Soft Decoding Techniques for Codes and Lattices, Including the Golay Code and the Leech Lattice[*]

J. H. Conway
Department of Pure Mathematics and Mathematical Statistics
University of Cambridge
Cambridge CB2 1SB, England

N. J. A. Sloane
Mathematical Sciences Research Center
AT&T Bell Laboratories
Murray Hill, NJ 07974, USA

ABSTRACT

This paper considers two kinds of algorithms. (i) If # is a binary code of length n, a ‘‘soft decision’’ decoding algorithm for # changes an arbitrary point of $\mathbf{R}^n$ into a nearest codeword (nearest in Euclidean distance). (ii) Similarly a decoding algorithm for a lattice Λ in $\mathbf{R}^n$ changes an arbitrary point of $\mathbf{R}^n$ into a closest lattice point. Some general methods are given for constructing such algorithms, and are used to obtain new and faster decoding algorithms for the Gosset lattice E_8, the Golay code and the Leech lattice.

[*] This paper appeared in ‘‘IEEE Trans. Information Theory’’, vol. 32 (1986), 41–50.

I. Introduction

Let # be an $[n,k]$ binary code. We regard the codewords as points of n-dimensional Euclidean space $\mathbf{R}^n$, and wish to find a ‘‘soft decision’’ decoder for # (also called an ‘‘analog’’ or ‘‘maximum likelihood’’ decoder). By this we mean an algorithm which, when presented with an arbitrary point x of $\mathbf{R}^n$, will find a codeword $u \in$ # that minimizes dist(x,u), the distance being Euclidean distance. Soft decision decoding has been investigated by many distinguished information theorists over the years — see [1]-[12], [14], [31]-[35], [40]-[45], [51], [52], [54], [58], [60], [62], [66]-[69]. However, the majority of these papers study decoding algorithms that only perform correctly *most* of the time. For example, Hackett’s decoding algorithm [42] for the Golay code is ‘‘only a few tenths of a decibel different from ideal correlation detection’’. In the present paper we are only interested in algorithms that *always* find a closest codeword or lattice point.

The decoding problem for a lattice Λ in $\mathbf{R}^n$ is similar. We wish to find an algorithm which, when presented with an arbitrary point of $\mathbf{R}^n$, will find a lattice point $u \in \Lambda$ that minimizes dist(x,u). Decoding algorithms for several classes of lattices were given in [22], [28].

In Section II we collect together all the methods we know for constructing decoding algorithms (of the type just mentioned, that always give the correct answer) for codes and lattices. Most of these methods were known already, although some are new. We then apply these methods to obtain improved decoding algorithms for the E_8 lattice (Section 2.13), the Golay code (Section III) and the Leech lattice (Section IV).

Potential applications of these algorithms are to channel coding and vector quantizing (see the references already mentioned, and [12], [13], [21], [27], [36], [37], [64]). It is worth mentioning that there is already a considerable literature devoted to ‘‘hard decision’’ or conventional binary decoding of the Golay code ([50, Chapter 16, § 9], [5], [31], [39], [70], [71]).

Notation. Two codes or lattices ! and @ are *geometrically similar* if one can be obtained from the other by (possibly) a translation, rotation, reflection and change of scale. The *direct sum* [50, p. 76] of two codes or lattices ! and @ is written $! \oplus @$. The *componentwise product* of two vectors *u* and *v* is written $u * v$.

II. Fast Decoding Algorithms

We first consider codes. Let # be an $[n,k]$ binary code. It is often more convenient to write the codewords as vectors of +1’s and −1’s rather than 0’s and 1’s. Note that if

$$w = u + v \qquad \text{in } 0,1 \text{ notation} \tag{1}$$

then

$$w = u * v \qquad \text{in } +1,\ -1 \text{ notation} . \tag{2}$$

The +1, −1 notation allows us to replace distance calculations with inner product calculations. For, if $x \ \varepsilon\ \mathbf{R}^n$ and $u \ \varepsilon\ \#$,

$$\begin{aligned} \operatorname{dist}^2(x,u) &= (x-u) \cdot (x-u) \\ &= x \cdot x - 2x \cdot u + u \cdot u \\ &= x \cdot x - 2x \cdot u + n . \end{aligned} \tag{3}$$

Thus finding a closest codeword to *x* is equivalent to finding a codeword (in +1, −1 notation) that has the largest inner product with *x*.

The following codes, and those geometrically similar to them, are most of the codes we know that have a fast ‘‘soft decision’’ decoding algorithm as described in Section I. We give rough estimates[(1)] of the number of arithmetic steps (additions, multiplications, etc.) required. From (2.2) onwards, ‘‘fast’’ means that the algorithm is substantially faster than the direct search method used in (2.1).

(2.1) **Any small code** may be decoded by a direct search. For an $[n,k]$ code we compute the inner product of the given vector x with every codeword and choose the closest. Assuming we have precomputed a list of the codewords, this requires roughly $2n \cdot 2^k$ steps, and is therefore only applicable to small codes. If the code contains the vector $(-1, -1, \ldots, -1)$ — i.e. the all 1’s vector in 0,1 notation — the codewords come in pairs $+u$ and $-u$, and the number of steps drops to $n2^k$.

(2.2) **A first-order Reed-Muller code**, with parameters $[n = 2^m, k = m+1]$, may be efficiently decoded using the fast Hadamard transform (the so-called Green machine described in [40], [41], [58], [50, Chap. 14]). This computes the 2^{m+1} inner products $x \cdot u,\ u \in \#$, in about $m2^m$ steps. It is worth pointing out that a first-order Reed-Muller code is geometrically similar to an octahedron (β_n in Coxeter’s notation [30]). For example, the codewords of the code of length 4 shown in Fig. 1a when multiplied by the orthogonal matrix of Fig. 1b become the 8 vertices

$$(\pm 2,0,0,0)\ , \ldots ,\ (0,0,0,\pm 2)$$

of a 4-dimensional octahedron.

(1) The footnotes are on page 30.

A $[2^m - 1, m]$ *simplex code* [50, p. 30] may be decoded by a straightforward modification of the fast Hadamard transform. (Set one of the inputs, say the last, to zero, and only calculate the linear combinations in which the last variable occurs with a + sign.) This code is geometrically similar to a simplex (α_n in the notation of [30]).

(2.3) **The universe code** ^ $_n$, consisting of all binary vectors of length n, may be decoded in just n steps. To decode $x = (x_1, \ldots, x_n)$, we simply replace each x_i by $\mathrm{sgn}(x_i)$, where

$$\begin{aligned} \mathrm{sgn}(x) &= +1 \text{ if } x \geq 0, \\ &= -1 \text{ if } x < 0. \end{aligned}$$

This is more efficient even than the fast Hadamard transform of (2.2), since it selects one out of 2^n codewords in n steps, whereas the fast Hadamard transform selects one of $2n$ codewords in $n \log_2 n$ steps. ^ $_n$ is geometrically similar to a cube (γ_n in the notation of [30]), the reciprocal polytope to the octahedron.

Of course not many interesting codes contain ^ $_n$ itself as a subcode. But codes geometrically similar to ^ $_n$ are quite common. For example, the Golay code &$_{24}$ (see Section III) contains a subcode of dimension 3 generated by

$$\begin{bmatrix} 11111111 & 00000000 & 00000000 \\ 00000000 & 11111111 & 00000000 \\ 00000000 & 00000000 & 11111111 \end{bmatrix} \qquad (4)$$

in 0,1 notation, i.e. by

$$-1^8 \; +1^{16}, \; +1^8 \; -1^8 \; +1^8, \; +1^{16} \; -1^8 \qquad \text{in } \pm 1 \text{ notation.}$$

This subcode is geometrically similar to ^ $_3$, and may be decoded as follows. Given

$x = (x_1 , \ldots , x_{24})$, we decode it as

$$u = aa \cdots a\ bb \cdots b\ cc \cdots c , \tag{5}$$

where a, b and c each appear 8 times, and

$$a = \operatorname{sgn}(x_1 + \ldots + x_8) ,$$

$$b = \operatorname{sgn}(x_9 + \ldots + x_{16}) ,$$

$$c = \operatorname{sgn}(x_{17} + \ldots + x_{24}) .$$

Furthermore, we see from (5) that the maximal inner product $x \cdot u$ is given by

$$\Big| \sum_{i=1}^{8} x_i \Big| + \Big| \sum_{i=9}^{16} x_i \Big| + \Big| \sum_{i=17}^{24} x_i \Big| . \tag{6}$$

A slightly more general family of codes can be decoded in the same way. Let T_a denote the $[a, 1]$ repetition code of length a. Then

$$T_{a_1} \oplus T_{a_2} \oplus \cdots \oplus T_{a_n} ,$$

although in general not geometrically similar to ^ $_n$, can be decoded by an obvious modification of the preceding algorithm.

