2, \ldots, t_{n_2})$ are two ascending admissible sequences in $\Sigma \cup \overline{\Sigma}$ and $\mathbf{r} \cdot e = \mathbf{t} \cdot e$ then $n_1 = n_2$.*

The *rank* of an element $w \in \mathcal{I}_\theta$ is the number of elements in an ascending admissible sequence for it. Let w_0 denote the unique longest element of the group W. For all θ, w_0 is an element of $\mathcal{I}_\theta$. Denote the rank of w_0 by k_0.

We extend the notion of an admissible sequence as follows. Given an element $w \in \mathcal{I}_\theta$ and a sequence $\mathbf{s} = (s_1, \ldots, s_k)$ in Σ, induce a sequence in $\mathcal{I}_\theta$ defined by induction as follows: $\mathbf{a}(\mathbf{s}) = (a_0, a_1, \ldots, a_k)$, where $a_0 = w$ and $a_i = s_i \circ a_{i-1} = s_i \circ \cdots \circ s_1 \circ e$ for $i \in [1, k]$. Let $\mathbf{r_s} = (r_1, r_2, \ldots, r_k)$ be the sequence in $\Sigma \cup \overline{\Sigma}$ defined

by $r_i = \overline{s_i}$ if $s_i \circ a_{i-1} = \overline{s_i} a_{i-1}$ in $\mathbf{a}(\mathbf{s})$ and $r_i = s_i$ otherwise. A sequence $\mathbf{r} \in \Sigma \cup \overline{\Sigma}$ is called a *w-admissible sequence* if it is induced from a sequence in Σ by this process. The sequence $\mathbf{r}$ is called an *ascending w-admissible sequence* for $\mathcal{I}_\theta$ if for all i, $l(r_i r_{(i-1)} \cdots r_2 r_1 w) > l(r_{(i-1)} \cdots r_2 r_1 w)$. The sequence $\mathbf{r}$ is called a *descending w-admissible sequence* for $\mathcal{I}_\theta$ if for all i, $l(r_i r_{(i-1)} \cdots r_2 r_1 w) < l(r_{(i-1)} \cdots r_2 r_1 w)$. An ascending e-admissible sequence will be simply an ascending admissible sequence, and a descending w_0-admissible sequence will be simply a descending admissible sequence.

Relations between $\mathcal{I}_\theta$ and $\mathcal{I}_{\mathrm{id}}$. Theorem 1.2 of [14] gives a relation between $\mathcal{I}_\theta$ and $\mathcal{I}_{\mathrm{id}}$ whenever the root system, Weyl group, etc. come from a maximal k-split torus of a reductive algebraic group G defined over a field k and the involution θ is the restriction of an involution of the group G to the root system of the maximal k-split torus (see [14] for more details). The involutions that arise in this manner only lead to nontrivial diagram automorphisms of irreducible root systems for which $-\mathrm{id} \notin W$. In particular, the diagram automorphism of D_{2n} does not occur in this manner and this result also does not hold for D_{2n}. A discussion of the poset for D_{2n} can be found in [15]. This gives a useful tool for producing and studying the poset of $\mathcal{I}_\theta$.

Theorem 1. [14, Theorem 1.2] *Let (W, Σ) be a Coxeter system, and θ an involution such that $\theta(\Delta) = \Delta$ as above. Denote by $\cdot$ the action of a sequence $\mathbf{r} \in \Sigma \cup \overline{\Sigma}$ on $\mathcal{I}_\theta$, and $\cdot'$ the action of $\mathbf{r}$ on $\mathcal{I}_{\mathrm{Id}}$.*

(i) *A sequence $\mathbf{r} = (r_1, r_2, \ldots, r_n)$ in $\Sigma \cup \overline{\Sigma}$ is an ascending admissible sequence for $\mathcal{I}_\theta$ if and only if it is an descending w_0-admissible sequence for $\mathcal{I}_{\mathrm{Id}}$.*

(ii) *A sequence $\mathbf{r} = (r_1, r_2, \ldots, r_n)$ in $\Sigma \cup \overline{\Sigma}$ is an descending w_0-admissible sequence for $\mathcal{I}_\theta$ if and only if it is also an ascending admissible sequence for $\mathcal{I}_{\mathrm{Id}}$.*

(iii) *Let $\mathbf{q}$ and $\mathbf{r}$ be two sequences in $\Sigma \cup \overline{\Sigma}$. They are two ascending admissible sequences for $\mathcal{I}_\theta$ and $\mathbf{q} \cdot e = \mathbf{r} \cdot e$ in $\mathcal{I}_\theta$ if and only if they are two descending w_0-admissible sequences for $\mathcal{I}_{\mathrm{Id}}$ and $\mathbf{q} \cdot' w_0 = \mathbf{r} \cdot' w_0$ in $\mathcal{I}_{\mathrm{Id}}$.*

In [15] we show there is a correspondence between $\mathcal{I}_\theta$ and $\mathcal{I}_{\mathrm{Id}}$ in all cases as follows. Let Φ be a root system with Weyl group W and $\theta \in \mathrm{Aut}(\Phi)$ an involution. Let $\mathcal{I}_\theta = \{w \in W \mid \theta(w) = w^{-1}\}$. Let $w_\theta \in W$ be the unique element such that

$$\theta(\Phi^+) = w_\theta(\Phi^+).$$

and let $\theta' = \theta w_\theta$. Note that if Φ is irreducible, then θ' is the identity or a diagram automorphism. The elements w_θ and θ' satisfy the following conditions:

Proposition 2. [15] *Let Φ, Φ^+, θ, w_θ and θ' be as above. Then we have the following properties:*

(i) $w_\theta \in \mathcal{I}_\theta$.

(ii) $\theta'(\Phi^+) = \Phi^+$.

(iii) *θ' is an involution of Φ.*

(iv) $\mathcal{I}_{\theta'} = \mathcal{I}_\theta \cdot w_\theta$.

To relate $\mathcal{I}_\theta$ and $\mathcal{I}_{\mathrm{id}}$ we first note the following:

Lemma 1. [15] $\mathcal{I}_{\mathrm{id}} = \mathcal{I}_{-\mathrm{id}}$.

Corollary 1. [15] *Assume the involution θ of Φ is induced from an involutorial automorphism of a related semisimple group G, like in* [14]. *Let $w_0 = w_0(-\mathrm{id})$ be the longest involution in the Weyl group. If θ' is a non-trivial diagram automorphism, then $\mathcal{I}_{\theta'} = \mathcal{I}_{-\mathrm{id}} \cdot w_0 = \mathcal{I}_{\mathrm{id}} \cdot w_0$.*

3 Algorithms for Elements and Poset of $\mathcal{I}_\theta$

The main goal of this paper is to give an algorithm for determining all ascending admissible sequences for all elements of $\mathcal{I}_\theta$, and their poset of relations. We begin with the simplest case where $\theta = \mathrm{Id}$. This algorithm is the main algorithm of this paper. Two methods of calculating $\mathcal{I}_\theta$, for θ a diagram automorphism, will make use of this algorithm.

