Implementing an algorithm for the twisted involution poset for Weyl groups

Kathryn Brenneman *
Department of Mathematics, Smith College
Northampton, MA 01063

Ruth Haas
Department of Mathematics, Smith College,
Northampton, MA 01063, rhaas@math.smith.edu

Aloysius G. Helminck †
Department of Mathematics, North Carolina State University,
Raleigh, N.C., 27695-8205, loek@unity.ncsu.edu

May 13, 2006

Abstract

In algebraic contexts Weyl group elements are usually represented in terms of generators and relations, where representation is not unique. For computational purposes, a more combinatorial representation for elements of the classical Weyl Group of type A_{n-1} as permutations can be useful. In this paper we describe and contrast two implementations of an algorithm to to compute a poset of the twisted involutions in a Weyl Group. The first uses generators and relations and the second a method that gives unique representation.

1 Introduction

Weyl groups are defined as reflection groups of root systems. That is, given a root system Φ in a Euclidian vector space $V = \mathbb{R}^n$, for each vector $\alpha \in \Phi$, let s_α be the reflection through α. The Weyl group is the group generated by these reflections. Weyl groups and several of their subsets play an important role in combinatorics, representation theory, Lie theory

*Supported by Schultz Fund for undergraduate summer research at Smith College
†Partially supported by N.S.F. Grant DMS-0532140

and geometry. For θ an involution of a Weyl group W, the set $\mathcal{I}_\theta = \{w \in W \mid \theta(w) = w^{-1}\}$ is called *the set of θ-twisted involutions in W.* In this paper we describe and contrast two implementations of an algorithm to to compute a natural poset of the twisted involutions in a Weyl Group.

The set of twisted involutions is important in the study of orbits of minimal parabolic k-subgroups acting on symmetric k-varieties, see [5, 6]. The geometry of these orbits and their closures induce a poset structure on the set $\mathcal{I}_\theta$. Understanding this poset structure is key to understanding the structure of the orbits. For $\theta = \mathrm{id}$, the identity map we get the set $\mathcal{I}_{\mathrm{id}} = \{w \in W \mid w = w^{-1}\}$ the set of involutions in W. 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 combinatorialists and has an interesting geometrical structure. There are many open questions about this poset of twisted involutions. Having an algorithm to compute this poset and its structure will be essential for further study of this poset.

In [3] and [4] we developed the theory and gave a set of algorithms to compute this poset and several variations including finding orbit closures. In this paper we discuss two implementations of the main algorithm. The first makes use of the existing structure of John Stembridge's Coxeter Package for Weyl Groups. The second utilizes a more combinatorial representation for Weyl Group elements and is implemented directly in Maple.

In section 2 we the give background and recall the algorithm for twisted involutions. In section 3 special properties using bottom row notation are discussed and we give variations of the algorithms which take advantage of these properties. The final section describes our experience with implementing the algorithms. We thank our undergraduate student Kathryn Brenneman for her excellent assistance in implementation.

2 Background and Algorithms

In the following 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 [3] 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 $\overline{w}a = waw(w)^{-1}$ 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 $\overline{\ }$ (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. Such a minimal representation is called *reduced*. 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 $0 = l(a_0) < l(a_1) < \ldots < l(a_k)$. Richardson and Springer [7] 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, i.e., have $l(w)$ elements. 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. In fact, ([7]) all ascending admissible sequences for a given element will have the same number of elements. So we may define the *size* of an element $w \in \mathcal{I}_\theta$ to be 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 size of w_0 by k_0.

2.1 Main Algorithm

In [4] we developed an algorithm for computing the twisted (and non-twisted) involutions. The algorithms generates the $\mathcal{I}_\theta$ and $\mathcal{I}_{\mathrm{id}}$ posets discussed in the introduction.

