Fast Decoding of the Leech Lattice

YAIR BE'ERY, BOAZ SHAHAR, AND JAKOV SNYDERS

Abstract—**New efficient algorithm for maximum likelihood soft-decision decoding of the Leech lattice is presented. The superiority of this decoder with respect to both computational and memory complexities, compared to previously published decoding methods is demonstrated. Gain factors in the range of 2–10 are achieved. The Leech lattice has been recently suggested for practical implementation of block-coded modulation and the decoder is described here in that context.**

I. Introduction

THE Leech lattice [1] has been suggested as a candidate for implementation of block-coded modulation techniques for band-limited channels [2], [3]. It has been also used recently in a 19 200 bit/s commercial modem developed by Codex–Motorola [4]. The construction of an efficient decoder for this lattice was addressed in the last two years by Conway and Sloane [5], Forney [3], and Lang and Longstaff [4]. New algorithms were introduced, more efficient than those previously known [2], [6].

Although for many years it was suggested that codes based on dense lattices could be used on band-limited channels to achieve substantial coding gain, only recently this idea really gained enough attention to be realized in practical applications. The pioneering work of Ungerboeck [7], along with the intensive research of his many successors who utilized trellis codes for band-limited channels, inspired also a great attention on lattices, not only by mathematicians but also by electrical engineers.

The Leech lattice Λ_{24}, which is a 24-dimensional lattice, is one of the most interesting and well studied lattices [8]. When used as a block code in a modem [4], this lattice offers a nominal coding gain of 6 dB [9], thus achieving [10] a performance comparable to that of the best trellis-coded modems [9]. Therefore, the controversy about the preferability of convolutional codes over block codes remains unresolved in this case. Since performances are comparable, the next important issue to be addressed is the complexity of implementation, which is dominated by the decoding algorithm [4]. The decoding algorithm presented in this paper seems to achieve a computational complexity which is also comparable to that of the equivalent trellis codes.

Manuscript received April 4, 1988; revised October 11, 1988. This paper was presented in part at the Beijing International Workshop on Information Theory, Beijing, China, July 4–7, 1988.

Y. Be'ery and J. Snyders are with the Department of Electronic Communications Control and Computer Systems, School of Engineering, Tel-Aviv University, Ramat-Aviv 69978, Israel.

B. Shahar is with National Semiconductors (Israel) Ltd., Herzlia B 46104, Israel.

IEEE Log Number 8928303.

The Leech lattice has a high degree of structure, and can be defined in a number of ways [8]. The way of constructing Λ_{24} has a major influence of the decoding complexity. As customary, we regard Λ_{24} as the union of two cosets of the Leech half-lattice H_{24}. For each of the cosets of H_{24} we use a description given in terms of the (24, 12) extended Golay code $\mathcal{G}_{24}$ combined with a one-bit parity-check code and a repetition code. This viewpoint enables us to employ our fast soft-decision decoding algorithm for $\mathcal{G}_{24}$ [12], [13], [16]. The description of Λ_{24} and the related lattice code are elaborated in the following paragraphs.

For constructing the 24-dimensional lattice we begin with a two-dimensional lattice, which is partitioned into 16 subsets according to Ungerboeck's method [7]. The partition tree along with the labels of the subsets, as defined in [2], is given in Fig. 1. The two-dimensional lattice is generated by taking the basic pattern given by the 16-point constellation of Fig. 2, and tiling all of the two-space with nonoverlapping copies of this pattern. For assigning actual coordinate values to the labels of the constellation of Fig. 2, and consequently to all the points of the two-dimensional lattice, in a way that conforms with the formal definition of Λ_{24}, the following method can be used (for details see the Appendix of [2]). Choose a point A_{000} as the origin and rotate the configuration by 45°. By proper scaling, the A_{ijk} points then run through all pairs of even coordinates and the B_{ijk} points are all pairs of odd coordinates. This assignment of coordinates agrees, within a scale factor, with that of Sloane [14].