3.1 Algorithm for elements and poset of $\mathcal{I}_{\mathrm{Id}}$. For each element w in $\mathcal{I}_{\mathrm{Id}}$ we keep track of the following: (i) its rank, (ii) pointers from elements x of rank one less and the element $s \in \Sigma \cup \overline{\Sigma}$ such that $w = sx$. Note that we do not need to have an explicit representation of w. Ascending admissible sequences (and thus reduced representations) of an element w can be retrieved by following the labelled pointers from the identity element to w.

Algorithm 1. *Input Weyl group. Output poset for $\mathcal{I}_{\mathrm{Id}}$.*

0. *Let the stack consist of the identity element of rank* 0, *and set $k = 0$.*

I. *For each x of rank k from the stack, do*

   A. *For each s_i such that there is no pointer into x labelled s_i or $\overline{s_i}$ (no ascending admissible sequence for x begins with s_i or $\overline{s_i}$) do:*
      (i) *Calculate $y' = s_i x s_i$. If $y' \neq x$ then let $y = y'$ which is of rank $k+1$. If $y' = x$ then let $y = s_i x$, which is of rank $k+1$.*
      (ii) *For each element z of rank $k+1$, check if $y = z$ if so put a pointer from x to y labelled $s_i, (\overline{s_i})$ (add sequence of y to list of ascending admissible sequences for z). If no such z add y to stack as new element of rank $k+1$, with a pointer from x labelled $s_i, (\overline{s_i})$.*

II. *If $k < k_0$, let $k = k+1$ and go to step* [I].

The largest value of k that must be considered is for $k = k_0$ the rank of w_0 the longest element in the Weyl group, which is always an element of $\mathcal{I}_\theta$ as well. In Section 4 we give the maximum rank for each of the classical Weyl groups. Even without knowing the explicit rank k_0 of w_0 the algorithm will end after reaching w_0 since for every $s \in \Sigma$, there is a pointer to w_0 labelled s or $\overline{s}$ (see [14]).

3.2 The general $\mathcal{I}_\theta$ case. A direct modification of Algorithm 1 leads to an algorithm for the elements and poset of $\mathcal{I}_\theta$. All that is required is to replace step (i) by:

(i′) Calculate $y' = s_i x \theta(s_i)^{-1}$. If $y' \neq x$ then let $y = y'$ which is of rank $k+1$. If $y' = x$ then let $y = s_i x$, which is of rank $k+1$.

Remark 1. Most computer algebra packages for Weyl groups use generators and relations and the Weyl group elements are expressed as reduced words in the generators. These reduced expressions are not unique. To check if two elements y and z are the same one usually checks if $zy^{-1} = e$. In step (ii) of Algorithm 1 it suffices to check if $zy = e$, since all elements in $\mathcal{I}_{\mathrm{Id}}$ are of order 2. Therefore the complexity of step (ii) in Algorithm 1 is less than that in the variation for $\mathcal{I}_\theta$.

A discussion of the computational complexity of Algorithm 1 and the general version is given in Section 4. Another option for computing the poset of $\mathcal{I}_\theta$ comes from Theorem 1. It implies the following algorithm for finding $\mathcal{I}_\theta$. The only work here is to change the orientation of all pointers. Thus while for sequences in $\mathcal{I}_\theta$, the operation of $\overline{s}w$ means the twisted product $sw\theta(s^{-1})$, this simple translation scheme does not require performing any twisted computations.

3.3 Algorithm for elements and poset of $\mathcal{I}_\theta$.

I. Run Algorithm 1.

II. For each element w of rank k found in Algorithm 1 associate an element w' of rank $k_0 - k$ in $\mathcal{I}_\theta$.

III. For each pointer $x \to w$ with label $s \in \Sigma \cup \overline{\Sigma}$ found in Algorithm 1, associate a pointer $w' \to x'$ in $\mathcal{I}_\theta$ with label $s \in \Sigma \cup \overline{\Sigma}$.

3.4 Special cases. From [14, Corollary 5.1] we have the following.

Corollary 2. *If Φ is of type $A_1, B_n, C_n, E_7, E_8, F_4$ or G_2, then every ascending admissible sequence is also a descending admissible sequence and vice versa. That is, the poset is symmetric.*

Hence in these cases, only the top half of the poset needs to be found. That is:

3.5 Algorithm for symmetric cases of $\mathcal{I}_\theta$.

I. Run Algorithm 1 (or 3.2) but stop when $k = \lceil k_0/2 \rceil$.

II. For each element w of rank $k < \lceil k_0/2 \rceil$ found in step I. associate an element w^* of rank $k_0 - k$.

III. For each pointer $x \to w$ with label $s \in \Sigma \cup \overline{\Sigma}$ found in step I, associate a pointer $w^* \to x^*$ in $\mathcal{I}_\theta$ with label $s \in \Sigma \cup \overline{\Sigma}$.

IV. Note the elements of rank $\lceil k_0/2 \rceil$ have representations as ascending and descending sequences. Associate pairs of these elements that are equal.

3.6 Conjugacy classes. An element x is conjugate to w if and only if there exist a w-admissible sequence $\mathbf{r}$ with $r_i \in \overline{\Sigma}$ such that $\mathbf{r} \cdot w = x$.

A variation of Algorithm 3.2 (or Algorithm 1 as appropriate) can be used to find the conjugacy class of an element w of $\mathcal{I}_\theta$. In this case begin the algorithm with w in the stack and do not keep track of rank but rather rank relative to the rank of w which we will call distance from w and denote by $d_w(x)$ = length of shortest path in the poset between x and w. Note that the rank of x is in fact $d_e(x)$, the distance from the identity element to x.

Algorithm 2. (Conjugacy in $\mathcal{I}_{\mathrm{Id}}$) *Input Weyl group W and element $w \in W$. This algorithm will return the set of elements conjugate to w.*

0. *Let the stack consist of the element w, where $d_w(w) = 0$, and set $k = 0$.*

I. *If there is no element x in the stack with $d_w(x) = k$ stop.*
For each x in the stack such that $d_w(x) = k$, do
A. *For each s_i such that there is no pointer into x labelled $\overline{s_i}$ (no admissible sequence for x begins with $\overline{s_i}$) do:*
(i) *Calculate $y' = s_i x s_i$. If $y' \neq x$ then let $y = y'$ and $d_w(y) = k + 1$.*
(ii) *For each element z such that $d_w(z) = k+1$, check if $y = z$ if so put a pointer from x to z labelled $\overline{s_i}$ (add sequence of y to list of admissible sequences for z). If no such z add y to stack as new element of rank $k + 1$, with a pointer from x labelled $\overline{s_i}$.*

II. *Let $k = k + 1$ and go to step* I.

It is clear that this algorithm will compute all elements conjugate to w. It is perhaps less clear that it will correctly check whether two elements conjugate to w are equal. Let y be an element conjugate to w such that $r_k \cdots r_1 w = t_{k+j} \cdots t_{k+1} t_k \cdots t_1 w = y$. The set of elements in both these paths of conjugate elements together form a cycle of length $2k+j$ in the poset. Since a poset is a bipartite graph this must be an even number. Hence j is even. Since the algorithm finds all conjugate elements to w of a fixed distance from w before proceeding to the next bigger distance from w it will find the representation of y as $r_k \cdots r_1 w$ first at level k. The other representation of y will be implicit in the cycle. These two paths (representations) will intersect at level $k+j/2$ at the unique element in the cycle furthest from w. That is we find $t_{(k+j/2+1)} \cdots t_{k+j} r_k \cdots r_1 w$ and $t_{k+j/2} \cdots t_{k+1} t_k \cdots t_1 w$ at the same level.