In this paper we will only consider the Weyl group W of type A_{n-1}. It is well known that this Weyl group is isomorphic to the symmetric group S_n. The standard generators for Weyl groups correspond to reflections. The

Weyl group of type A_{n-1} is generated by the transpositions $s_1 = (1,2), s_2 = (2,3), s_3 = (3,4), \ldots, s_{n-1} = (n-1,n)$. The algorithm is based on the following lemmas:

Lemma 2.2. *Given a Weyl group W, $w \in W$, an involution θ and a generator s_i.*

1. *If $w \in \mathcal{I}_{id}$ then $s_i w s_i \in \mathcal{I}_{id}$.*
2. *If $w \in \mathcal{I}_\theta$ then $s_i w \theta(s_i) \in \mathcal{I}_\theta$.*

Lemma 2.3. *Given a Weyl group W, $w \in W$, an involution θ and a generator s_i.*

1. *If $w \in \mathcal{I}_{id}$ and $s_i w s_i = w$, then $s_i w \in \mathcal{I}_{id}$*
2. *If $w \in \mathcal{I}_\theta$ and $s_i w \theta(s_i) = w$, then $s_i w \in \mathcal{I}_\theta$*

We first give the algorithm to compute $\mathcal{I}_{\mathrm{Id}}$. The operation of Lemma 2.2 will be denoted by $\overline{s_i}$. For each element w in $\mathcal{I}_{\mathrm{Id}}$ we keep track of the following: (i) its size, (ii) pointers from elements x of size 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 labeled pointers from the identity element to w.

Algorithm 2.4. *Input Weyl group. Output poset for $\mathcal{I}_{\mathrm{Id}}$.*

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

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

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

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 size of w_0 the longest element in the Weyl group, which is always an element of $\mathcal{I}_\theta$ as well. Even without knowing the explicit size 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 labeled s or $\overline{s}$ (see [3]).

2.5 The twisted case.

A direct modification of Algorithm 2.4 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 size $k+1$. If $y' = x$ then let $y = s_i x$, which is of size $k+1$.

Figure 1 gives the poset of the elements of $\mathcal{I}_\theta$ for the Weyl group of type A_4. The elements here are listed using *bottom row* notation. This

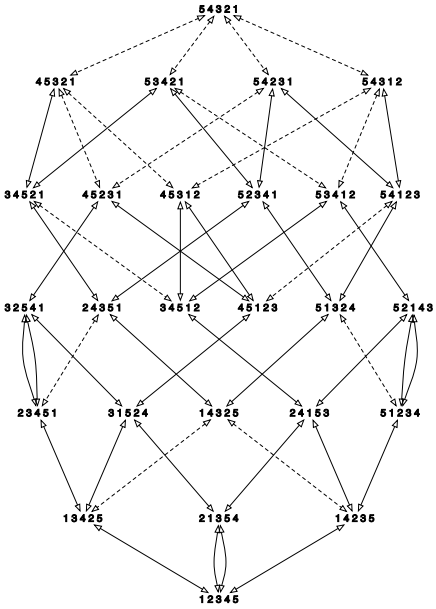

Figure 1: Poset of $\mathcal{I}_\theta$ for type A_4

is a modification of a standard unique representation for permutations. Specifically, the permutation

$$\begin{pmatrix} 1 & 2 & \dots & n \\ a_1 & a_2 & \dots & a_n \end{pmatrix},$$

can be abbreviated by just listing the bottom row. From each element there is an edge for each group generator. The edges will correspond to one of two possible operations. The lemmas 2.2 and 2.3 give the two possibilities. In Figure 1, the operation in Lemma 2.2 is designated by a solid line while the operation in Lemma 2.3 is designated by a dashed line.

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 the case of Algorithm 2.4 step (ii) 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 2.4 is less than that in the variation for $\mathcal{I}_\theta$. Nonetheless, in both variations the number of comparisons is very large. This lead us to suspect that implementation using such a package would be less efficient than using a system with unique representation. In the next section we discuss an variation of the above algorithms, using unique representation for the Weyl group elements.

3 Properties of elements in bottom row notation

Besides being a convenient notation for diagrams, bottom row notations provides a concise unique representation system for the elements of S_n. Given the high number times our algorithm must check if two elements are equal, we speculated that describing the elements with a unique representation would prove much more computationally efficient. Thus we rewrote the algorithm to take advantage of bottom row notation.

The following proposition describes multiplication by generators in bottom row notation.

Proposition 3.1. *For* $1 \leq i < n$,

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

(ii) If $a_k = i$, *and* $a_l = i + 1$ *then* $s_i(a_1, a_2, \ldots, a_n) = (a_1, a_2, \ldots, a_{k-1}, a_l, a_{k+1}, \ldots, a_{l-1}, a_k, a_{l+1}, \ldots, a_n)$,

The next results characterizes when the operations of lemmas 2.2 and 2.3 are used.