The 24-dimensional Leech lattice consists of all sequences of 12 points, each point taken from the two-dimensional lattice, such that in each sequence [2]:

a) all points are either A_{ijk} points or B_{ijk} points,

b) the 24 (i, j) subscripts are a codeword of $\mathcal{G}_{24}$, and

c) the third subscripts k are constrained to satisfy an overall parity check in the following way. If the sequence is of all A points, then the overall k parity is even, while if the sequence is of all B points, the overall k parity is odd.

By restricting the points of the two-dimensional lattice to be only A_{ijk} points, the Leech half lattice H_{24} is obtained, whereas the B_{ijk} points constitute a coset of H_{24}.

In practice, using the conventional digital quadrature amplitude modulation (QAM) signaling scheme, which is usefully represented by a two-dimensional constellation of all possible signal points, only a finite subset of points of the two-dimensional lattice is utilized. A simple example of a 64-point square constellation, which is partitioned into 16 subsets of 4 points each, is given by Fig.

0733-8716/89/0800-0959$01.00

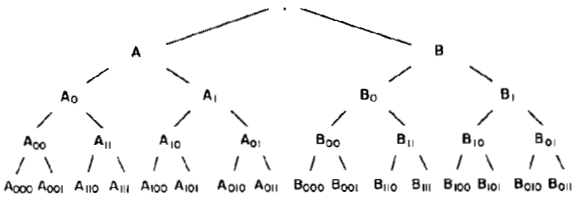

Fig. 1. The partition tree of the 16 subsets.

$$\begin{array}{cccc} A_{000} & B_{000} & A_{110} & B_{110} \\ B_{101} & A_{010} & B_{010} & A_{101} \\ A_{111} & B_{111} & A_{001} & B_{001} \\ B_{011} & A_{100} & B_{100} & A_{011} \end{array}$$

Fig. 2. The basic configuration of 16-point.

3. By restricting the A_{ijk}'s and B_{ijk}'s to belong to such a finite two-dimensional constellation, only a finite subset of the vectors of the Leech lattice is generated. Such subset is called a Leech lattice code, and its encoding scheme is illustrated by Fig. 4.

According to this description, a maximum likelihood decoding procedure for the Leech lattice, as well as for any Leech lattice-code, can be separated into two similar parts [2]: one for finding the closest A sequence, assuming that only A points were transmitted, and the other for finding the closest B sequence, assuming that only B points were sent. By comparing the best A and B sequences and choosing the better one, the final decision is reached. This separation of the decoding procedure can also be viewed by regarding Λ_{24} as the union of two cosets of H_{24}. Hereafter, only the decoder for the closest A sequence, which is acutally a decoder for H_{24}, will be considered. The computational complexity of the complete decoding algorithm is twice that of the A sequence decoder plus one.

The decoder for Λ_{24}, which is derived in this paper, utilizes the symmetric attributes of certain sublattices and subcodes of H_{24} and $\mathcal{G}_{24}$ to a greater extent than in previous papers [2]–[6]. To the best of our knowledge, this is the most efficient decoder even published for the Leech lattice. In Section II, we derive the basic ideas of the new decoder. This decoder is, in part, a utilization of the well-known Wagner idea of changing the least reliable decision if a single overall parity-check fails. This method was introduced as a soft-decision decoder for single-parity-check codes [17]. Although some of the ideas appearing in our decoder are mentioned in [3]–[5], they were not utilized there in the most efficient way. The explicit expressions derived in Section II helped to achieve a more efficient decoder. In Section III, the decoding algorithm is formulated, and the complexity is assessed. Section IV concludes the paper with some more advanced ideas for achieving a further reduction of the complexity. These ideas are based on a generalization of the Wagner decoding method to two parity constraints (see also [16]). A comparison with the complexity of some trellis-coded modulation schemes will also be discussed in Section IV.

