Weyl Group Orbits

DENNIS M. SNOW
University of Notre Dame

A new technique is presented for calculating the orbits of the finite Weyl group of a semisimple Lie group G in the weight lattice of G. Such calculations are important in the representation theory of G, and have previously been difficult to carry out for large Weyl groups such as E_8. This new technique allows large orbits to be computed using only a small fraction of the computer memory required when using standard techniques. In the case of E_8, the memory requirements can be reduced by a factor of 30,000.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problem—*computations on discrete structures*; I.1.2 [**Algebraic Manipulation**]: Algorithms—*algebraic algorithms, analysis of algorithms*; G.4 [**Mathematics of Computing**]: Mathematical Software—*efficiency*

General Terms: Algorithms, Theory, Performance

Additional Key Words and Phrases: Lie groups, representation theory, Weyl group

Many formulas in the representation theory of a simple complex Lie group, G, involve an associated finite group, W, the Weyl group of G. The most celebrated of these is the formula of Weyl [6], which expresses the character of a representation as the quotient of two alternating sums, both indexed by W-orbits in the weight lattice of G. In Bott's Theorem [2] the level of a weight, that is, the number of reflections it takes to move the weight to the dominant chamber, identifies the degree in which the cohomology group of an irreducible homogeneous vector bundle does not vanish. In some applications, only certain subsets of W, consisting of "distinguished coset representatives," are needed, and these too can be found by calculating the W-orbit of an appropriately chosen weight. A difficulty one often encounters in explicit calculation with W is the large size of its orbits. The number of elements in W greatly exceeds the order of magnitude of other parameters of the group. For example, the exceptional group E_8 has dimension 248 and 120 positive roots, but its Weyl group has 696,729,600 elements.

A straightforward way to compute an orbit of W is to start with an appropriate dominant weight and reflect it by all the simple reflections. This produces a list of weights of level 1. If the weight were regular, this list would correspond to the

This work was supported by NSF grant DMS 8420315.
Author's address: Department of Mathematics, University of Notre Dame, Notre Dame, IN 46556.

list of all elements in W of length 1, the simple reflections themselves. Now reflect each of these weights by each simple reflection again to create a new list of weights. Disregarding repetitions and weights of level 1—which we already know—we now have a list of all weights in the orbit of level 2. For a regular weight this would correspond to the list of elements in W of length 2. By repeating this procedure, we can obtain a list of weights of any level using only the weights of the next lower level and the simple reflections. Thus, the entire W-orbit, or an image of the whole group if the original weight were regular, can be generated this way. Now we see how the size of W can create a problem for explicit calculation. If we use this technique to generate the Weyl group, W, of E_8, the lists which need to be remembered at any stage will eventually contain more than 18 million entries, and such lists can easily outstrip available computer memory. Another problem with this technique is the number of repetitions that occur while generating the next list. Simply searching the list being created for repetitions for each newly created orbit element is extremely slow and inefficient.

I present here a simple technique for deciding whether or not an orbit element should be added to the list of weights of next highest level which requires only "local" data, that is, the new weight itself and the simple reflection that created it. This solves the problem of repetitions and also enables a new technique for generating Weyl group orbits. Instead of proceeding one level at a time, finding all the weights of a given level, we may think of the orbit as a tree and compute "depth first," following the orbit of a weight as far as possible according to the criterion for saving a weight before returning to uncomputed "branches." This latter technique can be managed easily with a stack that has an a priori bound on the number of entries, namely, the number of positive roots in G. This is a significant reduction in memory requirements. While it might not be easy to store a large orbit permanently, this technique, due to its small memory requirements, could be used to calculate on the spot where it is needed in other formulas.

An implementation of this algorithm is given in Section 4. The procedures are written in C and take advantage of the two-fold symmetry of Weyl groups. For testing purposes, the Weyl groups of $E6$, $E7$, and $E8$ were generated and the number of elements of each length was counted, see Table I. Example times on a Sun 3/280 were as follows: 2.0 seconds for $E6$ which has only 51,840 elements; 117.9 seconds for $E7$ which has 2,903,040 elements; and 31611.0 seconds (8.8 hours) for $E8$ which has the above-mentioned 696,729,600 elements. Average performance was thus somewhere between 22,000 to 25,000 elements processed per second.

1. PRELIMINARIES

The following facts are well known and can be found in [4] or [5]. We recall them here to establish notation and terminology. Let G be a simple complex Lie group and let $\mathscr{G}$ be its Lie algebra. Let $\mathscr{H}$ be a Cartan subalgebra of $\mathscr{G}$ (a maximal Abelian subalgebra). The *roots* of $\mathscr{G}$ are defined to be the nonzero Eigenvalues of $\mathscr{H}$ acting on $\mathscr{G}$ via the adjoint representation. The roots are viewed as linear functionals on $\mathscr{H}$ so that if α is a root and $x \in \mathscr{H}$ is a corresponding eigenvector, then $[h, x] = \alpha(h)x$ for all $h \in \mathscr{H}$. The set of roots of $\mathscr{G}$ will be denoted by

Table I. Levels in the Weyl group orbit of δ (times are in seconds)

