THE SNOWFLAKE DECODING ALGORITHM

J. B. NATION AND CATHERINE WALKER

Abstract. This paper describes an automated algorithm for generating a group code using any unitary group, initial vector, and generating set that satisfy a necessary condition. Examples with exceptional complex reflection groups, as well as an analysis of the decoding complexity, are also included. Extending these examples, group codes based on wreath products of complex matrix groups are constructed. Efficient algorithms for encoding and decoding these codes are described.

1. Introduction

Group codes, introduced by Slepian [7, 8], all follow the same basic plan. The code is determined by a finite group $\mathbf{G}$ of isometries acting on a vector space V, and an initial vector $\mathbf{x}_0$. The codewords are a subset of the orbit $\mathbf{G}\mathbf{x}_0 = \{g\mathbf{x}_0 : g \in \mathbf{G}\}$. A codeword $\mathbf{x} = g^{-1}\mathbf{x}_0$ is transmitted, and the received vector is $\mathbf{r} = \mathbf{x} + \mathbf{n}$ where $\mathbf{n}$ represents channel noise. Let $\mathbf{r}_0 = \mathbf{r}$. Recursively, given $\mathbf{r}_k$, we can apply a transformation c_{k+1} from some specified set $X_k \subseteq \mathbf{G}$ to obtain $\mathbf{r}_{k+1} = c_{k+1}\mathbf{r}_k$. The transformation c_{k+1} is chosen so that $\|\mathbf{r}_{k+1} - \mathbf{x}_0\| < \|\mathbf{r}_k - \mathbf{x}_0\|$ in some metric. (We use the Euclidean metric throughout.) After a fixed number m of steps, depending on the group, terminate and decode as the group element $c_m \dots c_1$. The decoding is successful if $c_m \dots c_1 = g$.

In this paper a method to construct effective unitary group codes will be presented. Given any finite unitary group $\mathbf{G}$, a suitable initial vector $\mathbf{x}_0$, and a set X of generators for $\mathbf{G}$ satisfying

(†) for every $\mathbf{w} \in \mathbf{G}\mathbf{x}_0 - \{\mathbf{x}_0\}$ there exists $c \in X$ such that $\|c\mathbf{w} - \mathbf{x}_0\| < \|\mathbf{w} - \mathbf{x}_0\|$

it yields a decoding algorithm. For some groups $\mathbf{G}$, with a suitable choice of $\mathbf{x}_0$ and X, decoding algorithms of very low complexity are obtained. Examples using complex reflection groups will also be provided, along with an analysis of the decoding complexity.

Slepian used orthogonal groups on real vector spaces $\mathbb{R}^n$. Among the groups proposed by Slepian are the groups $\mathbf{B}_n$ of signed permutation matrices (a.k.a. variant II). These real reflection groups have the form of a wreath product $\mathbf{C}_2 \wr \mathbf{S}_n$, where $\mathbf{C}_2$ denotes the group $\{1, -1\}$. Fossorier,

Date: September 29, 2015.

2010 *Mathematics Subject Classification.* 94B60, 20G20.

Key words and phrases. group code, complex reflection group, wreath product.

Nation and Peterson [9] introduced subgroup decoding as an efficient way to decode codes based on real reflection groups. This was extended to certain complex reflection groups $\mathbf{G}(r, 1, n)$ acting on $\mathbb{C}^n$ in Kim, Nation and Shepler [5]. The groups $\mathbf{G}(r, 1, n)$ are wreath products of cyclic groups, with $\mathbf{G}(r, 1, n) = \mathbf{C}_r \wr \mathbf{S}_n$ for the group $\mathbf{C}_r$ of complex r-th roots of unity. For example, the Coxeter group $\mathbf{B}_n$ is $\mathbf{G}(2, 1, n)$. This structure allows us to generalize the methods used on real reflection groups. However, subgroup decoding does not work properly on most other types of complex reflection groups. Kim [4] devised an algorithm to decode these groups correctly, which was considerably refined by Walker [15], using the Snowflake algorithm described below. Thus for certain relatively small complex reflection group codes, we now have efficient decoding algorithms. One can then take wreath products of these codes, to produce much larger codes that can still be decoded effectively.

The authors wish to thank Ian Blake for a useful suggestion.

2. Preliminaries

All vectors are column vectors and $\mathbf{M}^H$ denotes the conjugate transpose of a complex vector or matrix $\mathbf{M}$. A *unitary matrix* $\mathbf{U}$ is a square complex matrix such that $\mathbf{U}^H\mathbf{U} = \mathbf{I}$. A *unitary group* is a group of $n \times n$ unitary matrices, with the group operation of the usual matrix multiplication. The *inner product* is $\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^H\mathbf{y}$.

Recall that the *norm* of the inner product space is defined as $\|\mathbf{x}\| = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle}$. The distance between vectors $\mathbf{x}$ and $\mathbf{y}$ is given by $\|\mathbf{x} - \mathbf{y}\|$. It is well known that a unitary group acting on a vector space preserves the inner product. Thus a unitary group acting on a vector space is an isometry with respect to the norm from the inner product.

3. The Snowflake Decoding Algorithm

The *Snowflake* program described below produces decoding schemes for group codes. The input to the program consists of

- a finite unitary group $\mathbf{G}$ acting on a complex vector space $\mathbb{C}^n$ (or more generally, a group of isometries on a vector space),
- a fixed set X of generators for $\mathbf{G}$,
- a fixed initial vector $\mathbf{x}_0$,
- a set of codewords $W \subseteq \mathbf{G}\mathbf{x}_0$,

such that the condition (†) from the introduction is satisfied.

The output of the program is

- an integer m (the number of decoding passes),
- real numbers $d_1 > d_2 > \cdots > d_m = 0$ and $\epsilon_1, \ldots, \epsilon_m$ with each $\epsilon_j > 0$,
- a sequence of linearly ordered sets $X_1, \ldots, X_m$ with each $X_j \subseteq X \cup \{\mathbf{I}\}$.