(2.4) **The even weight code** %$_n$, with parameters $[n,\ k = n-1]$, consists of all 0,1 vectors with an even number of 1's, and may be decoded in about $2n$ steps. To decode $x = (x_1 , \ldots , x_n)$, we first replace each x_i by $\operatorname{sgn}(x_i)$. If there are an even number of minus signs we stop, but if there are an odd number we reverse the sign on an x_i of smallest magnitude. %$_n$ is geometrically similar to a hemi-cube ($h\gamma_n$ in the notation of [30, p. 155]).

Example 1. The Golay code contains a [24,5] subcode with generator matrix

$$\begin{bmatrix} 1111 & 1111 & 0000 & 0000 & 0000 & 0000 \\ 1111 & 0000 & 1111 & 0000 & 0000 & 0000 \\ 1111 & 0000 & 0000 & 1111 & 0000 & 0000 \\ 1111 & 0000 & 0000 & 0000 & 1111 & 0000 \\ 1111 & 0000 & 0000 & 0000 & 0000 & 1111 \end{bmatrix} , \tag{7}$$

the *sextet code* (cf. [15]), which is geometrically similar to $\%_5$. Since this code plays a key role in Section III, we give the precise decoding algorithm. To decode $x = (x_1 , \ldots , x_{24})$ we first compute

$$s_{\mathrm{I}} := \sum_{i=1}^{4} x_i , \; s_{\mathrm{II}} := \sum_{i=5}^{8} x_i , \ldots , s_{\mathrm{VI}} := \sum_{i=21}^{24} x_i , \tag{8}$$

and

$$u_{\mathrm{I}} := \operatorname{sgn}(s_{\mathrm{I}}) , \ldots , u_{\mathrm{VI}} := \operatorname{sgn}(s_{\mathrm{VI}}) . \tag{9}$$

If an odd number of the u_N are negative we change the sign of a u_N for which $|s_N|$ is minimal. Then x is decoded as

$$u := u_{\mathrm{I}} u_{\mathrm{I}} u_{\mathrm{I}} u_{\mathrm{I}} \; u_{\mathrm{II}} u_{\mathrm{II}} u_{\mathrm{II}} u_{\mathrm{II}} \; \cdots \; u_{\mathrm{VI}} u_{\mathrm{VI}} u_{\mathrm{VI}} u_{\mathrm{VI}} , \tag{10}$$

and the maximal inner product is given by

$$x \cdot u = |s_{\mathrm{I}}| + \ldots + |s_{\mathrm{VI}}| \tag{11a}$$

if an even number of the u_N in (9) are negative, otherwise by

$$x \cdot u = |s_{\mathrm{I}}| + \ldots + |s_{\mathrm{VI}}| - 2 \min_N |s_N| . \tag{11b}$$

Example 2. The $[2n, n-1]$ code d_n generated by 111100..., 0011110..., 000011110... , $\cdots$ [57, p. 320] is geometrically similar to $\%_n$.

Remark. *Permutation codes* [6], [7], [62] are a class of (in general) non-binary codes that include first-order Reed-Muller codes, simplex codes, ^ $_n$ and %$_n$ as special cases, and may also be decoded rapidly.

(2.5) **Codes with small redundancy** may be regarded as trellis codes, as pointed out by Solomon [11], [66] and Wolf [69], and therefore can be efficiently decoded by the Viterbi algorithm [35], [68]. For an $[n,k]$ code, the trellis has 2^{n-k} states, and the number of decoding steps is roughly $4n2^{n-k}$.

(2.6) **Direct sums** of codes on this list can also be decoded easily. To decode ! $\oplus$ @, for example, we apply the decoder for ! to the first part of x and the decoder for @ to the second part. The number of steps is the sum of the numbers of steps for decoding ! and @.

(2.7) **Supercodes**. If @ is one of the above codes, and # contains @ as a subcode of small index, then # can also be decoded easily. Let us write

$$\# = \bigcup_{j=0}^{t-1} (\rho^{(j)} + @) \tag{12}$$

where $t := |\#|/|@|$ is the *index* of @ in #, and the $\rho^{(j)}$ $(j = 0, \ldots, t-1)$ are *coset representatives* for @ in #. # is called a *supercode* of @. Eq. (12) applies to 0,1 vectors, while in terms of +1, −1 vectors we have

$$\# = \bigcup_{j=0}^{t-1} \rho^{(j)} * @ \tag{13}$$

(compare (1), (2) above). Finding the largest inner product of x with the vectors in $\rho^{(j)} * @$ is equivalent to finding the largest inner product of $\rho^{(j)} * x$ with the vectors in

@. So provided t is not too large, we may decode x as follows.

Compute $y^{(j)} := \rho^{(j)} * x$, and (14)

Use the decoder for @ to find $u^{(j)} :=$ the closest vector in @ to $y^{(j)}$ and $ip^{(j)} := u^{(j)} \cdot y^{(j)}$.

After doing this for $j = 0, 1, \ldots, t-1$, the largest $ip^{(j)}$ is the final inner product, and the corresponding $u^{(j)}$ is used to produce the decoder output, $\rho^{(j)} * u^{(j)}$.

This is similar to Algorithm 3 of [22]. The number of steps required is about t times the number of steps to decode @, plus $2nt$ to compute the $y^{(j)}$ in (14). In practice this number $2nt$ can often be greatly reduced, as we shall see in Section III.

Example. Codes formed by ‘‘gluing’’ several subcodes together can be decoded in this way. Figure 2 shows a generator matrix for a typical code of this type, consisting of a direct sum of subcodes $@_1, @_2, \ldots,$ together with certain additional generators called *glue vectors*. If there are g additional generators, the index of the subcode $@_1 \oplus @_2 \oplus \cdots$ in the full code is $t = 2^g$. Furthermore the subcode generated by these g glue vectors (the *glue code*) consists of precisely the coset representatives $\rho^{(j)}$ appearing in (12). Numerous examples of such codes may be found in [18], [56], [57].

(2.8) **Shortened codes**. If an $[n,k]$ code # can be decoded efficiently, so can the $[n-1,k]$ code @ obtained by deleting the last coordinate of every codeword. To decode $x = (x_1, \ldots, x_{n-1})$, we simply apply the decoder for # to $(x_1, \ldots, x_{n-1}, 0)$. Since the codewords in # end with +1 or −1, this will find the closest codeword to x.

Next we consider lattices. Given a lattice Λ in $\mathbf{R}^n$, the decoding algorithm changes an arbitrary point of $\mathbf{R}^n$ into a closest lattice point. (We can no longer work with inner products.) The following lattices, and those geometrically similar to them, are most of the lattices we know that have a fast decoding algorithm.

(2.9) **The cubic lattice $\mathbf{Z}^n$**, consisting of all points $(u_1, \ldots, u_n)$ with integer coordinates, can be decoded in about n steps. If x_i is a real number we define

$$f(x_i) := \text{nearest integer to } x_i , \tag{15}$$

rounding towards zero in case of a tie, and for a vector $x = (x_1, \ldots, x_n)$ we define

$$f(x) = (f(x_1), \ldots, f(x_n)) . \tag{16}$$

Then the decoder for $\mathbf{Z}^n$ simply changes x to $f(x)$ [22, Sect. III].