$E6$		$E7$		$E8$			
level	size	level	size	level	size	level	size
0, 36	1	0, 63	1	0, 120	1	32, 88	3319268
1, 35	6	1, 62	7	1, 119	8	33, 87	3780640
2, 34	20	2, 61	27	2, 118	35	34, 86	4278429
3, 33	50	3, 60	77	3, 117	112	35, 85	4811752
4, 32	105	4, 59	182	4, 116	294	36, 84	5379194
5, 31	195	5, 58	378	5, 115	672	37, 83	5978792
6, 30	329	6, 57	713	6, 114	1386	38, 82	6608029
7, 29	514	7, 56	1247	7, 113	2640	39, 81	7263840
8, 28	754	8, 55	2051	8, 112	4718	40, 80	7942628
9, 27	1048	9, 54	3205	9, 111	8000	41, 79	8640288
10, 26	1389	10, 53	4795	10, 110	12978	42, 78	9352240
11, 25	1765	11, 52	6909	11, 109	20272	43, 77	10073472
12, 24	2159	12, 51	9632	12, 108	30645	44, 76	10798593
13, 23	2549	13, 50	13040	13, 107	45016	45, 75	11521896
14, 22	2911	14, 49	17194	14, 106	64470	46, 74	12237428
15, 21	3222	15, 48	22134	15, 105	90264	47, 73	12939064
16, 20	3461	16, 47	27874	16, 104	123829	48, 72	13620586
17, 19	3611	17, 46	34398	17, 103	166768	49, 71	14275768
18	3662	18, 45	41657	18, 102	220849	50, 70	14898464
		19, 44	49567	19, 101	287992	51, 69	15482696
		20, 43	58009	20, 100	370250	52, 68	16022740
		21, 42	66831	21, 99	469784	53, 67	16513208
		22, 41	75852	22, 98	588833	54, 66	16949127
		23, 40	84868	23, 97	729680	55, 65	17326016
		24, 39	93659	24, 96	894613	56, 64	17639957
		25, 38	101997	25, 95	1085880	57, 63	17887656
		26, 37	109655	26, 94	1305640	58, 62	18066494
		27, 36	116417	27, 93	1555912	59, 61	18174568
		28, 35	122087	28, 92	1838523	60	18210722
		29, 34	126497	29, 91	2155056		
		30, 33	129514	30, 90	2506798		
		31, 32	131046	31, 89	2894688		
total	51840	total	2903040			total	696729600
time	2.0	time	117.9			time	31611.0

Φ. They span a *real* subspace, E, in the dual space $\mathscr{H}^*$ of real dimension $l = \dim_{\mathbb{C}}\mathscr{H} = \text{rank } \mathscr{G}$. There always exists a special set of roots, called a *base*, $\Delta = \{\alpha_1, \ldots, \alpha_l\}$, which forms a basis for E and such that any other root $\alpha \in \Phi$ can be written as a linear combination $\alpha = \sum_{i=1}^{l} n_i \alpha_i$ where the n_i are either *all* positive integers, in which case we say that α is a positive root, or *all* negative integers, in which case we say that α is a negative root. The subset of positive, respectively negative, roots is denoted by Φ^+, respectively Φ^-. The elements of a base Δ are also called *simple* roots.

There is a natural inner product on $\mathscr{G}$, called the Killing form, defined by $(x, y) := \mathrm{Tr}(ad(x)ad(y))$ where $ad: \mathscr{G} \to gl(\mathscr{G})$, $ad(x)(v) := [x, v]$, is the adjoint representation. This form takes real values when restricted to E and is positive definite there. One can show that for all pairs of roots $\alpha, \beta \in \Phi$,

$$\langle \beta, \alpha \rangle := \frac{2(\beta, \alpha)}{(\alpha, \alpha)} \in \mathbf{Z}.$$

Moreover, Φ is invariant under the reflection through a hyperplane orthogonal to any given root. Explicitly, given a root $\alpha \in \Phi$, the reflection defined by

$$\sigma_\alpha(x) := x - \langle x, \alpha \rangle \alpha$$

is an orthogonal linear transformation with respect to the Killing form such that for all $\beta \in \Phi$ we have that $\sigma_\alpha(\beta) \in \Phi$. These two properties are the essential tools in the classification of simple Lie algebras (and groups).

The finite group W generated by all reflections, σ_α, $\alpha \in \Phi$, is called the *Weyl group* of $\mathscr{G}$ (or G). A reflection associated to a simple root is called a *simple reflection.* Any element of the Weyl group, $\sigma \in W$, can be written (not necessarily uniquely) as a product of simple reflections $\sigma = \sigma_{i_1}\sigma_{i_2} \cdots \sigma_{i_k}$. The minimum number of simple reflections in an expression for σ is called the *length* of σ and is denoted by $l(\sigma)$. It can be shown that the length of σ is equal to the number of positive roots which are sent to negative roots under σ. For example, a simple reflection σ_i has length 1; it sends the simple root α_i to $-\alpha_i$ and permutes the other positive roots. From this, it follows that

$$l(\sigma_i \sigma) = \begin{cases} l(\sigma) + 1, & \text{if } \sigma^{-1}(\alpha_i) \in \Phi^+ \\ l(\sigma) - 1, & \text{if } \sigma^{-1}(\alpha_i) \in \Phi^- \end{cases} \tag{1.1}$$