$$\begin{array}{cccccccc} A_{000} & B_{000} & A_{110} & B_{110} & A_{000} & B_{000} & A_{110} & B_{110} \\ B_{101} & A_{010} & B_{010} & A_{101} & B_{101} & A_{010} & B_{010} & A_{101} \\ A_{111} & B_{111} & A_{001} & B_{001} & A_{111} & B_{111} & A_{001} & B_{001} \\ B_{011} & A_{100} & B_{100} & A_{011} & B_{011} & A_{100} & B_{100} & A_{011} \\ A_{000} & B_{000} & A_{110} & B_{110} & A_{000} & B_{000} & A_{110} & B_{110} \\ B_{101} & A_{010} & B_{010} & A_{101} & B_{101} & A_{010} & B_{010} & A_{101} \\ A_{111} & B_{111} & A_{001} & B_{001} & A_{111} & B_{111} & A_{001} & B_{001} \\ B_{011} & A_{100} & B_{100} & A_{011} & B_{011} & A_{100} & B_{100} & A_{011} \end{array}$$

Fig. 3. 64-point square constellation.

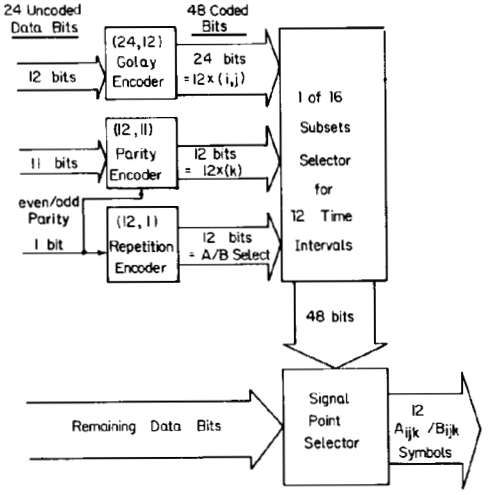

Fig. 4. A Leech lattice encoder.

The complexity of the new decoder, as described in Section III, is at most ~ 8000 operations, like addition/subtraction. By utilizing the ideas of Section IV the complexity is reducible to ~ 6000 operations in the worst case, (~ 5000 operations on the average). Compare this to $\sim 56\,000$ operations achieved in [5] where Λ_{24} is regarded as the union of 2^{12} cosets of a much simpler lattice, and to $\sim 15\,000$ operations achieved in [3] by using a 256-state trellis diagram to represent Λ_{24}, which was reduced in [4] to about 10 000 operations by using a Wagner-type decoding idea. (As stated in [5], these figures are for comparison only. The actual running time will depend on the specific technology in use. We tried, however, to evaluate all the algorithms in a uniform manner.) Furthermore, a reduction in the memory requirement is achieved by the decoder proposed here.

II. The Basic Ideas of the New Decoder

The decoder receives a sequence of twelve two-dimensional symbols, $r(n)$, $n = 1, 2, \cdots, 12$. As its first step

[2], the decoder can always choose the closest point in each subset A_{ijk} to each received symbol as a representative of that subset (as there is no reason to prefer any more distant point), and evaluate the squared Euclidean distance (SED) to that representative of the subset, denoted by $d_{ijk}(n)$, $n = 1, 2, \cdots, 12$. Now, the decoder can proceed to determine the best (with minimum SED) A sequence of subsets using the above representatives, with their distances from the received points, as proxies for the corresponding subsets [2].