The *Snowflake decoding algorithm* for this setup proceeds thusly. As usual, a codeword $\mathbf{w} = g^{-1}\mathbf{x}_0$ is transmitted, corresponding to the message "g", and the received vector is $\mathbf{r} = \mathbf{w} + \mathbf{n}$, where $\mathbf{n}$ represents noise. To decode, let $\mathbf{r}_0 = \mathbf{r}$. Recursively, given $\mathbf{r}_k$ with $k < m$, choose c_{k+1} to be the first element c of the ordered set X_{k+1} such that $\|c\mathbf{r}_k - \mathbf{x}_0\| < d_{k+1} + \epsilon_{k+1}$; if no such $c \in X_{k+1}$ exists, or if $\|\mathbf{r}_k - \mathbf{x}_0\| < d_{k+1} + \epsilon_{k+1}$ already holds, let $c_{k+1} = \mathbf{I}$. After m steps, terminate and decode as the group element $c_m \dots c_1$.

The two key parts to the Snowflake coding scheme are how the parameters d_i and ϵ_i are determined, and which elements should be included in each set X_i, along with their ordering.

3.1. **How to choose d_i and ϵ_i.** In general, the generating set X for $\mathbf{G}$ may include any generating matrices, and powers thereof, elements used in the group's presentation along with their inverses. In order for the algorithm to work, X must be chosen so that the condition (†) is satisfied. Let $W_0 = W$ be the set of all codewords and $d_0 = \max_{\mathbf{w}\in W_0} \|\mathbf{w} - \mathbf{x}_0\|$. Define

$$d_1 = \max_{\mathbf{w}\in W_0} \min_{h\in X} \|h\mathbf{w} - \mathbf{x}_0\|.$$

The distance d_1 is the closest all of the codewords can get to the initial vector using any of the transformations in X. Now set

$$W_1 = \{\mathbf{w} \in W_0 : \|\mathbf{w} - \mathbf{x}_0\| \le d_1\}.$$

Thus we get the subset W_1 of codewords which are within the distance d_1 of the initial vector. Since (†) is satisfied, for every codeword $\mathbf{w} \in W_0$ there exists $h \in X$ such that $\|h\mathbf{w} - \mathbf{x}_0\| < \|\mathbf{w} - \mathbf{x}_0\|$. Therefore $d_1 < d_0$ and $W_1 \subset W_0$.

Recursively, for $j < m$ let

$$d_{j+1} = \max_{\mathbf{w}\in W_j} \min_{h\in X} \|h\mathbf{w} - \mathbf{x}_0\|$$
$$W_{j+1} = \{\mathbf{w} \in W_j : \|\mathbf{w} - \mathbf{x}_0\| \le d_{j+1}\}.$$

We continue this process until we get $d_1 > d_2 > \cdots > d_m = 0$, and $W_0 \supset W_1 \supset \cdots \supset W_m = \{\mathbf{x}_0\}$.

Ideally, we would choose c_{k+1} such that $\|c_{k+1}\mathbf{r}_k - \mathbf{x}_0\| \le d_{k+1}$. However some tolerance must be built in to account for noise. The tolerance, ϵ_i, should intuitively be half the difference between the distance d_i and distance to $\mathbf{x}_0$ of the nearest codeword outside the ball of radius d_i. For $k > 0$ let

$$e_k = \min_{\substack{\mathbf{w}\in W_0 \\ \|\mathbf{w}-\mathbf{x}_0\| > d_k}} \|\mathbf{w} - \mathbf{x}_0\|.$$

Then set $\epsilon_k = \frac{1}{2}(e_k - d_k)$.

3.2. **Determining the set X_{k+1}.** Again we consider how one would decode codewords $g^{-1}\mathbf{x}_0$ without noise. If we let $X_i = X$ for each step, then the decoding process will terminate; however, it will be much longer than necessary. The program will apply each transformation h in the set, then calculate $\|h\mathbf{r}_k - \mathbf{x}_0\|$ to see if it is at most d_{k+1}; if so, choose $c_{k+1} = h$. The program continues to make these comparisons until a desired transformation is found. Decreasing the number of comparisons by finding ordered sets X_i properly contained in X will increase the efficiency of the decoding process.

Example. Start by applying all transformations in the set X to all codewords in $W_k - W_{k+1} = \{\mathbf{w} \in W_k : \|\mathbf{w} - \mathbf{x}_0\| > d_{k+1}\}$. This will produce an array of codewords and the transformations which will move them to within d_{k+1} of the initial vector. For example, the table below shows a relationship between codewords and transformations. An entry $a_{i,j} = 1$ if $\|h_j\mathbf{w}_i - \mathbf{x}_0\| \leq d_{k+1}$, and zero otherwise.

TABLE 1. Output for d_{k+1}

	h_1	h_2	h_3	h_4	h_5
$\mathbf{w}_1$	1	0	1	1	1
$\mathbf{w}_2$	1	1	0	0	1
$\mathbf{w}_3$	1	1	1	0	0
$\mathbf{w}_4$	0	0	0	1	0
$\mathbf{w}_5$	0	0	1	0	0
$\mathbf{w}_6$	0	1	0	0	0
$\mathbf{w}_7$	1	0	0	1	0
sum	4	3	3	3	2

Naively, we could choose the transformations based on how many codewords they could be used for. The transformation h_1 is used 4 times, so it would then be the first one chosen. Then if we ignore rows 1, 2, 3, and 7 and sum the remaining rows, we find that h_2, h_3, and h_4 all need to be used once and must be included in the set X_{k+1} as well. The transformation h_5 can be excluded. Thus we obtain the ordered set $Y^{(1)} = \{h_1, h_2, h_3, h_4\}$ as our first candidate for X_{k+1}. If we compute the number of comparisons that need to be made to decode these seven elements in this one round we get one comparison each for codewords $\mathbf{w}_1$,$\mathbf{w}_2$,$\mathbf{w}_3$, and $\mathbf{w}_7$ since the program would choose h_1 for all of them. Then, for codewords $\mathbf{w}_6$, $\mathbf{w}_5$, and $\mathbf{w}_4$ we have 2, 3, and 4 comparisons respectively. This totals to 13 comparisons for this step of decoding.