3.6.1 Conjugacy for $\mathcal{I}_\theta$. As usual there are two choices here. We can either replace step (i) above with

(i′) Calculate $y' = s_i x \theta(s_i^{-1})$. If $y' \neq x$ then let $y = y'$ and $d_w(y) = k + 1$.

Or, run the algorithm for $\mathcal{I}_{\mathrm{Id}}$ and convert the results to $\mathcal{I}_\theta$ as in Algorithm 3.3. The second option requires knowing which element in $\mathcal{I}_{\mathrm{id}}$ is associated with w. In the case that the involution θ is the restriction of an involution of a reductive algebraic group G to the root system of a maximal k-split torus as in [14] (or equivalently if for each irreducible component of the root systems for which the induced diagram automorphism is non-trivial we have $-\mathrm{id} \notin W$) we have the following results.

Corollary 3. *Let (W, Σ) be a Coxeter system, and θ an involution such that $\theta(\Delta) = \Delta$. Then for any $w \in \mathcal{I}_{\mathrm{Id}}$, $\mathbf{r} = (r_1, r_2, \ldots, r_n)$ in $\Sigma \cup \overline{\Sigma}$ is an ascending admissible sequence for $w \in \mathcal{I}_{\mathrm{Id}}$ if and only if it is a descending w_0-admissible sequence for $ww_0 \in \mathcal{I}_\theta$.*

Proof. This corollary of Theorem 1 follows by induction, and the fact that for all nontrivial diagram automorphisms $\theta w_0 = -1$. □

In fact, a more general result is true:

Theorem 2. *Let (W, Σ) be a Coxeter system, and θ an involution. Then there exists an element $w_\theta \in W$ such that $w \in \mathcal{I}_{\mathrm{id}}$ if and only if $ww_\theta \in \mathcal{I}_\theta$.*

3.7 Closure of an element. An element $x \in \mathcal{I}_\theta$ is in the *+closure* of w if and only if there exists an ascending w-admissible sequence $\mathbf{r} = (r_1, r_2, \ldots, r_k)$ where

$r_i \in \Sigma \cup \overline{\Sigma}$ such that $r \cdot w = x$. The $-$closure of w is similarly defined to be all x such that there exists a descending w-admissible sequence $\mathbf{r} = (r_1, r_2, \ldots, r_k)$ where $r_i \in \Sigma \cup \overline{\Sigma}$ such that $r \cdot w = x$.

The main algorithm can also be modified to give the $(\pm)$closure of any particular element. To do this we must keep track of the length of each element found as well as the distance from w.

Algorithm 3. (Closure for $\mathcal{I}_{\mathrm{Id}}$) *Input an element w and specify Weyl group W. This algorithm will return the set of elements in the $+$closure of w.*

0. *Let the stack consist of w, where $d_w(w) = 0$, and set $k = 0$.*

I. *For each x such that $d_w(x) = k$ from the stack, do*

A. *For each s_i such that there is no pointer into x labelled s_i or $\overline{s_i}$ (no ascending w-admissible sequence for x begins with s_i or $\overline{s_i}$) do:*

(i) *Calculate $y' = s_i x s_i$. If $y' \neq x$ and $l(y') > l(x)$ then let $y = y'$ and $d_w(y') = k + 1$. If $y' = x$ and $l(s_i x) > l(x)$ then let $y = s_i x$, and $d_w(y') = k + 1$.*

(ii) *If $y = w_0$ stop.*

(iii) *For each element z with $d_w(z) = k + 1$, check if $y = z$ if so put a pointer from x to y labelled $s_i, (\overline{s_i})$ (add sequence of y to list of ascending admissible sequences for z). If no such z add y to stack as new element of distance $k+1$, with a pointer from x labelled $s_i, (\overline{s_i})$.*

II. *Let $k = k + 1$ and go to step* I.

The algorithm will find all elements in the closure, but as written does not find the whole closure poset, but stops after completing one path to w_0 from w. To compute the entire closure poset step (ii) should be removed and the stopping rule added to step II, which should then read

II. If $l(y) < l(w_0)$ then let $k = k + 1$ and go to step I.

For the $-$closure, simply change the orientation of the inequality signs in step (i). For $\mathcal{I}_\theta$, as usual, we can either change the computation in step (i) to $y' = s_i x \theta(s_i^{-1})$ or, translate the results from the $\mathcal{I}_{\mathrm{Id}}$ case.

4 Complexity of Main Algorithms

We begin by discussing the complexity of Algorithm 1. The loop of step I will be done for each element of $\mathcal{I}_{\mathrm{Id}}$, except for w_0. The following proposition gives $|\mathcal{I}_\theta|$.

Proposition 3.

(i) *For A_{n-1} there are $\sum_{k=0}^{\lfloor n/2 \rfloor} \frac{n!}{(n-2k)!2^k k!}$ elements total in $\mathcal{I}_{\mathrm{Id}}$, and $\mathcal{I}_\theta$.*

(ii) *For B_n there are $\sum_{k=0}^{\lfloor n/2 \rfloor} \frac{n!2^{n-2k}}{(n-2k)!k!}$ elements total in $\mathcal{I}_{\mathrm{Id}}$.*

(iii) *For n odd, D_n has $\sum_{k=0}^{\lfloor n/2 \rfloor} \frac{n!2^{n-2k-1}}{(n-2k)!k!}$ elements total in $\mathcal{I}_{\mathrm{Id}}$, and $\mathcal{I}_\theta$.*

(iv) *For n even, D_n has $\frac{n!}{(n/2)!} + \sum_{k=0}^{\lfloor n/2 \rfloor - 1} \frac{n!2^{n-2k-1}}{(n-2k)!k!}$ elements total in $\mathcal{I}_{\mathrm{Id}}$, and there are $\frac{-(n-1)!}{((n-2)/2)!} + \sum_{k=0}^{\lfloor (n-2)/2 \rfloor} \frac{(n-1)!2^{(n-2-2k)}(n-2k+1)}{k!(n-1-2k)!}$ in $\mathcal{I}_\theta$ for the non-trivial automorphism.*

Proof. For all cases except D_{2n}, the results in [14] and Section 2 show it is sufficient to consider the case where $\theta = \mathrm{Id}$. In each case the result for $\mathcal{I}_{\mathrm{Id}}$ follows by counting the number of ways to break up $\{1, \ldots, n\}$ into 2-cycles and 1-cycles and assigning signs to them (see also [6]). The result for $\mathcal{I}_\theta$ in D_{2n} can be found in [15]. □