Let P_α, $\alpha \in \Phi$, be the hyperplane $\{x \in E \mid (\alpha, x) = 0\}$. The connected components of $E - \bigcup_{\alpha \in \Phi} P_\alpha$ are finite in number and are called the (open) *Weyl chambers* of E. We say $\xi \in E$ is regular if ξ is in one of these open Weyl chambers, that is, $(\alpha, \xi) \neq 0$ for all $\alpha \in \Phi$. If $(\alpha, \xi) = 0$ for some root α, we say that ξ is *singular.* The *fundamental Weyl chamber* is the unique chamber, $\mathscr{C}$, satisfying $\xi \in \mathscr{C} \Rightarrow (\alpha, \xi) > 0$ for all $\alpha \in \Phi$ (or for all $\alpha \in \Delta$). The Weyl group acts simply transitively on the Weyl chambers and the closure of the fundamental chamber, $\bar{\mathscr{C}}$ is a fundamental domain for the action of W on E. Thus, every $\xi \in E$ is conjugate to a unique point in $v \in \bar{\mathscr{C}}$. In this context, the word "conjugate" means "in the same Weyl group orbit." The *level* of ξ is the minimum length of a $\sigma \in W$ such that $\xi = \sigma(v)$. It is not hard to show that the level of ξ can also be defined as the number of positive roots α such that $(\alpha, \xi) < 0$, or equivalently, as the number of hyperplanes, P_α, crossed by a straight line from ξ to a general point in $\mathscr{C}$.

A *weight* is an element $\lambda \in E$ such that $\langle \lambda, \alpha \rangle \in \mathbf{Z}$ for all $\alpha \in \Phi$. The set of weights Λ forms a subgroup of E containing the set of roots Φ. The weights are the Eigenvalues of the Cartan subalgebra $\mathscr{H}$ which occur in finite dimensional representations of the Lie algebra $\mathscr{G}$. If $\Delta = \{\alpha_1, \ldots, \alpha_l\}$, then the vectors $2\alpha_i/(\alpha_i, \alpha_i)$ again form a basis of E. Let $\lambda_1, \ldots, \lambda_l$ be the associated dual basis of E relative to the Killing form. Thus, $2(\lambda_i, \alpha_j)/(\alpha_j, \alpha_j) = \delta_{ij}$, and the λ_i are themselves weights, called the *fundamental dominant weights*. Any $x \in E$ can be written as a linear combination $\sum_{i=1}^{l} m_i \lambda_i$ where the coefficients are given by $m_i = \langle x, \alpha_i \rangle$. If $\lambda \in \Lambda$, then λ is an *integral* linear combination of the λ_i, $\lambda = \sum_{i=1}^{l} \langle \lambda, \alpha_i \rangle \lambda_i$. Therefore, Λ is a lattice with basis $\lambda_1, \ldots, \lambda_l$. A weight λ is *dominant* if its coefficients are nonnegative, $\langle \lambda, \alpha_i \rangle \geq 0$, $i = 1, \ldots, l$. The set of dominant weights is denoted by Λ^+. Notice that $\Lambda^+ = \Lambda \cap \overline{\mathscr{C}}$, so that any weight is conjugate to a unique dominant weight.

We are interested in calculating the action of the Weyl group W on the weights Λ. For a fixed base $\Delta = \{\alpha_1, \ldots, \alpha_l\}$ the set of fundamental dominant weights, $\lambda_1, \ldots, \lambda_l$, is the best basis in which to perform these calculations, so we simply write $(m_1, \ldots, m_l)$ for the weight $\sum_{i=1}^{l} m_i \lambda_i$. The action of a simple reflection $\sigma_i = \sigma_{\alpha_i}$ on a weight $\mu = (m_1, \ldots, m_l)$ is given by

$$\sigma_i(\mu) = \mu - \langle \mu, \alpha_i \rangle \alpha_i = (m_1 - m_i c_{i1}, \ldots, m_l - m_i c_{il}) \tag{1.2}$$

where $(c_{i1}, \ldots, c_{il})$ is the vector expression for the root α_i. Since $c_{ij} = \langle \alpha_i, \alpha_j \rangle$, this vector is just the ith row of the Cartan matrix $(\langle \alpha_i, \alpha_j \rangle)_{i,j=1}^{l}$. The action of a general element $\sigma \in W$ can be found by expressing σ as a product of simple reflections and applying the above formula.

2. COMPUTING ORBITS BY LEVEL

Let us describe in more detail the procedure mentioned in the introduction for generating a Weyl group orbit. We need, first of all, a simple way to determine the level of $\sigma_i(\xi)$, for a simple reflection σ_i and a given point $\xi = (x_1, \ldots, x_l) \in E$. Let $\sigma \in W$ be such that $\xi = \sigma(v)$ for some $v \in \overline{\mathscr{C}}$ with level $(\xi) = l(\sigma)$. If $x_i = 0$, then $\sigma_i(\xi) = \xi$ and level $(\sigma_i(\xi)) = \text{level}(\xi)$. If $x_i > 0$, then $(\alpha_i, \xi) > 0$, and therefore $(\sigma^{-1}(\alpha_i), v) > 0$, since the Killing form is invariant under W. Now, v is dominant, so it must be that $\sigma^{-1}(\alpha_i)$ is a positive root, and this implies that $l(\sigma_i \sigma) = l(\sigma) + 1$, see (1.1). Therefore, level $(\sigma_i(\xi)) = \text{level}(\xi) + 1$. A similar argument shows that if $x_i < 0$, then level $(\sigma_i(\xi)) = \text{level}(\xi) - 1$. To summarize: If $\xi = (x_1, \ldots, x_l) \in E$, then

$$\text{level}(\sigma_i(\xi)) = \begin{cases} \text{level}(\xi) + 1 & \text{if } x_i > 0, \\ \text{level}(\xi) & \text{if } x_i = 0, \\ \text{level}(\xi) - 1 & \text{if } x_i < 0. \end{cases} \tag{2.1}$$