Now we will look at another way to determine the set X_{k+1}. Instead of considering only the frequency of a transformation, let us also take into account the criticality of the transformation. In Table 1 we can see that h_2, h_3, and h_4 must be used since they are the only transformations available for $\mathbf{w}_6$, $\mathbf{w}_5$, and $\mathbf{w}_4$. Therefore, they are more critical than a transformation which only has overlapping uses. To quantify this idea, consider the weighted

table below which has row sums equal to 1. According to Table 2, h_4 should

TABLE 2. Weighted Output for d_{k+1}

	h_1	h_2	h_3	h_4	h_5
$\mathbf{w}_1$	$1/4$	0	$1/4$	$1/4$	$1/4$
$\mathbf{w}_2$	$1/3$	$1/3$	0	0	$1/3$
$\mathbf{w}_3$	$1/3$	$1/3$	$1/3$	0	0
$\mathbf{w}_4$	0	0	0	1	0
$\mathbf{w}_5$	0	0	1	0	0
$\mathbf{w}_6$	0	1	0	0	0
$\mathbf{w}_7$	$1/2$	0	0	$1/2$	0
sum	$17/12$	$20/12$	$19/12$	$21/12$	$7/12$

be the first transformation in X_{k+1}. Then, we omit column 4 and rows 1, 4, and 7 and recompute the sums. We get that column 2 has the highest sum, so h_2 should be the next transformation in X_{k+1}. The only remaining codeword is $\mathbf{w}_5$, making h_3 the final transformation in our set. In this way we obtain $Y^{(2)} = \{h_4, h_2, h_3\}$ as our candidate for X_{k+1}. Note that $|Y^{(2)}| < |Y^{(1)}|$ but more importantly, fewer comparisons are made using the set $Y^{(2)}$ rather than the set $Y^{(1)}$. With $Y^{(2)}$, a total of $(3\cdot1)+(3\cdot2)+(1\cdot3) = 12$ comparisons are made for all codewords in this round. Thus we should choose $X_{k+1} = Y^{(2)}$.

The previous example highlighted some of the key concepts for analysis when forming the decoding sets X_i. Here is the general algorithm to find the ordered set X_k for $1 \le k \le m$. The first element is $h_0 = \mathbf{I}$, to cover the possibility that $\|\mathbf{r}_k - \mathbf{x}_0\| < d_{k+1} + \epsilon_{k+1}$ already holds. To find the generators from X that go into X_k, define the function

$$N(\mathbf{w}, d) = |\{g \in X : \|g\mathbf{w} - \mathbf{x}_0\| \le d\}|$$

which counts the number of transformations which move codeword $\mathbf{w}$ to within d of the initial vector. Recall that in the first part of the algorithm setup we have defined the sets of codewords $W = W_0 \supset W_1 \supset \cdots \supset W_m = \{\mathbf{x}_0\}$. Let

$$B_{k,1}(g) = \{\mathbf{w} \in W_{k-1} - W_k : \|g\mathbf{w} - \mathbf{x}_0\| \le d_k\}$$

and

$$S_{k,1}(g) = \sum_{\mathbf{w} \in B_{k,1}(g)} \frac{1}{N(\mathbf{w}, d_k)}.$$

Choose h_1 that maximizes $S_{k,1}$, i.e., $S_{k,1}(h_1) \ge S_{k,1}(g)$ for all $g \in X$. Recursively, let

$$B_{k,i+1}(g) = \{\mathbf{w} \in W_{k-1} - W_k : \|g\mathbf{w} - \mathbf{x}_0\| \le d_k\} - \bigcup_{0 \le t \le i} B_{k,t}(h_t)$$

$$S_{k,i+1}(g) = \sum_{\mathbf{w} \in B_{k,i+1}(g)} \frac{1}{N(\mathbf{w}, d_k)}$$

and choose h_{i+1} that maximizes $S_{k,i+1}$. Terminate when $S_{k,r+1}(g) = 0$ for all $g \in X$, or equivalently when $B_{k,r+1}(g) = \emptyset$ for all $g \in X$. The definition of d_k guarantees that for all codewords $\mathbf{w} \in W_{k-1}$, there exists $c \in X$ such that $\|c\mathbf{w} - \mathbf{x}_0\| \leq d_k$. Therefore this process will terminate in at most $|X|$ steps. The ordered set $X_k = \{\mathbf{I}, h_1, \ldots, h_r\}$ is the ordered set of transformations to be used in round k of decoding.

With these sets each codeword now has a decoding canonical form. A codeword $\mathbf{w} = g^{-1}\mathbf{x}_0$ will be decoded as $g = c_m \ldots c_1$ where each c_i is the first element of the ordered set X_i such that $\|c_i \ldots c_1\mathbf{w} - \mathbf{x}_0\| \leq d_i$.

3.3. **Decoding Theorem.** Now that the decoding algorithm has been generated, we use the following theorem to ensure proper decoding with sufficiently small noise.

Theorem 3.1. *If* $\|\mathbf{r} - \mathbf{w}\| < \eta$, *where* $\mathbf{w} = g^{-1}\mathbf{x}_0$ *and* $\eta = \min(\epsilon_1, \ldots, \epsilon_m)$, *then the Snowflake algorithm decodes* $\mathbf{r}$ *to* g.

Proof. Let $\mathbf{w}$ decode to $g = c_m \ldots c_1$. Let $\mathbf{r}_0 = \mathbf{r}$, $\mathbf{w}_0 = \mathbf{w}$, $\mathbf{r}_{j-1} = c_{j-1} \ldots c_1\mathbf{r}_0$ and $\mathbf{w}_{j-1} = c_{j-1} \ldots c_1\mathbf{w}_0$. The received vector $\mathbf{r}$ would not decode to g if at some step c_j is not chosen. This would occur if a preceding transformation $h \neq c_j$ in the ordered set X_j is chosen, or c_j fails to move r_{j-1} to within d_j of $\mathbf{x}_0$.

Suppose $\mathbf{r}$ has decoded correctly through step $j-1$. Note that

$$\|\mathbf{r}_{j-1} - \mathbf{w}_{j-1}\| = \|c_{j-1} \ldots c_1\mathbf{r}_0 - c_{j-1} \ldots c_1\mathbf{w}_0\| = \|\mathbf{r}_0 - \mathbf{w}_0\| < \eta.$$

By the triangle inequality,

$$(1)\quad \big|\|h\mathbf{r}_{j-1} - \mathbf{x}_0\| - \|h\mathbf{w}_{j-1} - \mathbf{x}_0\|\big| \leq \|h\mathbf{r}_{j-1} - h\mathbf{w}_{j-1}\| = \|\mathbf{r}_{j-1} - \mathbf{w}_{j-1}\| < \eta$$

for all $h \in \mathbf{G}$.