It is useful to think of the poset of elements of $\mathcal{I}_{\mathrm{Id}}$ as a labelled directed graph. For type A_n, B_n and D_n, this graph is regular of degree n, that is, each element will have a total of n edges incident to it, one for each basis element. The number of s_i and $\overline{s_i}$ which are the first element of an ascending admissible sequence of x will vary, and these correspond to the number of edges into x. However, the total number of edges is equal to $(n-1)/2$ times the total number of vertices. Proposition 3 provides the number of elements in $\mathcal{I}_{\mathrm{Id}}$ in all classical cases. The number of edges is then $E = |\mathcal{I}_{\mathrm{Id}}|(n-1)/2$. Thus the work of step (i) will be done exactly E times. One occurrence of step (i) requires two multiplications by basis elements and a comparison of two elements. Each occurrence of step (ii) requires a comparison. Step (ii) will occur once for each pair of elements of each rank. While we do not know how many elements there will be of each rank we can get an upper bound for this quantity. The worst case would occur if there was exactly one element of each of all but one rank. The maximum rank that occurs is the rank of w_0 which is given for all classical cases in Theorem 3, the proof of which is in [16]. We refer to the sum of the number of pairs of elements of each rank as H. Thus in total there will be $2E$ multiplications and $E + H$ comparisons.

Theorem 3.

(i) *For the Weyl group of type A_{n-1}, the longest element w_0 has rank $(n^2-1)/4$ if n is odd, $n^2/4$ if n is even.*

(ii) *For the Weyl group of type B_n, the longest element w_0 has rank $(n+1)n/2$.*

(iii) *For the Weyl group of type D_n, the longest element w_0 has rank $n^2/2$ if n is even and $(n+1)(n-1)/2$ if n is odd.*

In Algorithm 3.2 each occurrence of step (i′) requires computing an inverse, computing $\theta(s_i)$, as well as two multiplications and a comparison. The number of times each step is done is the same. This results in a total of $2E$ multiplications, $E+H$ comparisons, E inverses, and E computations of $\theta(s_i)$. The difficulty of each of these types of operations will depend on the data structure used to represent the Weyl group elements.

On the other hand, Algorithm 3.3, requires running Algorithm 1, followed by a relabelling and redirecting of each vertex and edge in the poset graph of $\mathcal{I}_{\mathrm{Id}}$. That is, there are a total of $2E$ multiplications, $E + H$ comparisons and $E + H$ relabellings. This will be less work than Algorithm 3.2. However, variations of 3.2 may prove useful when only subgraphs of the the poset of $\mathcal{I}_\theta$ are needed (see Section 3.6).

If the elements of the Weyl group are represented as strings of generators (reduced representations) then multiplying two such elements is a trivial concatenation. On the other hand, comparing two reduced expressions to see if they represent the same element is a complicated process. It can be done by finding the inverse of one (which is just reversing the order of the generators in the reduced expression) and

then multiplying it by the other and then reducing by the known list of relations to see if the result is the identity.

5 Implementation

The algorithms in this paper can be implemented in any of the computer algebra programs which contain Weyl packages, including Magma, LiE, Chevie, GAP4 and John Stembridge's Coxeter Maple package. We implemented these algorithms in the Coxeter Maple package (see appendix), where we computed the poset for $\mathcal{I}_\theta$ for classical groups up to type A_7 and B_7. Running A_7 took approximately 6 hours, the computer crashed after 3 days of working on A_8. When we attempted to run E_7 the computer also ran out of memory after about 3 days. In all cases the work was done on a double processor 2.7 GHz Macintosh with 8GB of RAM.

We expect this inability to run large examples is due to the fact that in existing Weyl group packages the elements do not have unique representation. As described above, our algorithms require determining if two elements are the same a large number of times. It would thus seem more efficient to use a unique representation for Weyl group elements. We have started working on such a package and find a phenomenal increase in efficiency. We implemented algorithms for the Weyl group of all classical types. Runs of these algorithms showed more than a 1000 fold increase in efficiency. For example computing the involution poset in the case of A_6 in this new implementation took less than 1 second, while using the Coxeter package it took 18 minutes and 15.72 seconds. The Weyl group of type A_7 took 3.08 seconds in the new implementation and 6 hours, 5 minutes and 48.16 seconds for a total of 21948.16 seconds, which is almost 7000 times as fast. The case of A_8 only took 14.52 seconds while we were unable to compute the involution poset for this case using the Coxeter package.

The advantage of using an existing specialized package is clearly that one can take advantage of all the other structures of Weyl groups and root systems already implemented in such a package. However, for some algorithms, the computational disadvantages that stem from using non-unique representations may be significant. After the success of implementing this algorithm for A_n using a unique representation scheme, we are intending to build a complete Weyl group package suitable for all Weyl groups that can do the array of computations as done by the standard packages (such as Coxeter). This package should prove particularly useful for applications to symmetric spaces, for example, for algorithms included in recent work of Daniel [7] and Gagliardi [12].

Example 1. We conclude this section with an example of the poset as computed by the algorithm included in the appendix. For the example we choose the case of the Weyl group of type A_4, which is the symmetric group S_5. We denote the elements of the poset in one line notation. The poset was computer generated. From each element there is an edge for each group generator s_i. The edges will correspond to one of the two possible operations by the $\circ$ action. In Figure 1, the twisted conjugation operation by s_i is designated by a solid line while the left multiplication operation by s_i is designated by a dashed line.

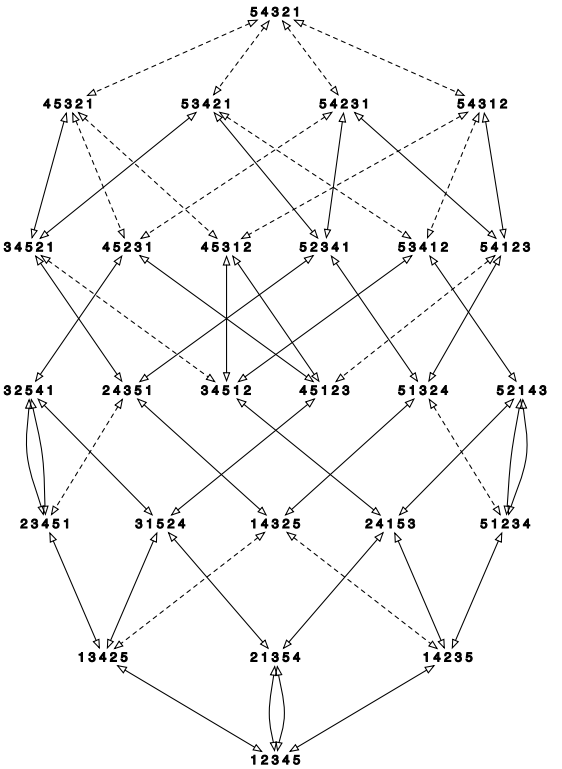

Figure 1. Poset of $\mathcal{I}_\theta$ for type A_4

6 Using Signed Permutation Vectors as a Data Structure

As we discussed in the previous section an implementation of the Weyl group using a unique representation scheme will lead to much more efficient algorithms. For the classical Weyl groups combinatorialists frequently use signed permutations, an efficient data structure, which gives unique representation and which we have found highly efficient for our computations. In this section we recall some of their properties, prove some additional ones and use them to derive more efficient algorithms to compute the twisted involution posets and the closures of elements.