(2.10) **The lattice D_n**, consisting of all points in $\mathbf{Z}^n$ whose coordinates add to an even number, can be decoded in about $4n$ steps. For $x \in \mathbf{Z}^n$, we define $g(x)$ to be the same as $f(x)$, except that the *worst* component of x — that furthest from an integer — is rounded the *wrong way*. In case of a tie, the component with the lowest subscript is rounded the wrong way. (Formal definitions of f and g may be found on page 228 of [22].)

The decoder for D_n computes $f(x)$ and $g(x)$, and the sum $f(x_1) + \ldots + f(x_n)$ of the components of $f(x)$. If the sum is even, the output is $f(x)$, and if it is odd the output is $g(x)$. (See [22] for a proof and an example.)

The number of steps may be estimated as follows. For each of the n components of x we compute (a) $f(x_i)$ and (b) the error $e_i = x_i - f(x_i)$, (c) test if e_i is a new worst error [needed to determine $g(x)$], and (d) update $\sum f(x_i)$. Thus the total number of steps is about $4n$.

(2.11) **The lattice** A_n, consisting of all points in $\mathbf{Z}^{n+1}$ whose coordinates add to 0, can be decoded in a constant times $n \log_2 n$ steps by the algorithm given in [22, Sect. VII]. As pointed out to us by A. M. Odlyzko this can be reduced to a constant times n by the following argument. If the discrepancy is Δ (see [22]), it is necessary to find the $|\Delta|$ largest (if $\Delta > 0$, or smallest, if $\Delta < 0$) of the numbers $\delta(x_i')$. This can be done in a constant times n steps using the Rivest-Tarjan algorithm (see [46]). For $n = 2$ and 3 there are better algorithms. A_2 is the familiar two-dimensional hexagonal lattice, and is best decoded using the fact that it is the union of a rectangular lattice and a translate, as suggested by Gersho ([37, p. 165], [28, p. 299]). A_3 is geometrically similar to the face-centered cubic lattice D_3, and is best decoded by the D_3 algorithm.

(2.12) **Lattices obtained from Construction A**. If # is an $[n,k]$ binary code, Construction A [48], [63] produces a lattice $\Lambda(\#)$ in $\mathbf{R}^n$. If the codewords of # are written in 0,1 notation, the points of $\Lambda(\#)$ consist of all vectors of the form

$$c + 2z, \qquad \text{for } c \in \#,\ z \in \mathbf{Z}^n , \tag{17}$$

where we regard the 0's and 1's in c as real numbers rather than elements of the Galois field $GF(2)$. For our present purposes we wish to write the codewords in +1, −1 notation, in which case the points of $\Lambda(\#)$ consist of all vectors of the form

$$c + 4z, \qquad \text{for } c \in \#,\ z \in \mathbf{Z}^n . \tag{18}$$

[The set of points (18) strictly speaking no longer forms a lattice, but rather is a translate of a lattice by the vector $(1, 1, \ldots, 1)$.]

The following lemma makes it possible to use a decoding algorithm for # to decode $\Lambda(\#)$.

Lemma. *Suppose* $x = (x_1, \ldots, x_n)$ *lies in the cube* $-1 \le x_i \le 1$ $(i = 1, \ldots, n)$. *Then no point of* $\Lambda(\#)$ *is closer to x than the closest codeword of* #.

Proof. Suppose the contrary, and let $u = (u_1, \ldots, u_n)$ be a closest lattice point to x. By hypothesis some u_i's are neither +1 nor –1. By subtracting a suitable vector $4z$, we may change these coordinates to +1 or –1 (depending on their parity) to produce a point of $\Lambda(\#)$ that is in #, and is at least as close to x as u is, a contradiction.

Decoding algorithm for $\Lambda(\#)$

(i) Given $x = (x_1, \ldots, x_n)$, we first reduce all x_i to the range $-1 \le x_i < 3$ by subtracting a vector $4z$.

(ii) Let S denote the set of i for which $1 < x_i < 3$. For $i \varepsilon S$, replace x_i by $2 - x_i$.

(iii) Since x is now in the cube $-1 \le x_i \le 1$ $(i = 1, \ldots, n)$, by the Lemma we are justified in applying the decoder for # to x, obtaining an output $c = (c_1, \ldots, c_n)$ say.

(iv) For $i \varepsilon S$ change c_i to $2 - c_i$. Then $c + 4z$ is a closest point of $\Lambda(\#)$ to the original vector x.

The number of steps may be estimated as follows. For each x_i we compute the nearest integer to $x_i - 1$ that is a multiple of 4, say $4z_i$, (19)

the difference $d_i = x_i - 4z_i$, and (20)

if $d_i > 2$, change d_i to $2 - d_i$ (for $i \varepsilon S$, say) . (21)

Then we apply the decoder for # to $(d_i, \ldots, d_n)$, producing $c = (c_1, \ldots, c_n)$. Finally we

change c_i to $2 - c_i$ for $i \varepsilon S$, and (22)

add $4z$ to c . (23)

The total number of steps is roughly $5n$ plus the number to decode #.

Many interesting sphere packings can be obtained from Construction A [29], [48], [63], the most important being the E_8 lattice — see (2.13).

Remark. Unfortunately it does not appear possible to modify this algorithm to apply to lattices obtained from Construction B [48], [63]. (Construction B differs from Construction A in that, in (17) and (18) $z = (z_1, \ldots, z_n)$ must satisfy $\sum z_i \equiv 0 \pmod 2$.) If such a modification were found, it would further speed up the decoding algorithm for the Leech lattice given in Section IV.

(2.13) **The Gosset lattice E_8.** This important lattice can be constructed in several ways (see [12], [21], [22], [24]-[26], [29], [48]), one of which is to apply Construction A (see (2.12)) to the [8,4] first-order Reed-Muller code #. In this form E_8 consists of the vectors

$$c + 4z, \qquad c \varepsilon \#, z \varepsilon \mathbf{Z}^8 , \tag{24}$$

c being a +1, −1 vector of length 8. [As mentioned above, with these coordinates E_8 is not a lattice but a translate of a lattice by the vector $(1, 1, \ldots, 1)$.] # can be decoded in about $3 \cdot 8 + 8 = 32$ steps by a fast Hadamard transform (see (2.2)). Therefore the algorithm given in (2.12) will decode this version of E_8 in roughly 72 steps. This is faster than the algorithm proposed in [22] (see (2.15) below) which requires about 104 steps.

If we require not only the closest point $u \in E_8$ to x, but also $\mathrm{dist}^2(x,u)$, this can be obtained at the end of step (iii) of the algorithm, using Eq. (3), at the cost of about 16 additional steps to compute $x \cdot x$. If steps (i) and (ii) can be carried out in advance, as will be the case when we use this algorithm in decoding the Leech lattice in Section IV, the number of steps to decode one x drops to about 48, or 56 if $\mathrm{dist}^2(x,u)$ is needed.

(2.14) **Direct sums** of lattices on this list can be handled in the same way as direct sums of codes — see (2.6).

(2.15) **Superlattices**. If M is one of the above lattices and Λ contains M as a sublattice of small index, then Λ can also be decoded easily. We write

$$\Lambda = \bigcup_{j=0}^{t-1} (\rho^{(j)} + \mathrm{M}) , \tag{25}$$

where $t := \det \mathrm{M} / \det \Lambda$ is the index of M in Λ, $\det \mathrm{M}$ is the volume of a fundamental domain for M (see [21], [63], [65]), and the $\rho^{(j)}$ $(j = 0, 1, \ldots, t-1)$ are coset representatives for M in Λ. Let Φ be a decoder for M.