Consider step j. Suppose some $h \in X_j$ precedes c_j in the ordering of X_j, and satisfies $\|h\mathbf{r}_{j-1} - \mathbf{x}_0\| < d_j + \epsilon_j$. However, since h was not chosen for step j by the Snowflake algorithm we know $\|h\mathbf{w}_{j-1} - \mathbf{x}_0\| \geq d_j + 2\epsilon_j$. We then have $\big|\|h\mathbf{w}_{j-1} - \mathbf{x}_0\| - \|h\mathbf{r}_{j-1} - \mathbf{x}_0\|\big| > \epsilon_j \geq \eta$ which contradicts equation (1). Thus, no preceding $h \neq c_j$ can be chosen.

For the other type of error to occur it would mean that $\|c_j\mathbf{r}_{j-1} - \mathbf{x}_0\| \geq d_j + \epsilon_j$. However, by definition $\|c_j\mathbf{w}_{j-1} - \mathbf{x}_0\| \leq d_j$. Therefore, $\big|\|c_j\mathbf{r}_{j-1} - \mathbf{x}_0\| - \|c_j\mathbf{w}_{j-1} - \mathbf{x}_0\|\big| \geq \epsilon_j \geq \eta$ which again contradicts equation (1). Hence, at each step j, c_j is chosen.

By induction, for step m we then have $\|\mathbf{r}_m - \mathbf{w}_m\| = \|c_m \ldots c_1\mathbf{r} - \mathbf{x}_0\| < \eta \leq \epsilon_m$. Therefore, $\mathbf{r}$ decodes to $c_m \ldots c_1 = g$. Thus for sufficiently small noise the Snowflake algorithm decodes correctly. □

4. Some exceptional complex reflection groups

The irreducible complex reflection groups were classified by Shephard and Todd [6]. This section uses describes codes based on some of these irreducible finite complex reflection groups. Examples of how to create and decode codes using two- and three-dimensional complex reflection groups

are presented. Then we describe group codes based on wreath products of some two-dimensional complex reflection groups, because there are efficient decoding schemes for their group codes separately.

The following two-dimensional groups are good candidates, with simple presentations (that can also be given as Coxeter diagrams).

- $\mathbf{G}_4$: $\mathbf{A}^3 = \mathbf{B}^3 = \mathbf{I}$, $\mathbf{ABA} = \mathbf{BAB}$ (24 elements)
- $\mathbf{G}_8$: $\mathbf{A}^4 = \mathbf{B}^4 = \mathbf{I}$, $\mathbf{ABA} = \mathbf{BAB}$ (96 elements)
- $\mathbf{G}_{16}$: $\mathbf{A}^5 = \mathbf{B}^5 = \mathbf{I}$, $\mathbf{ABA} = \mathbf{BAB}$ (600 elements)
- $\mathbf{G}_5$: $\mathbf{A}^3 = \mathbf{B}^3 = \mathbf{I}$, $\mathbf{ABAB} = \mathbf{BABA}$ (72 elements)
- $\mathbf{G}_{20}$: $\mathbf{A}^3 = \mathbf{B}^3 = \mathbf{I}$, $\mathbf{ABABA} = \mathbf{BABAB}$ (360 elements)

There is a straightforward way to explicitly construct generators for these groups, denoted $\mathbf{A}$ and $\mathbf{B}$, using the method of Householder [2, 3]. The eigenvalues for the generators are $\lambda = e^{\frac{2\pi i}{3}}$ for $\mathbf{G}_4$, $\mathbf{G}_5$ and $\mathbf{G}_{20}$; $\lambda = i$ for $\mathbf{G}_8$; and $\lambda = e^{\frac{2\pi i}{5}}$ for $\mathbf{G}_{16}$. We can use the matrices $\mathbf{A} = \mathbf{I} - (1-\lambda)\mathbf{u}\mathbf{u}^H$ where $\mathbf{u} = [0, 1]^T$ and $\mathbf{B} = \mathbf{I} - (1-\lambda)\mathbf{v}\mathbf{v}^H$ with $\mathbf{v} = [v_1, v_2]^T$ real and $v_1^2 + v_2^2 = 1$. In practice, we want $v_1 > 0$ and $v_2 < 0$. Computing using the relations $\mathbf{ABA} = \mathbf{BAB}$ etc., we obtain the following values for v_2.

- For $\mathbf{G}_4$, $v_2 = -\sqrt{\frac{1}{3}}$.
- For $\mathbf{G}_8$, $v_2 = -\sqrt{\frac{1}{2}}$.
- For $\mathbf{G}_{16}$, $v_2 = -\sqrt{\frac{5+\sqrt{5}}{10}}$.
- For $\mathbf{G}_5$, $v_2 = -\sqrt{\frac{2}{3}}$.
- For $\mathbf{G}_{20}$, $v_2 = -\sqrt{\frac{3+\sqrt{5}}{6}}$.

By symmetry, it makes sense to choose $\mathbf{x}_0$ so that the distances satisfy $\|\mathbf{A}^{-1}\mathbf{x}_0 - \mathbf{x}_0\| = \|\mathbf{B}^{-1}\mathbf{x}_0 - \mathbf{x}_0\|$. Now

$$\begin{aligned}\|\mathbf{A}^{-1}\mathbf{x}_0 - \mathbf{x}_0\| &= \|\mathbf{x}_0 - \mathbf{A}\mathbf{x}_0\| = \|(\mathbf{I} - \mathbf{A})\mathbf{x}_0\| \\ &= \|(1-\lambda)\mathbf{u}\mathbf{u}^H\mathbf{x}_0\| = |1-\lambda|(\mathbf{x}_0^H\mathbf{u}\mathbf{u}^H\mathbf{x}_0)^{\frac{1}{2}} \\ &= |1-\lambda||\mathbf{u}^H\mathbf{x}_0|.\end{aligned}$$

So the required condition is $|\mathbf{u}^H\mathbf{x}_0| = |\mathbf{v}^H\mathbf{x}_0|$. Again it is convenient to choose $\mathbf{x}_0$ real. For the groups we are considering, one obtains the following non-normalized initial vectors.