Now, suppose we are given a dominant weight $\mu \in \Lambda^+$. The weights in the orbit $W.\mu = \{\sigma.\mu \mid \sigma \in W\}$ can be organized by level. Let L_k denote the kth level of $W.\mu$, that is, the set of weights in $W.\mu$ of level k. Then $W.\mu$ is the disjoint union of $L_0, \ldots, L_N$ where N is the maximum possible level in $W.\mu$. Note that N is bounded by the number of positive roots in $\mathscr{G}$, since this is the maximum length of a Weyl group element. Level L_0 consists only of μ, $L_0 = \{\mu\}$. To create level L_1, we reflect μ by all the simple reflections. By (2.1) we need only reflect by the simple reflection σ_i if the ith coordinate of μ is positive. The procedure is similar for inductively generating level L_{k+1} from the previously computed level L_k. Obviously, every element of L_{k+1} will be of the form $\sigma_i(\nu)$ for some simple reflection σ_i and some weight $\nu \in L_k$. Therefore, by (2.1):

$$L_{k+1} = \{\sigma_i(\nu) \mid i = 1, \ldots, l, \nu = (n_1, \ldots, n_l) \in L_k, n_i > 0\}.$$

This formula shows how simple it is to decide whether a weight in level L_k should be reflected by a given simple reflection to obtain a weight in level L_{k+1}. On the other hand, the formula hides a real problem in actually computing a W-orbit this way, namely, how to avoid repetitions during the calculation of level

L_{k+1}. Since there are many occurrences of $\sigma_i(\nu_1) = \sigma_j(\nu_2)$ with $\nu_1, \nu_2 \in L_k$, and the level L_k can be very large, simply searching the level being created for every weight reflected from L_k to L_{k+1} is not practical. Hash tables would speed up this task, but for very large Weyl groups the process is still too inefficient. By taking advantage of a natural ordering of the weights, it is actually quite easy to avoid repetitions. We shall now prove that the decision to save a weight $\sigma_i(\nu)$ depends only on its coordinates and the index i.

THEOREM 2.1 *Let L_k be the kth level in the orbit $W.\mu$ of a dominant weight $\mu \in \bar{\mathscr{C}}$. Then, for each $\xi = (x_1, \ldots, x_l) \in L_{k+1}$, there exists a unique $\nu \in L_k$ and a unique simple reflection σ_i such that $\sigma_i(\nu) = \xi$ and $x_j \geq 0$ for $j > i$. In particular, the next level L_{k+1} can be constructed without repetitions from the weights ν in L_k by adding $\sigma_i(\nu)$ to L_{k+1} if and only if the ith coordinate of ν is positive and the coordinates of $\sigma_i(\nu)$ after the ith are nonnegative*:

$$L_{k+1} = \{\sigma_i(\nu) = (x_1, \ldots, x_l) \mid i = 1, \ldots, l, \\ \nu = (n_1, \ldots, n_l) \in L_k,\ n_i > 0,\ x_j \geq 0,\ j > i\}. \tag{2.2}$$

PROOF. Let $\xi = (x_1, \ldots, x_l)$ be an arbitrary element of L_{k+1}. Let i be the index of the last *negative* coordinate of ξ, that is, $x_i < 0$ and $x_j \geq 0$ for $j > i$. Let $\nu := \sigma_i(\xi) = (n_1, \ldots, n_l)$, so that $\sigma_i(\nu) = \xi$. By (1.2), $n_i = x_i - x_i c_{ii} = -x_i > 0$, and by (2.1), $\nu \in L_k$. This proves the equality (2.2). Now, suppose $\sigma_m(\omega) = \xi$ for some $\omega \in L_k$ other than ν. By (2.1), the mth coordinate of ω must be positive, so that $x_m < 0$. Hence $m < i$ by the choice of i ($i \neq m$ because $\omega \neq \nu$). Since $x_i < 0$, this proves that the representation of ξ as $\sigma_i(\nu)$ with $\nu \in L_k$ and $x_j \geq 0$ for $j > i$ is unique. □

3. MEMORY EFFICIENT GENERATION OF ORBITS

Now that we have discussed how to generate a Weyl group orbit with minimal computation, let us address the issue of memory requirements. If an orbit is computed by level, as outlined in the previous section, then an entire level must be stored somewhere in order to generate the next level. We have already mentioned that even for the group E_8, this means saving over 18 million 8-dimensional vectors at a time. For reasons of speed, it is most natural to want to keep these vectors in random access memory. However, in many systems today, this amount of memory is not available. Permanent storage devices of the required capacity are more readily available, but retrieval of this data slows the computations greatly. One may not even be interested in saving the weights generated, but only in using them for other computations.

These concerns about memory requirements can be circumvented by reorganizing the way an orbit is generated. Not only does Theorem 2.1 provide an efficient way to compute one level from another, it also shows that the decision to "save" a newly generated weight $\sigma_i(\nu)$ can be made solely on the basis of the coordinates of $\sigma_i(\nu)$ and the index i. This fact can be exploited in such a way that one need never remember more weights at any one time than the number of positive roots in the group, a significant reduction from the number of weights in the largest level. Let us now describe this new algorithm.