6.1 Signed permutation vectors. In the root systems corresponding to Dynkin Diagrams of type A, B, C, and D, the roots are combinations of the standard basis vectors e_i. Weyl groups of root systems of type B and C are the same, hence from now on we only discuss type B. The elements of W can be completely described by their actions on the e_i. In particular, $w(e_i) = \pm e_j$ for all i, j not necessarily different than i. Written as a permutation, w can be expressed in two line notation as $\begin{pmatrix} 1 & 2 & 3 & \dots & n \\ a_1 & a_2 & a_3 & \dots & a_n \end{pmatrix}$, where $a_i = \pm e_k$. We will use the bottom row of this array as our unique representation of w. For a nonzero real number a, define $\mathrm{sgn}(a) = +$ if $a > 0$ and $\mathrm{sgn}(a) = -$ if $a < 0$. Thus we represent $w \in W$ by the vector $(a_1, a_2, \dots, a_n)$, where $w(\mathrm{sgn}(a_i)e_{|a_i|}) = e_i$. That is, the vector corresponds to a

permutation of the set $\{1,\dots,n,-1,\dots,-n\}$ with the additional restriction that $w(-i) = -w(i)$. Thus these are called signed permutations, and sometimes also called one line notation or window notation (see [3]).

Basis elements of W. For the Weyl group of type A_{n-1} each basis element s_i corresponds to the reflection over $e_i - e_{i+1}$ for $i = 1,\dots,n-1$. Thus these are $s_1 = (2,1,3,4,\dots,n)$, $s_2 = (1,3,2,4,\dots,n)$, etc. Groups of type B_n also have a reflection over e_n in the basis which is represented by the vector $s_n = (1,2,3,\dots,(n-1),-n)$. Type D_n will have a basis element that reflects over $e_{n-1}+e_n$ which is represented by the vector $s_{\hat{n}} = (1,2,3,\dots,-n,-(n-1))$.

Using this notation, it is easy to multiply any group element w by any basis element $s \in \Sigma$. For $1 \le i < n$, $(a_1,a_2,\dots,a_n)s_i = (a_1,a_2,\dots,a_{i+1},a_i,\dots,a_n)$. When multiplying on the right by s_i any negative signs move with the a_i. To calculate $s_i(a_1,a_2,\dots,a_n)$, find $k,l \in \{1,\dots,n\}$ such that $a_k = i$, and $a_l = i+1$ and then,

$$\begin{aligned} &s_i(a_1,a_2,\dots,a_n)\\ &= (a_1,a_2,\dots,a_{k-1},\operatorname{sgn}(a_k)|a_l|,a_{k+1},\dots,a_{l-1},\operatorname{sgn}(a_l)|a_k|,a_{l+1},\dots,a_n), \end{aligned}$$

that is switch the k-th and l-th elements but keep the signs in the positions they were. For the special basis elements of B_n, C_n and D_n we get $(a_1,a_2,\dots,a_n)s_n = (a_1,a_2,\dots,a_{(n-1)},-a_n)$, and $(a_1,a_2,\dots,a_n)s_{\hat{n}} = (a_1,a_2,\dots,-a_n,-a_{(n-1)})$. Again, operating on the left acts on numbers, so find $k,l \in \{1,\dots,n\}$ such that $a_k = n-1$, and $a_l = n$ and then $s_n(a_1,a_2,\dots,a_n) = (a_1,a_2,\dots,-a_l,\dots,a_n)$; and

$$\begin{aligned} &s_{\hat{n}}(a_1,a_2,\dots,a_k,\dots,a_l,\dots,a_n)\\ &= (a_1,a_2,\dots,-\operatorname{sgn}(a_k)|a_l|,\dots,-\operatorname{sgn}(a_l)|a_k|,\dots,a_n). \end{aligned}$$

We summarize these calculations in a proposition.

Proposition 4. *For $1 \le i < n$, then*

(i) $(a_1,a_2,\dots,a_n)s_i = (a_1,a_2,\dots,a_{i+1},a_i,\dots,a_n)$.

(ii) *If $a_k = i$ and $a_l = i+1$ then*

$$\begin{aligned} &s_i(a_1,a_2,\dots,a_n)\\ &= (a_1,a_2,\dots,a_{k-1},\operatorname{sgn}(a_k)|a_l|,a_{k+1},\dots,a_{l-1},\operatorname{sgn}(a_l)|a_k|,a_{l+1},\dots,a_n). \end{aligned}$$

(iii) $(a_1,a_2,\dots,a_n)s_n = (a_1,a_2,\dots,a_{(n-1)},-a_n)$.

(iv) $(a_1,a_2,\dots,a_n)s_{\hat{n}} = (a_1,a_2,\dots,-a_n,-a_{(n-1)})$.

(v) *If $a_l = n$ then $s_n(a_1,a_2,\dots,a_n) = (a_1,a_2,\dots,-a_l,\dots,a_n)$.*

(vi) *If $a_k = n-1$ and $a_l = n$ then*

$$\begin{aligned} &s_{\hat{n}}(a_1,a_2,\dots,a_k,\dots,a_l,\dots,a_n)\\ &= (a_1,a_2,\dots,-\operatorname{sgn}(a_k)|a_l|,\dots,-\operatorname{sgn}(a_l)|a_k|,\dots,a_n). \end{aligned}$$

More generally, multiplication can be described as follows. For $(a_1,a_2,\dots,a_n)x$, $(a_1,a_2,\dots,a_n)$ acts on x by moving the number a_i to the i-th position. While in $x(b_1,b_2,\dots,b_n)$, b acts on x by acting on positions. Formally we have:

Proposition 5.

$$(a_1,a_2,\dots,a_n)(b_1,b_2,\dots,b_n) = (\operatorname{sgn}(b_1)a_{|b_1|},\operatorname{sgn}(b_2)a_{|b_2|}\dots,\operatorname{sgn}(b_n)a_{|b_n|}).$$

Compared to using reduced expressions, multiplication requires a true calculation with this notation (but a simple one). However, because the representation is unique, determining if two elements are equal is trivial. Thus for each of the Algorithms 1, 3.2, 3.3, it is more efficient to use this new data structure. The trade off is of course that with this notation in our Algorithms 1, 3.2, 3.3, we are in fact storing both some reduced expressions for an element (through their associated ascending admissible sequences) and the unique representation as a signed permutation vector.