- For $\mathbf{G}_4$, $\mathbf{x}_0 = (\frac{\sqrt{2}+\sqrt{6}}{2}, 1)$.
- For $\mathbf{G}_8$, $\mathbf{x}_0 = (1+\sqrt{2}, 1)$.
- For $\mathbf{G}_{16}$, $\mathbf{x}_0 = (\frac{1+\sqrt{5}+\sqrt{10+\sqrt{20}}}{2}, 1)$.
- For $\mathbf{G}_5$, $\mathbf{x}_0 = (\sqrt{2}+\sqrt{3}, 1)$.
- For $\mathbf{G}_{20}$, $\mathbf{x}_0 = (\frac{3+\sqrt{5}+\sqrt{18+\sqrt{180}}}{2}, 1)$.

Thus we can easily construct the generator matrices and initial vector for these $\mathbf{G}_k$. With a little more work, we can identify all the reflections, which

are conjugates of the generators. Encoding is easy: choose some canonical form for each $g \in \mathbf{G}$, and transmit $g^{-1}\mathbf{x}_0$.

These exceptional reflection groups generally do not admit subgroup decoding [4]. However, they can be efficiently decoded by the Snowflake algorithm. For the generating sets, let $X \subset \mathbf{G}$ consist of all the reflections in $\mathbf{G}$, the element in the presentation of $\mathbf{G}$, and its inverse. For example, for $\mathbf{G}_4$, $\mathbf{G}_8$, and $\mathbf{G}_{20}$, X would include the reflections, the element $\mathbf{ABA}$, and its inverse $\mathbf{A}^2\mathbf{B}^2\mathbf{A}^2$, $\mathbf{A}^3\mathbf{B}^3\mathbf{A}^3$, or $\mathbf{A}^4\mathbf{B}^4\mathbf{A}^4$, respectively. For larger dimensional groups, additional relation elements may also be included.

4.1. **The Snowflake Algorithm Results.** Table 3 shows the results of the Snowflake algorithm for the group $\mathbf{G}_8$. Not shown in the table is $d_0 = 2.0$ which is the maximum distance of all codewords before the decoding process begins. Following the steps described in section 3.1, we get the distances d_i, and from the procedure in section 3.2 we get the ordered sets X_i.

TABLE 3. Snowflake Results for $\mathbf{G}_8$

Step i	Distance d_i	Ordered Set X_i
1	1.000	{$\mathbf{I}$,$\mathbf{A}^2\mathbf{B}^2\mathbf{A}^2$, $\mathbf{B}^2\mathbf{A}^2\mathbf{B}^2$, $\mathbf{B}\mathbf{A}^2\mathbf{B}^3$, $\mathbf{B}^3\mathbf{A}^2\mathbf{B}$, $\mathbf{A}^3\mathbf{B}^3\mathbf{A}^3$, $\mathbf{ABA}$ }
2	0.541	{ $\mathbf{I}$,$\mathbf{A}^3\mathbf{B}^3\mathbf{A}^3$, $\mathbf{BAB}^3$, $\mathbf{ABA}^3$, $\mathbf{A}^3\mathbf{B}^3\mathbf{A}$, $\mathbf{AB}^3\mathbf{A}^3$,$\mathbf{ABA}$, $\mathbf{B}$, $\mathbf{A}$ }
3	0.000	$\{\mathbf{I}, \mathbf{B}, \mathbf{B}^3, \mathbf{A}, \mathbf{A}^3\}$

Within three rounds all codewords can be decoded.

One way to compute the efficiency of this type of algorithm is to count the average number of comparisons γ for decoding, and then calculate $\delta = \frac{\gamma}{\log_2 |\mathbf{G}|}$, the average number of comparisons per bit.

For $\mathbf{G}_4$, we have $m = 3$ rounds of comparisons, with $\gamma = 3.7$ and $\delta = .81$. For $\mathbf{G}_5$, there are $m = 3$ rounds of comparisons, with $\gamma = 5.5$ and $\delta = .90$. For $\mathbf{G}_8$, it is $m = 4$, with $\gamma = 5.6$ and $\delta = .85$. The group $\mathbf{G}_{16}$ did not work as well, with $m = 4$, $\gamma = 12.9$ and $\delta = 1.4$. The group $\mathbf{G}_{20}$ is in between, with $m = 4$, $\gamma = 8.9$ and $\delta = 1.04$.

4.2. **Example 2:** The three-dimensional complex reflection group $\mathbf{G}_{26}$ has the presentation $\mathbf{A}^2 = \mathbf{B}^3 = \mathbf{C}^3 = \mathbf{I}$, $\mathbf{ABAB} = \mathbf{BABA}$, $\mathbf{BCB} = \mathbf{CBC}$, and $\mathbf{AC} = \mathbf{CA}$. The group $\mathbf{G}_{26}$ has 1296 elements, and 33 reflections.

Following the procedures in sections 3.1 and 3.2 produces the distances and sets for decoding $\mathbf{G}_{26}$. Not shown in table 4 is $d_0 = 2$, the maximum distance of all codewords prior to decoding. Within five rounds all codewords can be decoded, with $\gamma = 15.3$ and $\delta = 1.48$.