To decode Λ we proceed as follows. Given x, we calculate

$$y^{(j)} := x - \rho^{(j)} , \tag{26}$$

$$z^{(j)} := \Phi (y^{(j)}) , \tag{27}$$

$$d_j := (z^{(j)} - y^{(j)}) \cdot (z^{(j)} - y^{(j)}) , \tag{28}$$

for $j = 0 , \ldots , t-1$, and find a j, j^* say, for which d_j is minimized. Then the decoder output is

$$u := z^{(j^*)} + \rho^{(j^*)} , \tag{29}$$

and d_{j^*} is the squared distance from x to u [22, Sect. V].

The number of steps required is about t times the number of steps to decode M, plus $n(t-1)$ steps to compute the $y^{(j)}$ (in practice this can often be greatly reduced — see Section IV), plus $3nt$ steps to compute the d_j.

Example 1. The E_8 lattice (see (2.13)) is also given by

$$E_8 = D_8 \cup (\tfrac12\tfrac12\tfrac12\tfrac12\tfrac12\tfrac12\tfrac12\tfrac12) + D_8 , \tag{30}$$

an example in which $t = 2$ and M $= D_8$. The above algorithm (proposed in [22]) therefore takes about $2 \cdot 32 + 8 + 32 = 104$ steps. This drops to about 96 if $y^{(0)}$ and $y^{(1)}$ have been precomputed.

Example 2. The dual lattices D_n^* and A_n^* (the so-called Voronoi lattice of the first type, which is important for the covering problem [61]) may also be decoded by this algorithm (see [22]).

Example 3. Lattices formed by ‘‘gluing’’ sublattices together (in exactly the same way that codes were glued together in (2.7)) may be decoded by this algorithm. This includes the twenty-three 24-dimensional Niemeier lattices [53], [19], as well as many other lattices described in [19], [20].

Based on the results assembled in this section, we can now state our general strategy for finding an efficient decoding algorithm for a code or lattice: find a subcode (or sublattice) of smallest index for which a fast algorithm exists, and then use (2.7), (2.12) or (2.15). Algorithms for decoding the lattices E_6, E_6^*, E_7, E_7^*, K_{12} and Λ_{16} were obtained in [28] by using this strategy. For example the Coxeter-Todd lattice K_{12} contains $A_2 \oplus \cdots \oplus A_2$ (6 summands) as a sublattice of index 64. In the next two sections we apply the strategy to the Golay code and the Leech lattice.

III. Fast decoding algorithms for the Golay code

The direct search algorithm (see (2.1)) for the [24,12] Golay code $\&_{24}$ takes about $24 \cdot 2^{12} = 98304$ steps, while the trellis decoding method of (2.5) is even slower. Hwang's algorithm [44], [45] for the shortened Golay code of length 23 takes roughly $24 \cdot 1376 = 33024$ steps. In this section we give two algorithms that are much faster. The first is based on the subcode (7), takes about 1584 steps, and is described in detail. The second is based on the subcode (4), takes about 1728 steps, and is only sketched. The second algorithm is included because it could easily be faster than the first if a different method were used to count steps.

(3.1) **The first algorithm**. There are three parts to the algorithm, the design stage, which is only performed once, and the precomputation and main stages, which are both performed each time a vector is decoded.

The design stage. $\&_{24}$ contains the [24,5] subcode @ shown in (7). As coset representatives for @ in $\&_{24}$ we take the 128 words of the code defined by the following generator matrix.

$$\begin{bmatrix} 1100 & 1100 & 1100 & 1100 & 0000 & 0000 \\ 1010 & 1010 & 1010 & 1010 & 0000 & 0000 \\ 1010 & 1001 & 1100 & 0000 & 1100 & 0000 \\ 1001 & 1100 & 1010 & 0000 & 1010 & 0000 \\ 0111 & 1000 & 1000 & 1000 & 1000 & 1000 \\ 0000 & 0000 & 1100 & 1100 & 1100 & 1100 \\ 0000 & 0000 & 1010 & 1010 & 1010 & 1010 \end{bmatrix} \qquad (31)$$

(7) and (31) together generate $\&_{24}$ [28, Fig. 6].

We begin by preparing a table of all 128 coset representatives, written in +1, −1 notation, labeling them $\rho^{(0)}, \ldots, \rho^{(127)}$ in some arbitrary order. The beginning of this table (using one obvious ordering) is shown in Table I.

For a typical coset representative

$$\rho^{(j)} = (\rho_1^j, \ldots, \rho_{24}^j)$$

the algorithm will involve the componentwise product

$$\rho^{(j)} * x = (\rho_1^j x_1, \ldots, \rho_{24}^j x_{24}) .$$

Inspection of (31) reveals that, in the first block of four coordinates of $\rho^{(j)} * x$, all possible sign combinations of x_1, x_2, x_3, x_4 occur. The precomputation stage of the algorithm will calculate these 16 combinations in the Gray code order shown in Table III. Similarly in the other five blocks of coordinates, except that in blocks III to VI, the final x_i's (i.e. x_{12}, x_{16}, x_{20}, x_{24}) only occur with + signs.

We now prepare a second table, the *cross-reference table*, as follows. For each entry $\rho^{(j)}$ in Table I, we write down for each block of four coordinates where that sign combination is to be found in Table III. For $\rho^{(4)}$, for example, the sign combinations on the six blocks are

$$-+-+ \qquad -++- \qquad --++ \qquad ++++ \qquad --++ \qquad ++++$$

which are entries

$$6 \quad 14 \quad 2 \quad 0 \quad 2 \quad 0$$

respectively of Table III. This gives the fifth line

$$\chi_{\mathrm{I}}^{4} = 6,\ \chi_{\mathrm{II}}^{4} = 14,\ \chi_{\mathrm{III}}^{4} = 2\ ,\ldots,\ \chi_{\mathrm{VI}}^{4} = 0$$

of Table II.

The decoding algorithm

To decode $x = (x_1\ ,\ldots,\ x_{24})$.

Precomputation stage

We compute six *Gray code* tables, the first of which is shown in Table III. Each entry differs from the previous one in only one coordinate (since the entries are ordered by a Gray code [38], [55], [59]), and so can be computed with just one subtraction. Note also that the entries in the second half of Table III are simply the negatives of the entries in the first half.

Table III is the Gray code table for block I, and involves $x_1\ ,\ldots,\ x_4$. The other five tables correspond to blocks II ,..., VI, and involve $x_5\ ,\ldots,\ x_8\ ; \cdots ;\ x_{21}\ ,\ldots,\ x_{24}$ respectively, but have the same sign combinations. However the last eight rows of the tables for blocks III to VI can be omitted, since x_{12}, x_{16}, x_{20} and x_{24} only occur with + signs. We let $G_N(i)$ denote the *i*-th entry in the table for block N $(N = \mathrm{I}\ ,\ldots,\ \mathrm{VI})$.

Main stage

Set record = 0 and $j^* = 0$.

```
For $j = 0$ through 127,
    obtain $\chi_{\mathrm{I}}^j, \ldots, \chi_{\mathrm{VI}}^j$ from Table II,
    obtain $G_{\mathrm{I}}(\chi_{\mathrm{I}}^j), \ldots, G_{\mathrm{VI}}(\chi_{\mathrm{VI}}^j)$ from the Gray code tables, and
    compute the inner product
        $ip = |G_{\mathrm{I}}(\chi_{\mathrm{I}}^j)| + \ldots + |G_{\mathrm{VI}}(\chi_{\mathrm{VI}}^j)|$        (32a)
    if an even number of the $|G_N(\chi_N^j)|$ are even, or
        $ip = |G_{\mathrm{I}}(\chi_{\mathrm{I}}^j)| + \cdots + |G_{\mathrm{VI}}(\chi_{\mathrm{VI}}^j)| - 2 \min_N |G_N(\chi_N^j)|$ .        (32b)
    otherwise, and
        if $ip >$ record, set record $= ip$ and $j^* = j$ .
```

After the 127-th step, a closest codeword in the Golay code to x is

$$u := u_{\mathrm{I}} u_{\mathrm{I}} u_{\mathrm{I}} u_{\mathrm{I}} \; u_{\mathrm{II}} u_{\mathrm{II}} \; ..... \; u_{\mathrm{VI}} u_{\mathrm{VI}} u_{\mathrm{VI}} u_{\mathrm{VI}} \tag{33}$$

where

$$u_N = \mathrm{sgn}(G_N(\chi_N^{j^*})) \quad , \quad N = \mathrm{I}, \ldots, \mathrm{VI} ,$$

with the sign of the smallest $|G_N(\chi_N^{j^*})|$ reversed if an even number of the u_N are negative. ‘‘Record’’ gives the inner product $u \cdot x$.