Lengths. The next theorem shows that it is easy to calculate the length of an element w when it is presented in this notation. Since the roots of all classical Weyl groups are of the form $\pm e_i \pm e_j$, and $\pm e_i$, the length of an element will only depend on examining at the e_i's singly or in pairs. For A_n the length is simply the number of inversions of the permutation. For an element $w = (a_1, a_2, \ldots, a_n)$, define $N(w) = |\{i : a_i < 0\}|$ and

$$p(a_i, a_j) = \begin{cases} 0 & \text{if } |a_i| < |a_j|,\ a_i > 0, \\ 2 & \text{if } |a_i| < |a_j|,\ a_i < 0, \\ 1 & \text{if } |a_j| < |a_i|. \end{cases}$$

Theorem 4. *If the signed permutation vector of $w \in W$ is $(a_1, a_2, \ldots, a_n)$ then the length of w is*

$$l(w) = \begin{cases} \sum_{i<j} p(a_i, a_j) & \text{in } A_n, D_n, \\ \sum_{i<j} p(a_i, a_j) + N(w) & \text{in } B_n, C_n. \end{cases}$$

Proof. The proof relies on the characterization of length as the number of positive roots mapped to negative roots under w. For $i < j$, the function $p(a_i, a_j)$ counts the number of roots of the set $\{e_i - e_j, e_i + e_j\}$ that are mapped to negative roots under w while $N(w)$ counts the number of roots of the form e_i that are mapped to negative roots. The table below shows the calculations for $p(a_i, a_j)$. In the table, the notation (x, y) is a shorthand for the order and sign of the elements a_i and a_j in w. Let $k < l$.

	e_i	e_j	$e_i - e_j$	$e_i + e_j$	p
(k, l)	e_k	e_l	$e_k - e_l$	$e_k + e_l$	0
$(k, -l)$	e_k	$-e_l$	$e_k + e_l$	$e_k - e_l$	0
$(-k, l)$	$-e_k$	e_l	$-e_k - e_l$	$-e_k + e_l$	2
$(-k, -l)$	$-e_k$	$-e_l$	$-e_k + e_l$	$-e_k - e_l$	2
(l, k)	e_l	e_k	$e_l - e_k$	$e_l + e_k$	1
$(-l, k)$	$-e_l$	e_k	$-e_l - e_k$	$-e_l + e_k$	1
$(l, -k)$	e_l	$-e_k$	$e_l + e_k$	$e_l - e_k$	1
$(-l, -k)$	$-e_l$	$-e_k$	$-e_l + e_k$	$-e_l - e_k$	1

This completes the proof. □

6.2. Characterizations of special types of elements. Many types of elements are easily characterized in signed permutation notation. The longest element in A_n is $(n, n-1, \ldots, 1)$. For B_n, C_n the longest element is $(-1, -2, \ldots, -n)$. For D_n the

longest element is also $(-1,-2,\ldots,-n)$ if n is even and $(-1,-2,\ldots,-(n-1),n)$ if n is odd. Below we give characterizations for inverses and members of $\mathcal{I}_{\mathrm{Id}}$ and $\mathcal{I}_\theta$.

Proposition 6. $(a_1,a_2,\ldots,a_n)^{-1} = (b_1,b_2,\ldots,b_n)$ *if* $|b_{|a_i|}| = i$, *and* $\mathrm{sgn}(b_i) = \mathrm{sgn}(a_{b_i})$.

It is difficult to tell whether an element given by generators is in $\mathcal{I}_{\mathrm{Id}}$ or $\mathcal{I}_\theta$. It takes work to determine if a given sequence in $\overline{\Sigma} \cup \Sigma$ is an ascending admissible sequence, and so represents an element in $\mathcal{I}_{\mathrm{Id}}$. On the other hand, we can quickly see if an element given as a signed permutation vector is in $\mathcal{I}_{\mathrm{Id}}$. The following results are from [17].

Lemma 2. *An element* $w = (a_1,a_2,\ldots,a_n)$ *is in* $\mathcal{I}_{\mathrm{Id}}$ *if and only if* $|a_{|a_i|}| = i$ *for all* i, *and* $\mathrm{sgn}(a_i) = \mathrm{sgn}(a_{|a_i|})$.

For the classical groups of type A_n and D_n we can also efficiently recognize elements of $\mathcal{I}_\theta$ as follows.

Proposition 7. *Let* W *be the Weyl group of type* A_n, *and* θ *the nontrivial diagram involution. An element* $w = (a_1,a_2,\ldots,a_n) \in W$ *is in* $\mathcal{I}_\theta$ *if and only if* $a_{(n+1)-a_i} = (n+1) - i$ *for all* i.

Proposition 8. *Let* W *be the Weyl group of type* D_n, *and* θ *the nontrivial diagram involution. An element* $w = (a_1,a_2,\ldots,a_n)$ *is in* $\mathcal{I}_\theta$ *if and only if each of the following conditions holds:*

(i) *For* $i \neq n$, *if* $a_i = n$ *then* $a_n = -i$.

(ii) *For* $i \neq n$, *if* $a_i = -n$ *then* $a_n = i$.

(iii) *For* $i \neq n$ *and* $j \neq n$, *if* $a_i = j$ *then* $a_j = i$.

(iv) *For* $i \neq n$ *and* $j \neq n$, *if* $a_i = -j$ *then* $a_j = -i$.

(v) *No restrictions if* $a_i = \pm i$ *(even if* $i = n$*).*

Converting between representations. As discussed above, a representation of an element w of $\mathcal{I}_{\mathrm{Id}}$ as an ascending admissible sequence implies a representation of w by a sequence of generators and from either of these the expression of w as a signed permutation vector can be calculated by multiplication. To calculate any or all admissible sequences for an element $w \in \mathcal{I}_{\mathrm{Id}}$ given as a signed permutation vector requires recognizing which operations will decrease the length. That is, when is $l(s_i w s_i) = l(w) - 2$.

The following algorithm will find one admissible sequence for w of type A_n, B_n or D_n.

Algorithm 4. (Find one admissible sequence for w) *Input Weyl group* W *and an element* $w \in \mathcal{I}_{\mathrm{Id}}$.

0. *Set* $i = 1$.
1. *If* $i = n$ *go to step* 5.
2. (i) *If* $|a_i| > |a_{i+1}|$ *and* $a_{i+1} > 0$ *go to step* 3. (ii) *OR* $|a_i| < |a_{i+1}|$ *and* $a_i < 0$ *then go to step* 3. *If neither of these conditions holds then* $i = i + 1$ *and go to step* 1.

3. (i) *If $i+1$ occurs before i and $i+1$ occurs as a positive entry ($a_l = i+1$ for some l) OR* (ii) *If i occurs before $i+1$ and i occurs as a negative entry. If either one of these is true then $w =: s_i w s_i$ and add $\overline{s_i}$ to the sequence. If neither of these conditions is true then set $w =: s_i w$ and s_i to the sequence.*
4. *If $w = e$ then go to step 8 otherwise reset $i = 1$ and go to step 1.*
5. *If group is of type A_n then stop and return answer that $w \notin \mathcal{I}_{\mathrm{Id}}$. If group is of type B_n call subroutine* I. *If group is of type D_n call subroutine* II.
8. *If $w = e$ then go to step 9 otherwise, reset $i = 1$ and go to step 1.*
9. *This step turns the found sequence into admissible sequence for the original w. If the sequence found is $(r_1, \ldots, r_t)$ then $(r_t, \ldots, r_1)$ is an ascending admissible sequence for w. And w has size t.*

Subroutine I:

6. *If $a_n < 0$ then continue to step 7. If not, then stop and return the answer that $w \notin \mathcal{I}_{\mathrm{Id}}$.*
7. *Find m such that $|a_m| = n$. If $\operatorname{sgn} a_m = -$ then $w =: s_n w s_n$ and add $\overline{s_n}$ to the sequence. If neither of these conditions is true then set $w =: s_n w$ and add s_n to the sequence.*

End Subroutine I.