We may think of a Weyl group orbit $W.\mu$ as a directed graph whose nodes are the weights. Two weight-nodes ν_1, ν_2 are connected by an edge if there is a simple reflection $\sigma_i \in W$ such that $\sigma_i(\nu_1) = \nu_2$. The direction of that edge is from the weight-node at the lower level to the weight-node at the higher level. For a given weight $\nu_2 \in L_{k+1}$, there are usually many "predecessors" $\nu_1 \in L_k$ such that $\sigma_i(\nu_1) = \sigma_i(\nu_2)$. Theorem 2.1 shows, however, that there is a systematic way of finding a predecessor. If we include in our graph only those edges between the weight-nodes that correspond to these uniquely determined predecessors, then our graph becomes a *tree*. A weight-node can have several successors, but only one predecessor. The "root" of the tree, or starting weight-node, is the dominant weight in the orbit $\mu \in \overline{\mathscr{C}}$. The "depth" of a particular weight-node is the level of that weight; the collection of nodes at a particular depth forms a level, as we have defined it above. With this structure in mind, we can now generate the orbit $W.\mu$ in the same way that a tree is commonly searched, that is, "depth first." Let us define a recursive algorithm, called *Branch*(ν), which takes a weight-node $\nu = (n_1, \ldots, n_l) \in L_k$ and generates the branch of the tree which proceeds from the weight-node ν.

Branch(ν):

1. Set $i_0 = 0$.
2. Let i be the first index such that $i > i_0$ and $n_i > 0$. If there is no such index i, then stop.
3. Compute the reflected weight $\xi = \sigma_i(\nu) = (x_1, \ldots, x_l)$.
4. If $x_{i+1}, \ldots, x_l \geq 0$, then output ξ and execute *Branch*(ξ).
5. Set $i_0 = i$ and go to step 2.

Branch(μ) would report all the weight-nodes in the tree (all the weights in the orbit $W.\mu$). The only operations needed in *Branch* are to compare integers and to compute the reflection in step 3. The latter step can be streamlined into performing no more than three additions and a sign change, as we shall see in the next section. Clearly, in step 4, many other items of interest may also be reported along with ξ while the necessary information is readily at hand, for example, the number of weights computed up to that point, the simple reflection that generated ξ, the position of ν in the list, or the level of ξ, to name a few. Most of these things, of course, can be recovered from the weight ξ itself. To illustrate this, let us show how one can quickly compute the level of $\xi = (x_1, \ldots, x_l)$ by finding the shortest path in the tree—as a sequence of simple reflections—from ξ to the root node μ.

Reflections(ξ):

1. Let i be the index of the first negative coordinate, $x_i < 0$. If there is no such index, then stop.
2. Compute the weight $\nu = \sigma_i(\xi)$.
3. Output the index i.
4. Set $\xi = \nu$ and go to step 1.

The number of reflections reported by *Reflections*(ξ) is clearly the level of ξ, since the level is *reduced* every time by exactly one in step 2, see (2.1), and the process does not stop until all coordinates are positive.

The procedure *Reflections* is also useful as a "translator" between weights in a W-orbit and elements of W. If one is interested in computing the group W itself, then one can simply compute the orbit of the dominant weight $\delta = (1, 1, \ldots, 1)$. Then the weights in the orbit $W.\delta$ are in one-to-one correspondence with the elements in W, since the isotropy group of δ is trivial, $W_\delta = \{\sigma \in W \mid \sigma(\delta) = \delta\} = 1$. If $\nu \in W.\delta$, then *Reflections*(ν) outputs the sequence of simple reflections, $\sigma_{i_1}, \ldots, \sigma_{i_k}$, whose product, $\sigma = \sigma_{i_1} \cdots \sigma_{i_k}$, defines the Weyl group element corresponding to ν.

One of the many reasons to compute W-orbits is to determine "distinguished coset representatives." These are elements $\sigma \in W$ that represent a coset σW_J in W/W_J. Here, W_J is the subgroup of W generated by a subset of simple reflections, σ_j, indexed by $j \in J \subset \{1, \ldots, l\}$. The representative σ is called *distinguished* if it is the uniquely determined element of minimal length among the elements of the coset σW_J, [3]. Let $\mu = (m_1, \ldots, m_l) \in \bar{\mathscr{C}}$ be the dominant weight defined by $m_i = 0$ for $i \in J$, and $m_i = 1$ for $i \notin J$. Then the isotropy subgroup W_μ is precisely the subgroup W_J, since $\sigma_i(\mu) = \mu \Leftrightarrow m_i = 0$. The weights in the orbit $W.\mu$ are thus in one-to-one correspondence with the cosets in W/W_J. We can use *Branch*(μ) to find all the weights in $W.\mu$ and then apply the procedure *Reflections* to translate the weights to corresponding Weyl group elements. These elements will obviously have minimal length in their cosets, and therefore form the set of distinguished coset representatives. Notice that *Branch* could include the reflection information in step 4, so that a separate application of *Reflections* is not necessary.