5. WREATH PRODUCTS $\mathbf{G} \wr \mathbf{S}_n$

For a group $\mathbf{G}$ of $k \times k$ unitary matrices, the group $\mathbf{G} \wr \mathbf{S}_n$ consists of all $kn \times kn$ block permutation matrices with entries from $\mathbf{G}$. That is, thinking

TABLE 4. Snowflake Results for $\mathbf{G}_{26}$

Step i	Distance d_i	Ordered Set X_i
1	1.0647	$\{\mathbf{I}, \mathbf{ABC^2BAB^2CB^2A}$, $\mathbf{BC^2BAB^2CB^2}$, $\mathbf{CB^2ACBACB^2CB}$, $\mathbf{CB^2ACB^2ACB^2CB}$, $\mathbf{BAC^2B^2ACB^2}$, $\mathbf{B^2ACBAC^2B}$, $\mathbf{B^2ACB^2AC^2B}$, $\mathbf{BAC^2BACB^2}$, $\mathbf{BACBAC^2B^2}$, $\mathbf{B^2AC^2B^2ACB}$, $\mathbf{BACB^2AC^2B^2}$, $\mathbf{B^2AC^2BACB}$, $\mathbf{CB^2ABC^2}$, $\mathbf{C^2BAB^2C}\}$
2	0.9466	$\{\mathbf{I}, \mathbf{B^2C^2B^2}$, $\mathbf{BCB}$, $\mathbf{BACBAC^2B^2}$, $\mathbf{B^2AC^2B^2ACB}$, $\mathbf{ABAB}$, $\mathbf{B^2AB^2A}$, $\mathbf{AC}$, $\mathbf{AB^2C^2BA}$,$\mathbf{BC^2B^2}$, $\mathbf{B^2CB}$ $\}$
3	0.7437	$\{\mathbf{I}, \mathbf{ABA}$, $\mathbf{BACB^2AC^2B^2}$, $\mathbf{B^2AC^2BACB}$, $\mathbf{AB^2A}$,$\mathbf{ABCB^2A}$, $\mathbf{AB^2C^2BA}$, $\mathbf{B}$,$\mathbf{B^2}$, $\mathbf{BCB}$, $\mathbf{B^2C^2B^2}$, $\mathbf{AC}\}$
4	0.4585	$\{\mathbf{I}, \mathbf{B}, \mathbf{B^2}, \mathbf{BCB^2}, \mathbf{B^2C^2B}, \mathbf{BC^2B^2}, \mathbf{ABA}, \mathbf{AB^2A}$, $\mathbf{B^2CB}$, $\mathbf{AC}\}$
5	0.0000	$\{$ $\mathbf{I}, \mathbf{A}$, $\mathbf{B}$, $\mathbf{B^2}$, $\mathbf{C}$, $\mathbf{AC}$, $\mathbf{C^2}$, $\mathbf{AC^2}$ $\}$

of the elements of $\mathbf{G} \wr \mathbf{S}_n$ as $n \times n$ matrices of $k \times k$ matrices from $\mathbf{G}$, each member of $\mathbf{G} \wr \mathbf{S}_n$ would have one nonzero entry in each row and each column, and those nonzero entries would be members of $\mathbf{G}$.

The group $\mathbf{G} \wr \mathbf{S}_n$ acts on $\mathbb{C}^{kn}$, and we think of vectors in $\mathbb{C}^{kn}$ as n-vectors of k-vectors.

If $\mathbf{G}$ is generated by $\mathbf{A}$ and $\mathbf{B}$, then $\mathbf{G} \wr \mathbf{S}_n$ is generated by $a_1, b_1, t_2, t_3, \ldots, t_n$ where

$$(a_1)_{ij} = \begin{cases} \mathbf{A} & \text{if } i = j = 1 \\ \mathbf{I} & \text{if } i = j \neq 1 \\ \mathbf{O} & \text{otherwise.} \end{cases}$$

The matrix b_1 is defined similarly, with $\mathbf{B}$ replacing $\mathbf{A}$. The block permutation matrices t_i for $2 \leq i \leq n$ are given by

$$(t_i)_{jk} = \begin{cases} \mathbf{I} & \text{if } j + 1 = k = i \\ \mathbf{I} & \text{if } i = j = k + 1 \\ \mathbf{I} & \text{if } j = k \neq i, i - 1 \\ \mathbf{O} & \text{otherwise.} \end{cases}$$

Thus, for example, when $n = 3$ the generators are

$$a_1 = \begin{bmatrix} \mathbf{A} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix} \qquad b_1 = \begin{bmatrix} \mathbf{B} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix}$$

$$t_2 = \begin{bmatrix} \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix} \qquad t_3 = \begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} \end{bmatrix}.$$

In decoding, one also uses the block diagonal matrices $a_2, \ldots, a_n, b_2, \ldots, b_n$ defined analogously, but these are not needed as generators. Note that the size of $\mathbf{G} \wr \mathbf{S}_n$ is $|\mathbf{G}|^n n!$, which is considerably larger than either group code, $\mathbf{G}$ or $\mathbf{S}_n$, individually.

6. Initial vector

The initial vector for $\mathbf{G} \wr \mathbf{S}_n$ can be regarded as an n-vector of k-vectors for a k-dimensional group $\mathbf{G}$, where $k = 2$ in our examples. Let $\mathbf{v}_0$ denote the initial vector for $\mathbf{G}$. Mimicking the form of the initial vector for $\mathbf{G}(r, 1, n)$ in [5], let

$$\mathbf{x}_0 = \langle a\mathbf{v}_0, (a+b)\mathbf{v}_0, (a+2b)\mathbf{v}_0, \ldots, (a+(n-1)b)\mathbf{v}_0 \rangle$$

where the ratio b/a of the positive real numbers a and b is chosen (as below) to minimize the decoding error.

In a wreath product code based on $\mathbf{G} \wr \mathbf{S}_n$ with an initial vector $\mathbf{x}_0$ of the given form, the parameters a, $b > 0$ are at our disposal to minimize the error due to noise. For a normalized system with $\|\mathbf{x}_0\| = \|\mathbf{v}_0\| = 1$, there is really only one parameter, say b/a. We adjust b/a so that the minimum distance of the permutation subgroup $\mathbf{S}_n$ acting on $\mathbf{x}_0$ is equal to the minimum distance of the innermost copy of $\mathbf{G}$, acting on the first projection. Thus, if the normalized minimum distance of $\mathbf{G}$ acting on $\mathbf{v}_0$ is p, we set $\sqrt{2}b = ap$, so that $b/a = p/\sqrt{2}$.

7. Encoding

Let $\mathbf{H}_i$ denote the subgroup of $\mathbf{G} \wr \mathbf{S}_n$ consisting of those matrices that act on the ith coordinate only. Thus, for example, $\mathbf{H}_1$ is all block diagonal matrices $\mathrm{diag}(\mathbf{M}, \mathbf{I}, \ldots, \mathbf{I})$ with $M \in \mathbf{G}$. Use $\mathbf{S}_n$ to denote the subgroup of block permutation matrices. Then every element of $\mathbf{G} \wr \mathbf{S}_n$ can then be written uniquely as $g = \pi h_n \ldots h_1$ with $\pi \in \mathbf{S}_n$ and each $h_i \in \mathbf{H}_i$. Variations of this canonical form are possible, and indeed other forms were used in [5], [9] and our simulations, but this order is easy to describe and has good error control properties; see [5].