Subroutine II:

6′. *This step does $s_{\hat{n}}$.* (i) *If $|a_n| > |a_{n-1}|$ and $a_{n-1} < 0$ OR* (ii) *$|a_n| < |a_{n-1}|$ and $a_n < 0$. Continue to step 7′. If neither condition holds then stop and return answer that $w \notin \mathcal{I}_{\mathrm{Id}}$.*
7′. *Find m such that $|a_m| = n-1$. If $\operatorname{sgn} a_m = -$ then $w =: s_{\hat{n}} w s_{\hat{n}}$ and add $\overline{s_{\hat{n}}}$ to the sequence. If neither of these conditions is true then set $w =: s_{\hat{n}} w$ and add $s_{\hat{n}}$ to the sequence.*

End Subroutine II.

In order to find all possible admissible sequences for a given element w we must do a similar computation keeping track of all possibilities. Essentially we are combining the closure algorithm with algorithm 4.

Algorithm 5. *Input an element w and specify Weyl group W. This algorithm will return the poset of admissible sequences for w.*

0. *Let the stack consist of w, where $d_w(w) = 0$, and set $k = 0$.*

I. *For each x such that $d_w(x) = k$ from the stack, do*

A. *For each s_i such that there is no pointer into x labelled s_i or $\overline{s_i}$ (no ascending w-admissible sequence for x begins with s_i or $\overline{s_i}$) do:*

1. *If $i < n$ do 2 and 3. Else go to step 4.*
2. *If $|a_i| > |a_{i+1}|$ and $a_{i+1} > 0$ OR $|a_i| < |a_{i+1}|$ and $a_i < 0$ then go to step 3. If neither of these conditions holds then $i = i+1$ and go continue step* A *(next* (i)*).*
3. (i) *If $i+1$ occurs before i and $i+1$ occurs as a positive entry ($a_l = i+1$ for some l) OR* (ii) *If i occurs before $i+1$ and i occurs as a negative entry. If either one of these is true then set $y =: s_i w s_i$ with $d_w(y) = k+1$ with potential pointer labelled $\overline{s_i}$. If neither of these conditions is true then set $y = s_i w$ with $d_w(y) = k+1$ with potential pointer from labelled s_i.*

4. *If $i = n$ and group is of type A_n then set $i = 1$ and continue with step* I. (*next x*). *If group is of type B_n call subroutine* I. *If group is of type D_n call subroutine* II.
   Subroutine I:
   6. *If $a_n < 0$ then continue to step* 7. *If not, then set $i = 1$ and continue with step* I (*next x*).
   7. *Find m such that $|a_m| = n$. If $\operatorname{sgn} a_m = -$ then $y = s_n x s_n$ with potential pointer labelled $\overline{s_n}$. If neither of these conditions is true then set $y = s_n x$ and potential pointer s_n.*

   End Subroutine I.
   Subroutine II:
   6′. (i) *If $|a_n| > |a_{n-1}|$ and $a_{n-1} < 0$ OR* (ii) *$|a_n| < |a_{n-1}|$ and $a_n < 0$. Continue to step* 7′. *If neither condition holds then set $i = 1$ and continue with step* I (*next x*).
   7′. *Find m such that $|a_m| = n - 1$. If $\operatorname{sgn} a_m = -$ then $w =: s_{\hat{n}} w s_{\hat{n}}$ and add $\overline{s_{\hat{n}}}$ to the sequence. If neither of these conditions is true then set $w =: s_{\hat{n}} w$ and add $s_{\hat{n}}$ to the sequence.*

   End Subroutine II.
8. *For each element z with $d_w(z) = k + 1$, check if $y = z$ if so put a pointer from x to z labelled $s_i, (\overline{s_i})$ (add sequence of y to list of ascending admissible sequences for z). If no such z add y to stack as new element of distance $k+1$, with a pointer from x labelled $s_i, (\overline{s_i})$.*

II. *Let $k = k + 1$ if $y = e$ stop, poset has been completed*; *else go to step* I.

Appendix: Implementation of Main Algorithm

In this section we give the Maple code for the main algorithm, using the Coxeter package. The other algorithms in this paper are variations of this algorithm.

```
# Maple code for the Main Algorithm:
withcoxeter();
with(ListTools);
comparePoints := proc(n::(list), m::(list), R)
# R = a root system data structure
# n & m = lists of integers representing a member of W(R)
description "Determine if two lists of generators end at the same point.";
if reduce([op(n), op(m)],R) = [] then
return true;
else
return false;
end if;
end proc;

findInvolPoset := proc(R)
# 0. Let the stack consist of the identity element of rank 0, and set k = 0
local InvolPoset, id, k, temp, s, si, pt, pts, k1_pt, k1_pts, new_pt, point_unseen, numgen, gen,
bar, source;
```

```
# define constants
numgen := nops(base(R));
id := []; # intialize the identity element
print ("The Involution Poset for", R);
# Establish the generator points
k := 0; # initialize rank to 0
InvolPoset := table(); # initialize InvolPoset to a table
InvolPoset[k] := table(); # initialize InvolPoset[0] to a table
print(base(R));
pt := id;
InvolPoset[k][pt] := table(); # define identity element of rank 0
# I. For* each* x* of* rank* k from the* stack, do
pts := [indices(InvolPoset[k])];
'pts' := FlattenOnce(pts);
###Start Algorithm ###
while nops(pts) <> 0 do
print(nops(FlattenOnce([entries(InvolPoset[k])])),"element(s) of rank", k);
InvolPoset[k+1] := table(); # define next row
for pt in pts do
# A. For each si such that there is no pointer into x labelled si or si-bar
for si from 1 to numgen by 1 do
if not verify(si,FlattenOnce([indices(InvolPoset[k][pt])]), 'member') then
# (no ascending admissible sequence for x begins with si or si-bar) do:
# (i) Calcualte y' = sixsi. If y' != x then let y = y' which is of
# rank k+1. If y' = x then let y = six, which is of rank k+1
source := pt; # set source to current test point
gen := si; # set gen to si
bar := '_'; # report barring
temp := reduce([si,op(pt), si],R); # set temp to sixsi
if comparePoints(temp, pt, R) then # if this is equal to x
temp := reduce([si, op(pt)], R); # set temp to six
bar := ' '; # report unbarring
end if;

new_pt := [op(temp)];
k1_pts := [indices(InvolPoset[k+1])];
'k1_pts' := FlattenOnce(k1_pts);
# (ii) For each element z of rank k+1, check if new_pt=z if so put a
# pointer from x to y labelled si, (si-bar) (add sequence of y to
# list of ascending admissible sequences for z).
# If no such z add y to stack as new element of rank k+1,
# with a pointer from x labelled si, si-bar.
point_unseen := true; # assume new point
for k1_pt in k1_pts do
if comparePoints(new_pt, k1_pt, R) then
point_unseen := false;
new_pt := k1_pt; # set new point to first found shortest path
break; # break out of loop
else
point_unseen := true;
```

```
end if;
end do;
if point_unseen then
InvolPoset[k+1][new_pt] := table(); # define new point as a table
k1_pts := [indices(InvolPoset[k+1])];
'k1_pts' := FlattenOnce(k1_pts);
end if;
InvolPoset[k+1][new_pt][gen] := [source,bar];
# link new point to source via gen
end if;
end do;
end do;

'k' := k+1;
pts := [indices(InvolPoset[k])];
'pts' := FlattenOnce(pts);
end do;

print("The Poset of ", R, " in terms of generators and sources (gen = source (bar or unbar).");
for i from (nops([indices(InvolPoset)])-2) to 0 by -1 do
print("rank", i, " – ", FlattenOnce([eval(InvolPoset[i])]));
end do;
return(copy(InvolPoset));
end proc;
```