Denote, for $n = 1, 2, \cdots, 12$,

$$d_{ij}(n) = \min\left\{d_{ij0}(n), d_{ij1}(n)\right\}, \tag{1}$$

and define the parity $p_{ij}(n)$ associated with $d_{ij}(n)$ as follows:

$$p_{ij}(n) = \begin{cases} 0; & \text{if } d_{ij}(n) = d_{ij0}(n) \\ 1; & \text{if } d_{ij}(n) = d_{ij1}(n). \end{cases} \tag{2}$$

Denote

$$\Delta_{ij}(n) = \left|d_{ij0}(n) - d_{ij1}(n)\right| \tag{3}$$

where $|\cdot|$ stands for absolute value. Let $\boldsymbol{c}$ be a codeword of $\mathcal{G}_{24}$, and define its overall parity by

$$P(\boldsymbol{c}) = \oplus\sum_{n=1}^{12} p_{ij}(n) \tag{4}$$

where $\oplus\Sigma$ stands for summation in GF(2), and the 24 (i, j) subscripts constitute $\boldsymbol{c}$ (i.e., $P(\boldsymbol{c})$ is equal to the parity of the k indexes of $d_{ijk}(n)$, $n = 1, 2, \cdots, 12$). Notice that each $\Sigma_{n=1}^{12} d_{ij}(n)$ is a distance from a codeword of $\mathcal{G}_{24}$, provided that $P(\boldsymbol{c}) = 0$.

The idea of the preceding derivations and the decoding algorithm in the sequel may also be understood by regarding H_{24}, which is a lattice obtained from $\mathcal{G}_{24}$ under Construction B of [11], as a lattice obtained under Construction A [11], but with an additional parity constraint. In this way one can apply the method used in [5] to decode lattices obtained under Construction A (see algorithm 12 and a corresponding lemma in [5]), and then make a correction if the parity check fails. In [5], a remark therein states that it does not appear possible to modify algorithm 12, but if such modification were found it would further speed up the decoding algorithm for the Leech lattice given there. The method we found may be regarded as such modification. This observation seems to have a wider range of applicability. In most general terms, the decoding algorithm of [5] for lattices obtained from Construction A is apparently modifiable, by using a Wagner-type idea, to any lattice obtained from construction B.

These considerations yield the following straightforward decoding rule for H_{24}. For each of the 2^{12} $\boldsymbol{c} \in \mathcal{G}_{24}$ calculate

$$M(\boldsymbol{c}) = \sum_{n=1}^{12} d_{ij}(n). \tag{5}$$

Check the parity, and in case the parity-check fails set

$$M(\boldsymbol{c}) = \sum_{n=1}^{12} d_{ij}(n) + \Delta^{\min}(\boldsymbol{c}) \tag{6}$$

where

$$\Delta^{\min}(\boldsymbol{c}) = \min\left\{\Delta_{ij}(n), n = 1, 2, \cdots, 12\right\}. \tag{7}$$

Finally, let $\boldsymbol{c}_{\min}$ be that $\boldsymbol{c}$ (or, in the case of a tie, an arbitrary $\boldsymbol{c}$) which minimizes $M(\boldsymbol{c})$ over $\mathcal{G}_{24}$. The best A sequence is easily determined with the aid of $\boldsymbol{c}_{\min}$. The (i, j) indexes of the 12 A_{ijk} are the entries of $\boldsymbol{c}_{\min}$. The k indexes are given by $p_{ij}(n)$, $n = 1, 2, \cdots, 12$, which compose $P(\boldsymbol{c}_{\min})$, except that a single k index must be inverted if $M(\boldsymbol{c}_{\min})$ was calculated by (6); it is the lth k index where $\Delta^{\min}(\boldsymbol{c}_{\min}) = \Delta_{ij}(l)$.

It is convenient to combine (5) and (6) into the following single expression:

$$M(\boldsymbol{c}) = \sum_{n=1}^{12} d_{ij}(n) + \Delta^{\min}(\boldsymbol{c}) \cdot P(\boldsymbol{c}). \tag{8}$$

Because the aforementioned decoding rule is essentially a $\mathcal{G}_{24}$ decoder combined with an additional parity-check constraint, it is possible to exploit the symmetric attributes of the generator matrix, G, of $\mathcal{G}_{24}$ given in Fig. 5. This generator matrix is a row-permuted version of the generator matrix given in [3]. The structure of G and the decoding techniques of [12], [13], and [16], which employ parity constraints in soft decoding algorithms, lead to a substantial reduction in the computations involved in (8) and the minimization of $M(\boldsymbol{c})$ over $\mathcal{G}_{24}$.