There is a certain symmetry in all Weyl group orbits, which can be exploited to cut memory and time requirements for computing an orbit in half. Let ω_0 denote the unique Weyl group element of maximum length. This length $l(\omega_0)$ is equal to the number of positive roots in the group G. Now $\omega_0^2 = 1$, so ω_0 acts as an involution on any W-orbit. What makes this useful is that the action of ω_0 on the weight lattice is trivial to calculate. Namely, ω_0 maps a fundamental weight λ_i to $-\lambda_{s(i)}$ where s is a permutation of the indices $\{1, \ldots, l\}$ satisfying $s^2 = id$. In fact, s is the identity for all simple groups except types A_l, D_l, and E_6 where it is equivalent to the obvious Dynkin diagram automorphism. Furthermore, if $\nu = (n_1, \ldots, n_l)$ has level k, then

$$\omega_0(\nu) = (-n_{s(1)}, \ldots, -n_{s(l)}) \tag{3.1}$$

has level $N - k$ where N is the highest level in the orbit $W.\nu$. Note that N is always $\leq l(\omega_0)$. In practical terms, this means that we need only compute weights up to level $N/2$ and obtain the others by this simple formula.

4. COMPUTER IMPLEMENTATIONS

In this final section, we shall sketch in the language C how to implement the important parts of the above algorithms for computing Weyl group orbits. First, we need a fast procedure for carrying out the reflection of a weight by a simple root. As mentioned in the previous section, this need not take more than three additions and a sign change. To reduce the reflection operation to such minor calculations, we must first encode the Cartan matrix $M = \{c_{ij}\}$ into a more appropriate form, which we call the Dynkin matrix. Let D be a matrix whose ith row, D_i, is a list of the column numbers of the negative entries of the ith row of M with the added adjustment that if the entry is -2 (respectively -3), then the column number is repeated two (respectively three) times. The list D_i is terminated by 0 to mark its end. With this convention, the number of entries in the list D_i is never more than 4, and this we may take as the column dimension of D. The row dimension of D is, of course, always equal to the rank l. For example, the Cartan matrix for the group G_2 is

$$\begin{bmatrix} 2 & -1 \\ -3 & 2 \end{bmatrix},$$

The corresponding Dynkin matrix D has the form

$$\begin{bmatrix} 2 & 0 & * & * \\ 1 & 1 & 1 & 0 \end{bmatrix}.$$

For the group E_6 the Cartan matrix is

$$\begin{bmatrix} 2 & -1 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & -1 \\ 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & -1 & 2 & 0 \\ 0 & 0 & -1 & 0 & 0 & 2 \end{bmatrix}.$$

and the Dynkin matrix D is

$$\begin{bmatrix} 2 & 0 & * & * \\ 1 & 3 & 0 & * \\ 2 & 4 & 6 & 0 \\ 3 & 5 & 0 & * \\ 4 & 0 & * & * \\ 3 & 0 & * & * \end{bmatrix}.$$

According to formula (1.2), for the reflection of a weight $\mu = (m_1, \ldots, m_l)$, by the simple reflection σ_i, only the coordinates indexed by the numbers in the list D_i will change, along with the ith coordinate which always changes sign (since $c_{ii} = 2$). If $c_{ij} = -1$, then m_j changes to $m_j - m_i c_{ij} = m_j + m_i$; if $c_{ij} = -2$, then m_j changes to $m_j - m_j + m_i c_{ij} = m_j + m_i + m_i$; and if $c_{ij} = -3$, then m_j changes to $m_j - m_i c_{ij} = m_j + m_i + m_i + m_i$. Thus, to carry out the reflection of μ by σ_i, we must only change m_i to $-m_i$, and for each j in the list D_i, increment the coordinate m_j by m_i. An outline of this reflection algorithm is as follows (r is the row D_i

and m is the weight μ):

Reflect

Input:	$m = (m_1, \ldots, m_l)$: weight vector
	i: identifies the ith simple reflection
	$r = (r_0, r_1, r_2, r_3)$: row i of the Dynkin matrix
Output:	$m' = (m'_1, \ldots, m'_l)$: reflection of m under σ_i
Procedure:	set $n = m_i$, $j = r_0$, $k = 0$

```
set m_i = −n
while (j > 0) {
   add n to m_j
   increment k by 1
   set j = r_k
}
```

In C, the increment step can be performed directly on a pointer variable. In fact, the best way to maintain the Dynkin matrix is as an array of pointers to arrays of integers (int **d;) initialized so that d[i] points to the array of integers in D_i terminated by 0. The code for the above procedure is then very simple:

```
reflect(m,r,i)
  int *m, *r, i;
{
  int n = m[i], j;
  m[i] = −n;
  while (j = *(r++)) m[j] += n;
}
```

Thus, after reflect(m,d[i],i), the array m contains the coordinates of $\sigma_i(\mu)$. Notice that at most three additions and a change of sign are required in reflect.

It should be mentioned at this point that we are indexing the above arrays from 1 to l, instead of the usual 0 to $l - 1$ for arrays in C. (We still index the rows of D from 0, however.) One could either declare the arrays to be one element longer than necessary, and ignore the offset 0 element, or one could subtract one from the array name, right after it is declared to automatically adjust later references to the array. In any case, the procedures and programs to follow will be clearer if we retain the natural indexing 1 to l.

Before we present the routines for computing orbits, let us first sketch an implementation of the algorithm *Reflections* of the previous section. In step 1 of the algorithm *Reflections*, the index, say i, of the first negative coordinate of the weight must be found. However, after a reflection is performed, we do not need to go back to the first index to start searching for the first negative coordinate of the reflected weight. Before the reflection, all of the coordinates m_j, $j < i$, are nonnegative, and (excluding $m_i < 0$, which becomes $-m_i > 0$) the first coordinate altered by the reflection σ_i is given by the first element of D_i. Therefore, we should jump to this coordinate after any reflection in our search for the first negative coordinate of the reflected weight.