Theorem 3.2. *Let* W *be a Weyl group of type* A_{n-1}, s_i *a generator and* $w = (a_1, \ldots, a_n) \in W$.

1. $s_i w s_i = w$ *if and only if* w *satisfies* $\{a_i, a_{i+1}\} = \{i, i + 1\}$.

2. $s_i w \theta(s_i) = w$ *if and only if* w *satisfies* $\{a_{(n-i)}, a_{(n-i+1)}\} = \{i, i + 1\}$.

Proof. (1) From Proposition 3.1 if $i < n$ then $s_i w$ acts on w by switching the elements which are equal to i and $(i + 1)$, while ws_i switches the elements in the ith and $(i+1)$st positions. Hence $s_i w s_i = w$ if and only if $\{a_i, a_{i+1}\} = \{i, i + 1\}$.

(2) Observe that $\theta(s_i) = s_{n-i}$. As before from Proposition 3.1 it follows that $s_i w$ acts on w by switching the elements which are equal to i and $(i+1)$, while $w\theta(s_i) = ws_{n-i}$ switches the elements in the $(n-i)$th and $(n-i+1)$st positions. Hence $s_i w\theta(s_i) = w$ if and only if $\{a_{(n-i)}, a_{(n-i+1)}\} = \{i, i+1\}$. □

While the above theorem holds for all $w \in W$, we are in fact most interested in the case when $w \in \mathcal{I}_{\mathrm{id}}$, or $w \in \mathcal{I}_\theta$. The algorithm now can be modified to remove the computation in step I.A.i. The only modification made to algorithm 2.4 is to change step I.A.i to:

(i) If $\{a_i, a_{i+1}\} \neq \{i, i+1\}$, then let $y = y'$ which is of size $k+1$. If $\{a_i, a_{i+1}\} = \{i, i+1\}$ then let $y = s_i x$, which is of size $k+1$.

Similarly, for the twisted case the only modification required is to change to change step I.A.i' in the algorithm from section 2.5. It becomes:

(i') If $\{a_{(n-i)}, a_{(n-i+1)}\} \neq \{i, i+1\}$, then let $y = y'$ which is of size $k+1$. If $\{a_{(n-i)}, a_{(n-i+1)}\} = \{i, i+1\}$ then let $y = s_i x$, which is of size $k+1$.

3.3 Implementation

The algorithms were implemented on a double processor 2.7 GHz Macintosh G5 with 8GB of RAM. The original algorithm, 2.4, and its variation for twisted involutions, 2.5, were implemented in the Coxeter Package in Maple. Using this were only able to compute the poset for $\mathcal{I}_{\mathrm{id}}$ up to type A_7, which took approximately 6 hours. The computer crashed after 3 days of working on A_8.

On the other hand the implementation of the algorithms of section 3 utilizing bottom row notation led to an almost exponential increase in efficiency. With these modified algorithms finding $\mathcal{I}_{\mathrm{id}}$ for the Weyl group of type A_7 took 3.08 seconds and in the Coxeter Maple implementation it took 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. Both implementations are in Maple, so the main difference is the unique representation of the Weyl group elements.

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 [1] and Gagliardi [2].

References

[1] J. R. Daniel and A. G. Helminck, *Algorithms for computations in local symmetric spaces*, (2004), To appear.

[2] D.J. Gagliardi and A. G. Helminck, *Algorithms for weight lattices of symmetric spaces*, (2005), To appear.

[3] R. Haas and A. G. Helminck, *Computing admissible sequences for twisted involutions in Weyl groups*, (2004), To Appear.

[4] _____, *Algorithms for twisted involutions in Weyl groups*, (2005), To Appear.

[5] A. G. Helminck, *Computing B-orbits on G/H*, J. Symbolic Computation **21** (1996), 169–209.

[6] _____, *Computing orbits of minimal parabolic k-subgroups acting on symmetric k-varieties*, J. Symbolic Comput. **30** (2000), no. 5, 521–553.

[7] R. W. Richardson and T. A. Springer, *The Bruhat order on symmetric varieties*, Geom. Dedicata **35** (1990), no. 1-3, 389–436.