The number of steps is roughly 48 for the precomputation stage, plus $12 \cdot 128 = 1536$ for the main stage, a total of 1584.

To see that this works, we remark that it is just the algorithm of (2.7), modified so as to precompute the $y^{(j)}$ (using the Gray code tables). We have also made use of the special structure of the subcode @ to streamline the calculation of the inner products $ip^{(j)}$ ((32) follows from (11), and (33) from (10)).

(3.2) **The second algorithm**

The overall structure of the second algorithm is the same, but now we use the [24,3] subcode @ shown in (4). The 512 coset representatives for @ in $\&_{24}$ are generated by the following matrix.

$$\begin{bmatrix}
1111 & 0000 & 1111 & 0000 & 0000 & 0000 \\
1100 & 1100 & 1100 & 1100 & 0000 & 0000 \\
1010 & 1010 & 1010 & 1010 & 0000 & 0000 \\
1010 & 1001 & 1100 & 0000 & 1100 & 0000 \\
1001 & 1100 & 1010 & 0000 & 1010 & 0000 \\
1100 & 1010 & 1001 & 0000 & 1001 & 0000 \\
0111 & 1000 & 1000 & 1000 & 1000 & 1000 \\
0000 & 0000 & 1100 & 1100 & 1100 & 1100 \\
0000 & 0000 & 1010 & 1010 & 1010 & 1010
\end{bmatrix} \tag{34}$$

(4) and (34) together generate $\&_{24}$. The precomputation stage computes three Gray code tables, the first containing all 128 combinations

$$\pm x_1 \pm \cdots \pm x_7 \pm x_8 \ ,$$

the second all 64 combinations

$$\pm x_9 \pm \cdots \pm x_{15} + x_{16} \ ,$$

and the third all 64 combinations

$$\pm x_{17} \pm \cdots \pm x_{23} + x_{24} ,$$

with an even number of minus signs in each case.

In the main stage j now runs from 0 to 511, and (32a) and (32b) are replaced by the much simpler formula

$$ip = | G_{\mathrm{I}} (\chi_{\mathrm{I}}^j) | + | G_{\mathrm{II}} (\chi_{\mathrm{II}}^j) | + | G_{\mathrm{III}} (\chi_{\mathrm{III}}^j) | , \tag{35}$$

obtained from (6). The final output is

$$u_{\mathrm{I}} \cdots u_{\mathrm{I}}\ u_{\mathrm{II}} \cdots u_{\mathrm{II}}\ u_{\mathrm{III}} \cdots u_{\mathrm{III}} , \tag{36}$$

where $u_N = \mathrm{sgn} (G_N (\chi^j))$, $N = \mathrm{I}, \mathrm{II}, \mathrm{III}$ (cf. (5)). The total number of steps is roughly 192 for the precomputation stage, plus $3 \cdot 512 = 1536$ for the main stage, a total of 1728.

(3.3) **The Golay code of length 23** can be decoded by a straightforward modification of either algorithm (or alternatively by using the algorithms as they stand and the method of (2.8)).

IV. A fast decoding algorithm for the Leech lattice

The Leech lattice Λ_{24} is a 24-dimensional lattice that can be defined in a number of ways [10], [15], [17], [23]-[25], [29], [47]-[49], [53], [64], [65]. For example it may be obtained by applying Construction B (see (2.12)) to the Golay code $\&_{24}$, and then doubling the number of points. In [28] we proposed a decoding algorithm for Λ_{24} based on the fact that Λ_{24} contains D_{24} as a sublattice of index 8192. In view of (2.10) and (2.15), this algorithm takes about

$$4 \cdot 24 \cdot 8192 = 786432$$

steps to decode one point. In contrast the algorithm given below takes only about 55968 steps, which is about 14 times as fast. This algorithm is based on the ‘‘Turyn construction’’ of Λ_{24}.

(4.1) **The ‘‘Turyn construction’’ of the Leech lattice**. R. J. Turyn showed around 1965 that the Golay code may be constructed by gluing together three copies of the [8,4] first-order Reed-Muller code (see [50, Chap. 18]). The Leech lattice may be constructed in a similar manner by gluing together three copies of the E_8 lattice. Although this construction has been known for many years, the following particularly simple version has not appeared in print before. We give two sets of coordinates, the first being more elegant, while the second is easier to decode.

Let Λ_8 denote the particular version of E_8 obtained by multiplying the vectors in (30) by 4. Typical vectors in Λ_8 are (0^8), $(\pm 4, \pm 4, 0^6)$, and $(\pm 2^8)$ with an even number of minus signs.

Definition 1. The Leech lattice Λ_{24} consists of the vectors

$$(e_1 + a + t,\ e_2 + b + t,\ e_3 + c + t)\ , \tag{37}$$

where

e_1, e_2, e_3 are arbitrary vectors of Λ_8,

a, b are arbitrary vectors from the list of 16 given in Table IVa,

c is the unique vector in Table IVa satisfying

$$a + b + c \equiv 0 \ (\mathrm{mod}\ \Lambda_8)\ , \tag{38}$$

t is an arbitrary vector from the list of 16 given in Table IVt.

To see that (37) does define the Leech lattice, we begin with the standard Miracle Octad Generator (or MOG) construction of Λ_{24} (see [16] or [25] for example), in which the 24 coordinates are divided into 3 sets of 8. The *intersection* of Λ_{24} with any one of these 8-dimensional spaces is our Λ_8, and the *projection* onto the same space is $\frac{1}{2}\Lambda_8$. The quotient $\frac{1}{2}\Lambda_8 / \Lambda_8$ is an abelian group of order 256, and the vectors $a + t$, $a \varepsilon$ Table IVa, $t \varepsilon$ Table IVt, are coset representatives for Λ_8 in $\frac{1}{2}\Lambda_8$. (The blocks of four coordinates in Tables IVa and IVt represent columns of the MOG. See also Fig. 27 of [16].) The quotient $\Lambda_{24} / (\Lambda_8 \oplus \Lambda_8 \oplus \Lambda_8)$ is an abelian group of order 4096, and the vectors

$$(a + t,\ b + t,\ c + t)\ , \tag{39}$$

with $a, b, c \varepsilon$ Table IVa, t ε Table IVt, and

$$a + b + c \equiv 0 \ (\mathrm{mod}\ \Lambda_8)$$

are coset representatives for $\Lambda_8 \oplus \Lambda_8 \oplus \Lambda_8$ in Λ_{24}.

Definition 1 is based on the version of E_8 constructed in (30). But, as we saw in (2.13), the version constructed in (24) is easier to decode. To convert from (30) to (24) we multiply by

$$\frac{1}{2}\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix} \tag{40}$$

and add $(1,1,\ldots,1)$. This leads to the second definition of Λ_{24}.

Let Λ'_8 denote the E_8 lattice in the form in which the 240 minimal vectors consist of 16 of the shape $(\pm 4, 0^7)$, and $2^4 \cdot 14 = 224$ obtained from

$$\begin{array}{c} (2222\ 0000) \\ (2200\ 2200) \\ (2200\ 0022) \\ (2020\ 2020) \\ (2020\ 0202) \\ (2002\ 2002) \\ (2002\ 0220) \end{array} \tag{41}$$

by inserting arbitrary signs, and by interchanging 0's and ± 2's. Also let $\Lambda''_8 = \mathbf{1} + \Lambda'_8$, where $\mathbf{1} = (1,1,\ldots,1)$. Then Λ''_8 is precisely the version of E_8 defined in (24). Typical vectors in Λ''_8 are

$$\begin{array}{c} (1111\ 1111)\,, \\ (\overline{1111}\ 1111)\,, \\ (1111\ \overline{1111})\,, \\ (\overline{11}11\ \overline{11}11)\,, \\ (\overline{3}111\ 1111)\,, \\ \cdots \end{array} \tag{42}$$

where the bar indicates a negative number.