Reflections

Input: $m = (m_1, \ldots, m_l)$: weight vector
D: Dynkin matrix
l: rank of G

Output: list of integers identifying the sequence of simple reflections
level: the level of m
$m' = (m_1', \ldots, m_l')$: the dominant conjugate of m

Procedure: set *level* $= 0$, $i = 1$
while ($i \leq l$)
  if ($m_i < 0$) {
    increment *level* by 1
    output i
    replace m by its reflection under σ_i
    set i equal to the first element in D_i
  } otherwise increment i by 1
output m, *level*

This translates easily into C, as follows:

```
reflections(m, d, rank)
  int *m, **d, rank;
{
  int level = 0, i = 1;
  printf("reflections: ");
  while (i <= rank)
    if (m[i] < 0) {
      ++level;
      printf("\%d ",i);
      reflect(m,d[i],i);
      i = d[i][0];
    } else i++;
  printf("\nlevel: %d",level);
  printf("\nconjugate: ");
  for (i=1; i <= rank; i++) printf("%d ", m[i]);
  printf("\n");
}
```

Let us now tackle the computation of orbits. The first procedure we shall implement for this is the computation of orbits by level described in Section 2. It is important to be able to control the maximum number of levels computed, since an orbit need only be computed up to level $N/2$ where N is the number of levels in the orbit. The rest of the orbit can be obtained by the simple formula (3.1). A routine like *Reflections* applied to $\omega_0(\mu)$ would quickly give the value of N. The basic data structures needed to compute an orbit by level are two linked lists of *weight-nodes*. These *weight-nodes* each consist of a weight vetor and a pointer to the next *weight-node* in the list. The first list, *current-level*, stores the weights just computed (at the start, *current-level* only contains the dominant weight μ), and the second list, *next-level*, stores the new weights as they are generated from *current-level*. Each weight in *current-list* is examined for positive coordinates. If the ith coordinate is positive, the weight is reflected by σ_i and stored in *next-level* if it meets the requirements of Theorem 2.1. When *current-level* is exhausted, it is set equal to *next-level* and the original list is freed. To summarize:

Orbit-by-Level

Data Structures: *levelcount* : array of integers to count each level (initialized to 0)
current-level, next-level : linked lists of *weight-nodes*
each *weight-node* contains:
  $w = (w_1, \ldots, w_l)$:weight vector
  next : pointer to the next *weight-node*

Input: $m = (m_1, \ldots, m_l)$: dominant weight vector (all $m_i \geq 0$)
D : the Dynkin matrix
l : rank of G
maxlevel : maximum level to compute

Output: list of weight vectors $w = (w_1, \ldots, w_l)$ and three associated integers
  the first integer identifies w (its position in the list)
  the second and third integers identify the weight vector and the reflection, respectively, which generated w

Procedure:

```
output m
install m at the head of list current-level
set level = 0, levelcount[level] = 1
set from = 1 (position of parent weight)
set to = 1 (position of new weight, also the current total of weights
  computed)
while (levelcount[level] > 0 and level < maxlevel) {
  increment level by 1
  set from = to (start count-down of weights in current-level)
  for (each weight w in list current-level) {
    for (i = 1 to l)
      if (w_i > 0) {
        reflect w to v by σ_i
        if (v_j ≥ 0 for i < j ≤ l) { (save v?)
          install v at the head of list next-level
          increment levelcount[level] and to by 1
          output v, to, from, i
        }
      }
    decrement from by 1 (first-in, last-out order in current-list)
  }
  free the previous list current-level
  set list current-level = next-level
  set list next-level = empty list
}
```

The structure for weight-nodes can be set up with type-definitions in C.

```
typedef struct wnode {
  int weight[MAXRANK+1];
  struct wnode *next;
} WEIGHTNODE, *WEIGHTPTR;
```

The corresponding code in C for the above procedure is given below.

```
orbit_by_level(m, d, rank, levelcount, maxlevel)
  int *m, **d, rank, *levelcount, maxlevel;
{
  int level = 0, from = 1, to = 1, i;
  int v[MAXRANK+1];
  WEIGHTPTR current_level, next_level, wp, install( );
  levelcount[level] = 1;
  output(m,rank,from,0,to);
  current_level = install(m,rank,(WEIGHTPTR)NULL);
  next_level = NULL;
  while (levelcount[level] > 0 && ++level <= maxlevel) {
    from = to;
    for (wp = current_level; wp != NULL; wp = wp > next, from--) {
      for (i = 1; index <= rank; i++)
        if (wp > weight[i] > 0) {
          reflect(wp > weight,v,rank,d[i],i);
          if (verify(v,rank,i)) {
            next_level = install(v,rank,next_level);
            ++levelcount[level];
            output(v,rank,from,i,++to);
          }
      free((char *)wp);
    }
    current_level = next_level; next_level = NULL;
  }
  printf("total: %d\n", to);
}
```

The supporting routine verify(v,rank,i) returns one if the coordinates of v after the ith are nonnegative and 0 otherwise. This is the criterion of Theorem 2.1 for saving a new weight in the orbit. New weights are added to the head of the list of weights being created by install, which returns a pointer to the first weight in the list. The routine output prints out a weight in the orbit (v), its position on the list of weights (to), the reflection which created it (i), and the position of the weight from whence it came (from). The routine reflect(v1,v2,rank,r,i) has been modified here to reflect v1 *onto* v2 via the ith simple reflection. The memory used by a weight in the list current_level is freed as soon as possible. Nevertheless, the lists can grow quite large and become the main obstacle to using this technique for large orbits. MAXRANK is a constant representing the maximum rank of the group G allowed in the program. Since MAXRANK affects the size of the weight nodes, and thus the total size of the linked weight lists, its value will depend on the amount of memory available.