Some canonical form for the permutation π as a product of transpositions should also be chosen. Two different insertion sorts are suggested in Fossorier, Nation and Peterson [9], and are used in [5]. One uses the adjacent transpositions t_i $(2 \leq i \leq n)$ mentioned above, while the second uses a larger set of transpositions for more efficient decoding. Both yield a canonical form for permutations.

As usual, the vector $g^{-1}\mathbf{x}_0 = h_1^{-1} \ldots h_n^{-1} \pi^{-1} \mathbf{x}_0$ is transmitted through a noisy channel, and some corrupted vector $\mathbf{r}$ is received.

8. Decoding

To decode $\mathbf{r}$, we reverse the process. Let $\mathbf{r} = \langle \mathbf{u}_1, \ldots, \mathbf{u}_n \rangle$ and $\mathbf{x}_0 = \langle \mathbf{x}_{01}, \ldots, \mathbf{x}_{0n} \rangle$. First, using for example the Snowflake algorithm, find transformations $h'_1, \ldots, h'_n$ with $h'_i \in \mathbf{H}_i$ to minimize each $\|h'_i \mathbf{u}_i - \mathbf{x}_{0i}\|$. Applying these, we obtain $h'_n \ldots h'_1 \mathbf{r} = \mathbf{s} = \langle \mathbf{s}_1, \ldots, \mathbf{s}_n \rangle$.

Now find a permutation π' that sorts $\mathbf{s}$ according to size. That is, because $\mathbf{v}_0$ has the property $\|x_{01}\| < \|x_{02}\| < \cdots < \|\mathbf{x}_{0n}\|$, we find π' such that $\pi' \mathbf{s} = \mathbf{t} = \langle \mathbf{t}_1, \ldots, \mathbf{t}_n \rangle$ has $\|t_1\| < \|t_2\| < \cdots < \|\mathbf{t}_n\|$. This can be done in any number of ways, e.g., the insertion sorts suggested in [9]. Wadayama and Hagiwara suggest using linear programming methods for permutation codes [10].

Finally, decode the received vector as $g' = \pi' h'_n \ldots h'_1$.

9. Analysis of decoding complexity.

As above, one can measure the complexity of a decoding algorithm by the average number of comparisons, γ, for decoding, and the number of comparisons per bit, $\delta = \frac{\gamma}{\log_2 |\mathbf{G}|}$. For a general sorting algorithm, the theoretical lower bound for δ is 1 [14]. By utilizing the group structure, the Snowflake algorithm is able to produce group codes with δ less than 1. To get an idea of how the Snowflake algorithm affects δ, we will compare values for some complex reflection group codes using several schemes. In addition, δ values for several similar groups using various decoding methods are presented.

9.1. **Improvement in complex reflection group codes.** Table 5 lists the dimension and order for the group, along with the calculations γ and δ using three different decoding schemes. Scheme 1 uses the entire set X for each step, choosing c_{k+1} which minimizes $\|c_{k+1} r_k - \mathbf{x}_0\|$. Scheme 2 uses the entire set X for each step, but uses the distances as described in section 3.1 to choose the first $c_{k+1} \in X$ such that $\|c_{k+1} r_k - \mathbf{x}_0\| \leq d_{k+1}$, using a fixed ordering on X. Scheme 3 uses the Snowflake algorithm in its entirety as detailed in sections 3.1 and 3.2, choosing c_{k+1} from X_k.

Table 5. Efficiency Data

			Scheme 1		Scheme 2		Snowflake	
Group	Dim.	Order	γ	δ	γ	δ	γ	δ
$\mathbf{G}_4$	2	24	9.38	2.04	5.83	1.27	4.21	0.92
$\mathbf{G}_5$	2	72	20.78	3.37	12.08	1.96	5.76	0.93
$\mathbf{G}_8$	2	96	24.66	3.74	11.13	1.69	6.19	0.94
$\mathbf{G}_{16}$	2	600	95.14	10.31	55.07	6.0	16.8	1.82
$\mathbf{G}_{20}$	2	360	63.62	7.49	47.0	5.53	8.86	1.04
$\mathbf{G}_{26}$	3	1296	77.08	7.45	47.2	4.56	15.3	1.48

TABLE 6

Group	Decoding method	$\lvert\mathbf{G}\rvert$	δ
S_8	insertion sort	40,320	1.26
S_8	modified insertion	40,320	1.04
S_{16}	insertion sort	16!	1.64
S_{16}	modified insertion	16!	1.17
S_{32}	insertion sort	32!	2.35
S_{32}	modified insertion	32!	1.47
$\mathbf{G}(4,1,4)$	modified insertion	$4^4 \cdot 4!$	0.69
$\mathbf{G}(4,1,16)$	modified insertion	$4^{16} \cdot 16!$	0.89
$\mathbf{G}(8,1,4)$	modified insertion	$8^4 \cdot 4!$	0.52
$\mathbf{G}(8,1,16)$	modified insertion	$8^{16} \cdot 16!$	0.73
$\mathbf{G}_4$	Snowflake	24	0.92
$\mathbf{G}_5$	Snowflake	72	0.93
$\mathbf{G}_8$	Snowflake	96	0.94
$\mathbf{G}_{16}$	Snowflake	600	1.82
$\mathbf{G}_{20}$	Snowflake	360	1.04
$\mathbf{G}_{26}$	Snowflake	1296	1.48
$\mathbf{G}_8 \wr S_8$	Snowflake and mod. insert.	$96^8 \cdot 8!$	0.96
$\mathbf{G}_{20} \wr S_{16}$	Snowflake and mod. insert.	$360^{16} \cdot 16!$	1.07

As the table shows, the number of comparisons per bit is greatly reduced using the Snowflake algorithm. From scheme 1 to the Snowflake algorithm there is a 55% to 82% decrease in δ, depending upon the group. Even between scheme 2, which uses a portion of the Snowflake scheme, compared to the entire Snowflake algorithm we get a reduction in δ of 28% to 81%. Through manipulation of the algorithm choices of the sets X_i by hand, it is possible to make slight improvements for some of the groups. For example, for $\mathbf{G}_4$ we can get $\delta = 0.81$, and for $\mathbf{G}_8$, $\delta = 0.85$. However, this is a tedious process even with small groups for only a minor decrease in δ as compared to the vast improvement from applying the Snowflake algorithm.