Definition 2. The Leech lattice Λ_{24} consists of the vectors

$$\mathbf{1} + (e_1 + a + t,\ e_2 + b + t,\ e_3 + c + t)\ , \tag{43}$$

where

e_1, e_2, e_3 are arbitrary vectors of Λ'_8,

a, b are arbitrary vectors from Table Va,

c is the unique vector in Table Va satisfying

$$a + b + c \equiv 0 \pmod{\Lambda'_8}\ , \tag{44}$$

t is an arbitrary vector from Table Vt.

(4.2) **Decoding algorithm for the Leech lattice** Λ_{24}

We define Λ_{24} by (43). As in (3.1) there are three parts to the algorithm.

The design stage.

We begin by preparing a list of the 256 sums $a+t$, $a \epsilon$ Table Va, $t \epsilon$ Table Vt, labeling then $\rho^{(0)}, \ldots, \rho^{(255)}$ in some arbitrary order. The beginning of this table (using one obvious ordering) is shown in Table VI.

We now prepare a second table, the *cross-reference table*, as follows. For each of the 4096 vectors

$$(a+t,\ b+t,\ c+t)\ , \tag{45}$$

with $a,b,c \epsilon$ Table Va, $t \epsilon$ Table Vt, and satisfying (44), we write down a triple

$$\chi^j = (\chi_1^j, \chi_2^j, \chi_3^j)\quad,\quad 0 \le j \le 4095,$$

indicating that $a+t$ is entry χ_1^j of Table VI, $b+t$ is entry χ_2^j, and $c+t$ is entry χ_3^j. In other words

$$(\rho^{(\chi_1^j)}, \rho^{(\chi_2^j)}, \rho^{(\chi_3^j)})$$

is a triple (45) that satisfies (44). Part of the cross-reference table based on Table VI is shown in Table VII.

The decoding algorithm

To decode $x = (x_1, \ldots, x_{24})$.

Precomputation stage

In a moment we shall apply the E_8 decoder of (2.13) to the 256 vectors

$$(x_1, \ldots, x_8) + a + t,\ a \varepsilon \text{ Table Va},\ t \varepsilon \text{ Table Vt}\ .$$

Before doing this we carry out steps (19) to (21) of the decoder in advance. The components of the vectors $a+t$ range from -2 to $+4$. So our first step is to

compute the $7 \cdot 24 = 168$ numbers

$$y_{im} = x_i + m \qquad (-2 \leq m \leq 4),$$

find the nearest integer to $y_{im} - 1$ that is a multiple of 4, $4z_{im}$ say,

find the difference $d_{im} = y_{im} - 4z_{im}$,

if $d_{im} > 2$, change d_{im} to $2 - d_{im}$, keeping a record of this change, and

calculate d_{im}^2.

The second precomputation step is the most time-consuming part of the algorithm. For $j = 0$ through 255 we calculate

$$(x_1, \ldots, x_8) + \rho^{(j)}, \tag{46}$$

$$(x_9, \ldots, x_{16}) + \rho^{(j)}, \tag{47}$$

$$(x_{17}, \ldots, x_{24}) + \rho^{(j)}, \tag{48}$$

and apply the E_8 decoder of (2.13) to these three vectors (making use of the fact that we have already carried out steps (19)-(21) of that algorithm). Let the closest points of Λ''_8 to (46)-(48) be

$$p(j,1),\ p(j,2),\ p(j,3)$$

respectively, and let

$$d(j,k) \;=\; \mathrm{dist}^2(x, p(j,k)) \;. \tag{49}$$

Main stage

```
Set record $= 0$ and $j^* = 0$.
For $j = 0$ through 4095,
    obtain $\chi_1^j, \chi_2^j, \chi_3^j$ from Table VII, and calculate the squared distance
        $d = d(\chi_1^j, 1) + d(\chi_2^j, 2) + d(\chi_3^j, 3)$        (50)
    If $d <$ record, set record $= d$ and $j^* = j$.
```

After the 4095-th step, a closest point of Λ_{24} to x is

$$u \;=\; (p(\chi_1^j, 1),\; p(\chi_2^j, 2),\; p(\chi_3^j, 3))$$

and

$$\mathrm{dist}^2(x,u) \;=\; \text{record}.$$

The number of steps is roughly $4 \cdot 168 = 672$ for the first precomputation step, $3 \cdot 256 \cdot 56 = 43008$ for the second, plus $3 \cdot 4096 = 12288$ for the main stage, a total of 55968.

To see that this works, we remark that this is the algorithm of (2.15) based on the sublattice $\Lambda'_8 \oplus \Lambda'_8 \oplus \Lambda'_8$ of index 4096 in Λ_{24}. The quantity d in (50) is the squared distance from x to a nearest vector in the coset

$$\mathbf{1} \;+\; \Lambda'_8 \oplus \Lambda'_8 \oplus \Lambda'_8 \;+\; (a+t,\; b+t,\; c+t) \;.$$

Acknowledgements

We thank A. R. Calderbank, A. M. Odlyzko, M. R. Schroeder and A. D. Wyner for some helpful discussions.

List of Footnotes

(1) Page 4. As the automobile advertisements say, use these figures for comparison only. The actual running time will depend on the relative speeds of addition and multiplication, etc., and will probably be greater than the figures given here. We have tried, however, to evaluate all the algorithms in a uniform manner.

Figure Captions

Figure 1. (a) First-order Reed-Muller code of length 4. (b) An orthogonal matrix.

Figure 2. A code formed by ‘‘gluing’’ subcodes $@_1$, $@_2$, $@_3$ together.

List of Table Captions

Table I. 128 coset representatives for @ in $\&_{24}$, written in +1, −1 notation, and arranged in some arbitrary order. The coordinates are divided into six blocks of 4. A bar indicates a negative number.

Table II. The cross-reference table χ_N^j. An entry $\chi_N^j = m$ indicates that the sign combination appearing in block N of $\rho^{(j)}$ (in Table I) is to be found in row m of Table III.

Table III. Gray code table for block I. The i-th entry is $G_{\mathrm{I}}(i)$.

Table IV. (a) The 16 choices for a, b, c in (37), and (t) the 16 choices for t. A bar indicates a negative number. When multiplied by 2 these vectors are in Λ_8.

Table V. (a) The 16 choices for a, b, c in (43), and (t) the 16 choices for t. When multiplied by 2 these vectors are in Λ'_8.

Table VI. The 256 sums $a + t$, $a \varepsilon$ Table Va, $t \varepsilon$ Table Vt, arranged in some arbitrary order.

Table VII. The cross-reference table.

TABLE I

	I	II	III	IV	V	VI
$\rho^{(0)}$	1111	1111	1111	1111	1111	1111
$\rho^{(1)}$	$\overline{11}11$	$\overline{11}11$	$\overline{11}11$	$\overline{11}11$	1111	1111
$\rho^{(2)}$	$\bar{1}1\bar{1}1$	$\bar{1}1\bar{1}1$	$\bar{1}1\bar{1}1$	$\bar{1}1\bar{1}1$	1111	1111
$\rho^{(3)}$	$1\overline{11}1$	$1\overline{11}1$	$1\overline{11}1$	$1\overline{11}1$	1111	1111
$\rho^{(4)}$	$\bar{1}1\bar{1}1$	$\bar{1}11\bar{1}$	$\overline{11}11$	1111	$\overline{11}11$	1111
	...	...	...	...	...	...