References

[1] S. Abeasis, On a remarkable class of subvarieties of a symmetric variety, *Adv. in Math.* **71** (1988) 113–129.

[2] A. Beilinson, J. Bernstein, Localisation de $\mathfrak{g}$-modules, *C.R. Acad. Sci. Paris* **292** (I) (1981) 15–18.

[3] A. Bjorner, F. Brenti, *Combinatorics of Coxeter Groups*, Graduate Texts in Mathematics, 231, Springer-Verlag, Berlin, 2005.

[4] N. Bourbaki, *Groupes et algébres de Lie*, Éléments de Mathématique, ch. Chapitres 4, 5 et 6, Masson, Paris, 1981.

[5] J.-L. Brylinski, P. Delorme, Vecteurs distributions H-invariants pour les séries principales généralisées d'espaces symétriques réductifs et prolongement méromorphe d'intégrales d'Eisenstein, *Invent. Math.* **109** (3) (1992) 619–664.

[6] Chak-On Chow, Counting involutory, unimodal, and alternating signed permutations, *Discrete Mathematics* **306** (18) (2006) 2222–2228.

[7] J.R. Daniel, A.G. Helminck, Algorithms for computations in local symmetric spaces, *Comm. Algebra* **36** (5) (2008) 339–365.

[8] C. De Concini, C. Procesi, Complete symmetric varieties, in: *Invariant Theory* (Montecatini, 1982), Lecture notes in Math., 996, Springer-Verlag, Berlin, 1983, pp. 1–44.

[9] C. De Concini, C. Procesi, Complete symmetric varieties, II, in: *Intersection Theory, Algebraic Groups and Related Topics* (Kyoto/Nagoya, 1983), North-Holland, Amsterdam, 1985, pp. 481–513.

[10] P. Delorme, Formule de Plancherel pour les espaces symétriques réductifs, *Ann. of Math.* (Ser. 2) **147** (2) (1998) 417–452.

[11] M. Flensted-Jensen, Discrete series for semisimple symmetric spaces, *Annals of Math.* **111** (1980) 1758–1788.

[12] D.J. Gagliardi, A.G. Helminck,, Algorithms for computing characters for symmetric spaces, *Acta Appl. Math.* **99** (3) (2007) 339–365.
[13] I. Grojnowski, Character sheaves on symmetric spaces, Ph.D. thesis, Massachusetts Institute of Technology, June 1992.
[14] R. Haas, A.G. Helminck, Computing admissible sequences for twisted involutions in Weyl groups, *Canad. Math. Bull.* **54** (4) (2011) 663–675.
[15] R. Haas, A.G. Helminck, C. Baltera, R. Tramel, Involutions and twisted involution in D_{2n} (in preparation).
[16] R. Haas, A.G. Helminck, C. Cooley, N. Williams, Combinatorial properties of the Richardson-Springer involution poset (in preparation).
[17] R. Haas, A.G. Helminck, N. Rizki, Properties of twisted involutions in signed permutation notation, *J. Combin. Math. Combin. Comput.* **62** (2007) 121–128.
[18] Harish-Chandra, *Collected Papers*, Vol. I–IV (1944–1983), Edited by V.S. Varadarajan, Springer-Verlag, New York, 1984.
[19] A.G. Helminck, Computing B-orbits on G/H, *J. Symbolic Computation* **21** (1996) 169–209.
[20] A.G. Helminck, Computing orbits of minimal parabolic k-subgroups acting on symmetric k-varieties, *J. Symbolic Computation* **30** (5) (2000) 521–553.
[21] A.G. Helminck, S.P. Wang, On rationality properties of involutions of reductive groups, *Adv. in Math.* **99** (1993) 26–96.
[22] F. Hirzebruch, P.J. Slodowy, Elliptic genera, involutions, and homogeneous spin manifolds, *Geom. Dedicata* **35** (1-3) (1990) 309–343.
[23] G. Lusztig, Symmetric spaces over a finite field, in: *The Grothendieck Festschrift* (Boston, MA), Vol. III, Progr. Math., 88, Birkhäuser, Berlin, 1990, pp. 57–81.
[24] G. Lusztig, D.A. Vogan, Singularities of closures of K-orbits on flag manifolds, *Invent. Math.* **71** (1983) 365–379.
[25] T. Matsuki, The orbits of affine symmetric spaces under the action of minimal parabolic subgroups, *J. Math. Soc. Japan* **31** (1979) 331–357.
[26] T. Ōshima, T. Matsuki, A description of discrete series for semisimple symmetric spaces, in: *Group Representations and Systems of Differential Equations* (Tokyo, 1982), North-Holland, Amsterdam, 1984, pp. 331–390.
[27] T. Ōshima, J. Sekiguchi, Eigenspaces of invariant differential operators in an affine symmetric space, *Invent. Math.* **57** (1980) 1–81.
[28] R.W. Richardson, T.A. Springer, The Bruhat order on symmetric varieties, *Geom. Dedicata* **35** (1-3) (1990) 389–436.
[29] W. Rossmann, The structure of semisimple symmetric spaces, *Canad. J. Math.* **31** (1979) 157–180.
[30] T.A. Springer, Some results on algebraic groups with involutions, in: *Algebraic Groups and Related Topics* (Kyoto/Nagoya, 1983), North-Holland, Amsterdam, 1985, pp. 525–543.
[31] Y.L. Tong, S.P. Wang, Geometric realization of discrete series for semisimple symmetric space, *Invent. Math.* **96** (1989) 425–458.
[32] E.P. van den Ban, H. Schlichtkrull, The most continuous part of the Plancherel decomposition for a reductive symmetric space, *Ann. of Math.* (Ser. 2) **145** (2) (1997) 267–364.
[33] D.A. Vogan, Irreducible characters of semi-simple Lie groups IV, Character-multiplicity duality, *Duke Math. J.* **49** (1982) 943–1073.
[34] D.A. Vogan, Irreducible characters of semi-simple Lie groups III, *Invent. Math.* **71** (1983) 381–417.