9.2. **Comparison of similar groups.** Table 6 lists a group $\mathbf{G}$, the decoding method used, the order of $\mathbf{G}$, and the value δ. The modified insertion sort is described by Fossorier *et al.* [9] and produces a lower δ value than the usual insertion sort.

The codes using groups $\mathbf{G}(r,1,n)$ are taken from Kim, Nation and Shepler [5]. These are also wreath product codes, with $\mathbf{G}(r,1,n) \cong \mathbf{C}_r \wr \mathbf{S}_n$. The cyclic structure of the inner group allows for one-step decoding, making the codes very efficient in that regard. The methods used to optimize these codes could be applied to $\mathbf{G} \wr \mathbf{S}_n$ to further improve these codes as well.

Consider the group codes using a group $\mathbf{G}$ of unitary matrices to form the wreath product $\mathbf{G} \wr \mathbf{S}_n$. The size of $\mathbf{G} \wr \mathbf{S}_n$ is $|\mathbf{G}|^n n!$, which is significantly larger than either group code, $\mathbf{G}$ or $\mathbf{S}_n$, individually, though when combined,

the new code has only a slight increase in δ. The last two groups in Table 6 are examples of such wreath products, and are decoded using a combination of the Snowflake algorithm and the modified insertion sort. As you can see in the table, several of the group codes using the Snowflake algorithm, including $\mathbf{G}_8 \wr \mathbf{S}_8$, achieve $\delta < 1$, whereas the insertion sort must have $\delta \geq 1$. This makes the complex reflection group codes a competitive alternative to permutation codes which use an insertion sort.

For some reflection groups there is an alternative decoding method outlined by Hagiwara and Wadayama [10] and further discussed by Hagiwara and Kong [11] using LP-decoding of permutation codes. These codes do not produce a canonical form like the Snowflake algorithm does, but they are otherwise quite efficient. A δ value has not yet been calculated for the LP-decoding of permutation codes.

Though it is often a prominent topic in code discussions, speed and low complexity may not be the only desired attribute of a coding scheme. For example, in cryptography other code properties may be more desirable than speed. Memory storage may require more stability than efficiency as well, so different codes may be utilized. For example, Jiang *et al.* [12, 13] describe an algorithm utilizing permutations for use with flash memory cells. While their algorithm has benefits such as a canonical form, the use of an insertion sort leaves room for improvement in efficiency. See Barg and Mazumdar [1] for further analysis of permutation codes for this type of application.

10. Future Research

This coding scheme may be made more efficient and robust by utilizing a subset of the group rather than the entire group to make the code. Particular codewords which require more steps and/or comparisons to decode could be omitted entirely to decrease the decoding complexity and possibly allow for a higher noise tolerance. Variations in the generating set X may also improve the code and lead to lower δ values.

The Snowflake decoding algorithm can be used for any unitary group, thus further research could also include finding specific codes for other groups. Some unitary groups, like the Mathieu groups, have thus far been unused for coding. With this algorithm they may be used to make efficient group codes. Other group codes could also be concatenated to form larger codes with little change in decoding complexity.

The modified insertion sort lowers the decoding complexity for permutation codes. Applying the Snowflake algorithm to S_n may produce a further improved insertion sort with an even lower δ value. The algorithm described in this paper is so versatile that it could potentially be used in a variety of capacities, all of which should be explored.

References

[1] A. Barg and A. Mazumdar, *Codes in permutations and error correction for rank modulation*, IEEE Trans. on Information Theory 56 (2010), 6273–6293.

[2] A.S. Householder, The Theory of Matrices in Numerical Analysis, Blaisdell, New York, 1964.
[3] A.S. Householder, Lectures on Numerical Algebra, MAA, 1972.
[4] H.J. Kim, Decoding Complex Reflection Groups, Master's project, University of Hawaii, 2011.
[5] H.J. Kim, J.B. Nation and A. Shepler, *Group coding with complex reflection groups*, IEEE Trans. on Information Theory 61 (2015), 1–18.
[6] G.C. Shephard and J.A. Todd, *Finite unitary reflection groups*, Canad. J. Math., (1954), 274–304.
[7] D. Slepian, *Permutation modulation*, Proc. IEEE, **53** (1965), 228–236.
[8] D. Slepian, *Group codes for the Gaussian channel*, Bell Syst. Tech. J., **47** (1968), 575–602.
[9] M. Fossorier, J.B. Nation, and W. Peterson, *Reflection group codes and their decoding*, IEEE Trans. on Information Theory 56 (2010), 6273–6293.
[10] M. Hagiwara and T. Wadayama, *LP decodable permutation codes based on linearly constrained permutation matrices*, Proceedings ISIT (2011), 139–143.
[11] M. Hagiwara and J. Kong, *Comparing Euclidean, Kendall tau metrics toward extending LP decoding*, Proceedings ISITA (2012), 91–95.
[12] A. Jiang, R. Mateescu, M. Schwartz, and J. Bruck, *Rank modulation for flash memory*, Proceedings IEEE ISIT (2008), 1731–1735.
[13] A. Jiang, M. Schwartz, and J. Bruck, *Error correcting codes for rank modulation*, Proceedings IEEE ISIT (2008), 1736–1740.
[14] D. Knuth, *Searching and Sorting, the Art of Computer Programming*, vol. 3, Reading, MA, Addison-Wesley, 1973.
[15] C. Walker, The Snowflake Decoding Algorithm, Master's project, University of Hawaii, 2012.

Department of Mathematics, University of Hawaii, Honolulu, HI 96822, USA
E-mail address: jb@math.hawaii.edu

Department of Mathematics, Leeward Community College, Pearl City, HI 96782, USA
E-mail address: walkercl@hawaii.edu