TABLE II

j	I	II	III	IV	V	VI
0	0	0	0	0	0	0
1	2	2	2	2	0	0
2	6	6	6	6	0	0
3	4	4	4	4	0	0
4	6	14	2	0	2	0
			...			

TABLE III

0	$x_1 + x_2 + x_3 + x_4$
1	$-x_1 + x_2 + x_3 + x_4$
2	$-x_1 - x_2 + x_3 + x_4$
3	$x_1 - x_2 + x_3 + x_4$
4	$x_1 - x_2 - x_3 + x_4$
5	$-x_1 - x_2 - x_3 + x_4$
6	$-x_1 + x_2 - x_3 + x_4$
7	$x_1 + x_2 - x_3 + x_4$
8	$x_1 + x_2 - x_3 - x_4$
9	$-x_1 + x_2 - x_3 - x_4$
10	$-x_1 - x_2 - x_3 - x_4$
11	$x_1 - x_2 - x_3 - x_4$
12	$x_1 - x_2 + x_3 - x_4$
13	$-x_1 - x_2 + x_3 - x_4$
14	$-x_1 + x_2 + x_3 - x_4$
15	$x_1 + x_2 + x_3 - x_4$

TABLE IV

(a)	(t)
0000 0000	0000 0000
4000 0000	2220 0200
2222 0000	2202 0002
$\bar{2}222\ 0000$	2022 0020
2200 2200	0222 2000
$\bar{2}200\ 2200$	2200 2020
2200 0022	2020 2002
$\bar{2}200\ 0022$	2002 2200
2020 2020	$\bar{3}111\ 1111$
$\bar{2}020\ 2020$	$3\bar{1}\bar{1}1\ 1\bar{1}11$
2020 0202	$3\bar{1}1\bar{1}\ 111\bar{1}$
$\bar{2}020\ 0202$	$31\bar{1}\bar{1}\ 11\bar{1}1$
2002 2002	$3111\ 1\bar{1}\bar{1}\bar{1}$
$\bar{2}002\ 2002$	$3\bar{1}11\ \bar{1}1\bar{1}1$
2002 0220	$31\bar{1}1\ \bar{1}11\bar{1}$
$\bar{2}002\ 0220$	$311\bar{1}\ \bar{1}\bar{1}11$

TABLE V

(a)	(t)
0000 0000	0000 0000
2200 0000	$1111\ \bar{1}111$
2020 0000	$\bar{1}111\ 2000$
2002 0000	2000 1111
2000 2000	1102 1100
2000 0200	$2011\ 001\bar{1}$
2000 0020	$1100\ 20\bar{1}1$
2000 0002	$20\bar{1}1\ \bar{1}100$
1111 1111	1210 1010
$\bar{1}\bar{1}11\ 1111$	$2101\ 0\bar{1}01$
$\bar{1}1\bar{1}1\ 1111$	$1010\ 210\bar{1}$
$\bar{1}11\bar{1}\ 1111$	$210\bar{1}\ \bar{1}010$
$\bar{1}111\ \bar{1}111$	1021 1001
$\bar{1}111\ 1\bar{1}11$	$2110\ 01\bar{1}0$
$\bar{1}111\ 11\bar{1}1$	$1001\ 2\bar{1}10$
$\bar{1}111\ 111\bar{1}$	$2\bar{1}10\ \bar{1}001$

TABLE VI

$\rho^{(0)}$	0000 0000
$\rho^{(1)}$	2200 0000
$\rho^{(2)}$	2020 0000
	. . .
$\rho^{(8)}$	1111 1111
	. . .
$\rho^{(16)}$	1111 $\bar{1}$111
$\rho^{(17)}$	3311 $\bar{1}$111
	. . .

TABLE VII

$\chi^0 = (0,0,0)$
$\chi^1 = (1,1,0)$
$\chi^2 = (1,0,1)$
$\chi^3 = (0,1,1)$
$\chi^4 = (16,16,16)$
$\chi^5 = (17,17,16)$
. . .

References

1. G. Battail, *Décodage pondéré optimal des codes linéaires en blocs. I. Emploi simplifié du diagramme du trellis,* Ann. Télécommun. **38** (1983), 443-459.

2. G. Battail, M. C. Decouvelaere and P. Godlewski, *Replication decoding*, IEEE Trans. Information Theory, **IT-25** (1979), 332-345.

3. L. D. Baumert and R. J. McEliece, *Performance of some block codes on a Gaussian channel*, in *Proc. 1975 Internat. Telemetering Conf.*, Washington D.C., pp. 189-195.

4. L. D. Baumert and R. J. McEliece, *Soft decision decoding of block codes*, Deep Space Network Progress Report 42-47, Jet Propulsion Laboratory, Calif. Inst. Tech., Calif., July 1978, pp. 60-64.

5. L. D. Baumert, R. J. McEliece and G. Solomon, *Decoding with multipliers*, Deep Space Network Progress Report 42-34, Jet Propulsion Laboratory, Calif. Inst. Tech., Calif., August 1976, pp. 43-46.

6. T. Berger, *Optimum quantizers and permutation codes*, IEEE Trans. Information Theory, **IT-18** (1972), 759-765.

7. T. Berger, F. Jelinek and J. K. Wolf, *Permutation codes for sources*, IEEE Trans. Information Theory, **IT-18** (1972), 160-169.

8. E. R. Berlekamp, *The technology of error-correcting codes*, Proc. IEEE, **68** (1980), 564-593.

9. R. E. Blahut, *Theory and Practice of Error Control Codes*, Addison-Wesley, Reading, Mass., 1983, especially p. 482.

10. I. F. Blake, *The Leech lattice as a code for the Gaussian channel*, Information and Control, **19** (1971), 66-74.

11. R. W. D. Booth, M. A. Herro and G. Solomon, *Convolutional coding techniques for certain quadratic residue codes*, in *Proc. 1975 Internat. Telemetering Conf.*, Washington D.C., pp. 168-177.

12. P. de Buda, *Encoding and decoding algorithms for an optimal lattice-based code*, IEEE Conference Record 81CH1648-5, Internat. Conf. Commun., Denver June 1981, pp. 65.3.1 to 65.3.5.

13. R. de Buda, *The upper error bound of a new near-optimal code*, IEEE Trans. Information Theory, **IT-21** (1975), 441-445.

14. D. Chase, *A class of algorithms for decoding block codes with channel measurement information*, IEEE Trans. Information Theory, **IT-18** (1972), 170-182.

15. J. H. Conway, *Three lectures on exceptional groups*, in *Finite Simple Groups*, M. B. Powell and G. Higman, editors, Academic Press, N.Y., 1971, pp. 215-247.

16. J. H. Conway, *The Golay codes and the Mathieu groups*, Chapter 12 of [29].

17. J. H. Conway, R. A. Parker and N. J. A. Sloane, *The covering radius of the Leech lattice*, Proc. Royal Soc. London, **A 380** (1982), 261-290.

18. J. H. Conway and V. Pless, *On the enumeration of self-dual codes*, J. Combinatorial Theory, **28A** (1980), 26-53.

19. J. H. Conway and N. J. A. Sloane, *On the enumeration of lattices of determinant one*, J. Number Theory, **15** (1982), 83-94.

20. J. H. Conway and N. J. A. Sloane, *The unimodular lattices of dimension up to 23 and the Minkowski - Siegel mass constants*, Europ. J. Combinatorics, **3** (1982), 219-231.

21. J. H. Conway and N. J. A. Sloane, *Voronoi regions of lattices, second moments of polytopes, and quantization*, IEEE Trans. Information Theory, **IT-28** (1982), 211-226.

22. J. H. Conway and N. J. A. Sloane, *Fast quantizing and decoding algorithms for lattice quantizers and codes*, IEEE Trans. Information Theory, **IT-28** (1982), 227-232.

23. J. H. Conway and N. J. A. Sloane, *Twenty-three constructions for the Leech lattice*, Proc. Royal Soc. London, **A 381** (1982), 275-283.