The last example we give generates the Weyl group orbit *depth first* by implementing the algorithm *Branch* in Section 3.

Orbit-Depth-First

Data Structures: *levelcount* : array of integers to count each level (initialized to 0)
stack of *weight-structures*
each *weight-structure* contains:
  $w = (w_1, \ldots, w_l)$: weight vector
  level : level of w
  from : integer identifying the parent weight
  i : next index to examine

Input: $m = (m_1, \ldots, m_l)$: dominant weight
D: Dynkin matrix
l: rank of G
maxlevel: the maximum level to compute

Output: list of weight vectors $w = (w_1, \ldots, w_l)$ and four associated integers
the first integer identifies w (its position in the list)
the second integer is the level of w
the third and fourth integers identify the weight vector and the reflection, respectively, which generated w

Procedure:

```
set level = 0, levelcount[level] = 1, i = 1
set from = 1 (position of parent weight)
set to = 1 (position of new weight, also the current total of weights
  computed)
output m
push m, level, from, i onto stack
while (stack is not empty) {
  pop w, level, from, i from stack
  while (i ≤ l)
    if (w_i > 0, {
      reflect w to v by σ_i
      if (v_j ≥ 0 for i < j ≤ l) { (save v ?)
        if (i + 1 ≤ l)
          push w, level, from, i + 1 onto stack (return to w later)
        increment level by 1
        increment levelcount[level] and to by 1
        output v, level, from, i, to
        if (level ≥ maxlevel) jump out of this while loop
        copy v to w (continue with orbit of w = v)
        set from = to (position of w in list)
        set i = first element in D_i
      } otherwise increment i by 1
    } otherwise increment i by 1
}
```

The C code to implement this procedure appears below. The routines for managing the stack are the usual pop and push; copy(v1,v2,rank) simply copies the array v1 to the array v2. The variable from holds the position of the previous weight, the variable to holds the position of the new weight (which is also the current total number of weights generated), and i identifies the simple reflection that generates the new weight from the previous one. The level of each weight is maintained in the variable level and the total number in each level is recorded in the array levelcount. Thus, even though the output of orbit_depth_first is not naturally organized according to levels, the level information can still be maintained.

```
orbit_depth_first(m, d, rank, levelcount, maxlevel)
  int *m, **d, rank, *levelcount, maxlevel;
{
  int level = 0, from = 1, to = 1, i = 1;
  int w[MAXRANK+1], v[MAXRANK+1];
  levelcount[level] = 1;
  output(m,rank,level,from,0,to);
  push(m,rank,level,from,i);
```

```
    while (pop(w,rank,&level,&from,&i))
      while (i <= rank)
        if (w[i]>0) {
          reflect(w,v,rank,d[i],i);
          if (verify(v,rank,i)) {
            if (i < rank)
              push(w,rank,level,from,i+1);
            ++levelcount[++level];
            output(v,rank,level,from,i,++to);
            if (level >= maxlevel) break;
            copy(v,w,rank);
            from = to;
            i = d[i][0];
          } else i++;
        } else i++;
}
```

In this implementation the values of from, to, and i are saved and recalled with the weights to make it easier to reconstruct information about the orbit, but none of them is strictly necessary for the algorithm. The first coordinate to examine when a weight is popped from the stack is stored in i (one could always start at 0), and level is used to cut off the computation when maxlevel is reached (one could always compute the whole orbit and never apply (3.1)). In any case, the total memory requirements for this algorithm are quite small. The stack can be an array of structures, each containing the coordinates of the weight, the level, i, and so on. The total number of structures in the stack never needs to be more than the number of positive roots in the group, or even just half of this number if symmetry is exploited. Thus even for $E8$ one could manage with a stack of only 60 structures. A rough general estimate for the required stack size is the square of MAXRANK divided by 2 as long as the rank is at least 11.

Table I lists the size of each level of the Weyl group orbit of $\delta = \lambda_1 + \cdots + \lambda_l$ for the groups $E6$, $E7$, and $E8$. These sizes also correspond to the number of Weyl group elements of a given length (= level), see Section 3. The results were obtained using a version of the routine orbit_depth_first on a Sun 3/280.

REFERENCES

1. BOREL, A. *Linear Algebraic Groups.* Benjamin, New York, 1971.
2. BOTT, R. Homogeneous vector bundles. *Ann. Math. 66* (1957), 203–248.
3. CARTER, R. *Simple Groups of Lie Type.* Wiley, New York, 1972.
4. HUMPHREYS, J. *Introduction to Lie Algebras and Representation Theory.* Springer Verlag, New York, 1972.
5. JACOBSON, N. *Lie Algebras.* Wiley, New York, 1962.
6. WEYL, H. Theorie der Darstellung der halb-einfacher Gruppen durch lineare Transformationen. *Math. Z. 24* (1926), 377–395.

Received October 1988; revised April 1989; accepted April 1989