24. J. H. Conway and N. J. A. Sloane, *Lorentzian forms for the Leech lattice*, Bull. Amer. Math. Soc., **6** (1982), 215-217.

25. J. H. Conway and N. J. A. Sloane, *Laminated lattices*, Annals Math., **116** (1982), 593-620.

26. J. H. Conway and N. J. A. Sloane, *Complex and integral laminated lattices*, Trans. Amer. Math. Soc., **280** (1983), 463-490.

27. J. H. Conway and N. J. A. Sloane, *A fast encoding method for lattice codes and quantizers*, IEEE Trans. Information Theory, **IT-29** (1983), 820-824.

28. J. H. Conway and N. J. A. Sloane, *On the Voronoi regions of certain lattices*, SIAM J. Algebraic Discrete Methods, **5** (1984), 294-305.

29. J. H. Conway and N. J. A. Sloane, *The Leech lattice, Sphere Packings, and Related Topics*, Springer-Verlag, N.Y., in preparation.

30. H. S. M. Coxeter, *Regular Polytopes*, Dover Publications, N.Y., 3rd ed., 1973.

31. P. Delsarte, *Partial-optimal piecewise decoding of linear codes*, IEEE Trans. Information Theory, **IT-24** (1978), 70-75.

32. B. G. Dorsch, *A decoding algorithm for binary block codes and j-ary output channels*, IEEE Trans. Information Theory, **IT-20** (1974), 391-394.

33. G. S. Evseev, *Complexity of decoding for linear codes* (in Russian), Problemy Peradachi Informatsii, **19** (No. 1, 1983), 3-8. English translation in Problems of Information Transmission, **19** (No. 1, 1983), 1-6.

34. G. D. Forney, Jr., *Generalized minimum distance decoding*, IEEE Trans. Information Theory, **IT-12** (1966), 125-131.

35. G. D. Forney, Jr., *The Viterbi algorithm*, Proc. IEEE, **61** (1973), 268-278.

36. A. Gersho, *Asymptotically optimal block quantization*, IEEE Trans. Information Theory, **IT-25** (1979), 373-380.

37. A. Gersho, *On the structure of vector quantizers*, IEEE Trans. Information Theory, **IT-28** (1982), 157-166.

38. E. N. Gilbert, *Gray codes and paths on the n-cube*, Bell Syst. Tech. J., **37** (1958), 815-826.

39. D. M. Gordon, *Minimal permutation sets for decoding the binary Golay code*, IEEE Trans. Information Theory, **IT-28** (1982), 541-543.

40. R. R. Green, *A serial orthogonal decoder*, JPL Space Programs Summary, Vol. 37-39-IV (1966), 247-253.

41. R. R. Green, *Analysis of a serial orthogonal decoder*, JPL Space Programs Summary, Vol. 37-53-III (1968), 185-187.

42. C. M. Hackett, *An efficient algorithm for soft decision decoding of the* (*24,12*) *extended Golay code*, IEEE Trans. Comm., **COM-29** (1981), 909-911 and **COM-30** (1982), 554.

43. C. R. P. Hartmann and L. D. Rudolph, *An optimum symbol-by-symbol decoding rule for linear codes*, IEEE Trans. Information Theory, **IT-22** (1976), 514-517.

44. T. Y. Hwang, *Decoding linear block codes for minimizing word error rate*, IEEE Trans. Information Theory, **IT-25** (1979), 733-737.

45. T. Y. Hwang, *Efficient optimal decoding of linear block codes*, IEEE Trans. Information Theory, **IT-26** (1980), 603-606.

46. D. E. Knuth, The Art of Computer Programming, Addison-Wesley, Reading, Mass., Vol. 3, 1973, p. 216.

47. J. Leech, *Notes on sphere packings*, Canad. J. Math. **19** (1967), 251-267.

48. J. Leech and N. J. A. Sloane, *Sphere packing and error-correcting codes,* Canad. J. Math., **23** (1971), 718-745.

49. J. I. Lepowsky and A. E. Meurman, *An E_8-approach to the Leech lattice and the Conway group,* J. Algebra, **77** (1982), 484-504.

50. F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, North-Holland, Amsterdam, 1977.

51. J. L. Massey, *Foundation and method of channel encoding,* in *Proc. Int. Conf. on Info. Theory and Systems*, NTG-Fachberichte (Berlin), **65** (1978), 148-157.

52. H. Miyakawa and T. Kaneko, *Decoding algorithm for error-correcting codes by use of analog weights*, Electron. Commun. Japan, **58-A** (1975), 18-27.

53. H. V. Niemeier, *Definite quadratischer Formen der Dimension 24 und Diskriminante 1,* J. Number Theory, **5** (1973), 142-178.

54. J. Paluszkiewicz, R. Stasinski and Z. Szymanski, *An algorithm for soft-decision decoding of the Golay and other linear codes,* 1983 IEEE International Symposium on Information Theory, Abstracts of Papers, IEEE Press, N.Y., 1983, p. 137.

55. M. Phister, Jr., *Logical Design of Digital Computers*, Wiley, N.Y., 1960, pp. 232-234, 399-401.

56. V. Pless, *The children of the* (*32,16*) *doubly even codes*, IEEE Trans. Information Theory, **IT-24** (1978), 738-746.

57. V. Pless and N. J. A. Sloane, *On the classification and enumeration of self-dual codes*, J. Combinatorial Theory, **18** (1975), 313-335.

58. E. C. Posner, *Combinatorial structures in planetary reconnaissance*, in *Error Correcting Codes*, H. B. Mann, editor, Wiley, N.Y., 1969, pp. 15-46.

59. E. M. Reingold, J. Nievergelt and N. Deo, *Combinatorial Algorithms: Theory and Practice,* Prentice-Hall, Englewood Cliffs, N.J., 1977.

60. L. D. Rudolph, C. R. P. Hartmann, T. Y. Hwang and N. Q. Duc, *Algebraic analog decoding of lineary binary codes*, IEEE Trans. Information Theory, **IT-25** (1979), 430-440.

61. S. S. Ryskov and E. P. Baranovskii, *Solution of the problem of least dense lattice coverings of five-dimensional space by equal spheres* (in Russian), Dokl. Akad. Nauk SSSR, **222** (1975), 39-42. English translation in Soviet Math. Doklady **6** (1975), 586-590.

62. D. Slepian, *Permutation modulation,* Proc. IEEE, **53** (1965), 228-236.

63. N. J. A. Sloane, *Binary codes, lattices and sphere packings,* in *Combinatorial Surveys*, P. J. Cameron, ed., Academic Press, N.Y., 1977, pp. 117-164.

64. N. J. A. Sloane, *A note on the Leech lattice as a code for the Gaussian channel,* Information and Control, **46** (1980), 270-272.

65. N. J. A. Sloane, *Tables of sphere packings and spherical codes*, IEEE Trans. Information Theory, **IT-27** (1981), 327-338.

66. G. Solomon and H. C. A. van Tilborg, *A connection between block and convolutional codes*, SIAM J. Appl. Math., **37** (1979), 358-369.

67. N. N. Tendolkar and C. R. P. Hartmann, *Generalization of Chase algorithms for soft decision decoding of binary linear codes*, IEEE Trans. Information Theory, **IT-29** (1984), 714-721.

68. A. J. Viterbi, *Error bounds for convolutional codes and an asymptotically optimum decoding algorithm*, IEEE Trans. Information Theory, **IT-13** (1967), 260-269.

69. J. K. Wolf, *Efficient maximum likelihood decoding of linear block codes using a trellis*, IEEE Trans. Information Theory, **IT-24** (1978), 76-80.

70. J. Wolfmann, *Nouvelles methodes de decodage du code de Golay* (24, 12, 8), Rev. CETHEDEC Cahier, No. 2, 1981, pp. 79-88.

71. J. Wolfmann, A permutation decoding of the (24, 12, 8) Golay code, IEEE Trans. Information Theory, **IT-29** (1983), 748-750.