For utilizing the structure of $\mathcal{G}_{24}$ we split the first term on the right side of (8) into the three terms $\Sigma_{n=1}^{4} d_{ij}(n)$, $\Sigma_{n=4}^{8} d_{ij}(n)$, $\Sigma_{n=9}^{12} d_{ij}(n)$; these terms correspond to the three vertical sections of G. Accordingly, denote $\boldsymbol{c}^1 = (c_1, c_2, \cdots, c_8)$, $\boldsymbol{c}^2 = (c_9, c_{10}, \cdots, c_{16})$, and $\boldsymbol{c}^2 = (c_{17}, c_{18}, \cdots, c_{24})$. Thus, $\boldsymbol{c} = (\boldsymbol{c}^1, \boldsymbol{c}^2, \boldsymbol{c}^3)$. Write $\boldsymbol{c} = \boldsymbol{s}G$ where $\boldsymbol{s} = (s_1, s_2, \cdots, s_{12})$ is a source-word for $\mathcal{G}_{24}$. Then all $\boldsymbol{s}^1 = (s_1, s_4, s_6, s_7, s_{10}, s_{11}, s_{12})$, $\boldsymbol{s}^2 = (s_2, s_4 \oplus s_5, s_6 \oplus s_8, s_7 \oplus s_9, s_{10}, s_{11}, s_{12})$ and $\boldsymbol{s}^3 = (s_3, s_5, s_8, s_9, s_{10}, s_{11}, s_{12})$ are the source-words which generate all possible $\boldsymbol{c}^1$, $\boldsymbol{c}^2$, and $\boldsymbol{c}^3$, respectively. Let $\boldsymbol{c}' \in \mathcal{G}_{24}$ be a codeword generated by the nine lower rows of G, which corresponds to $\boldsymbol{s}' = (s_4, s_5, \cdots, s_{12})$. Note that in all the above definitions the mapping between the various families of codewords and the corresponding families of source-words is a one-to-one mapping. Therefore, hereafter, we shall freely exchange between the two types of words. For example, we shall write $M(\boldsymbol{s})$ instead of $M(\boldsymbol{c})$.

Let

$$\mathfrak{M}(\boldsymbol{s}') = \min_{s_1, s_2, s_3} M(\boldsymbol{s}). \tag{9}$$

It is evident that for a fixed $\boldsymbol{s}'$ one may choose s_1, s_2, and s_3 independently of each other, for minimizing $M(\boldsymbol{s})$. As will be explained in the next paragraphs, it will be rewarding to use a "normalized distance," $D_{ij}(n)$, along

$$G = \begin{bmatrix} 11111111 & 00000000 & 00000000 \\ 00000000 & 11111111 & 00000000 \\ 00000000 & 00000000 & 11111111 \\ 11110000 & 11110000 & 00000000 \\ 00000000 & 11110000 & 11110000 \\ 11001100 & 11001100 & 00000000 \\ 10101010 & 10101010 & 00000000 \\ 00000000 & 11001100 & 11001100 \\ 00000000 & 10101010 & 10101010 \\ 01111000 & 01111000 & 01111000 \\ 10011100 & 10011100 & 10011100 \\ 01010110 & 01010110 & 01010110 \end{bmatrix}$$

Fig. 5. The generator matrix for $\mathcal{G}_{24}$.

with an "offset adjustment," $OA_{ij}(n)$, which are defined, respectively, by

$$D_{ij}(n) = \left[d_{ij}(n) - d_{\bar{i}\bar{j}}(n)\right]/2; \quad n = 1, 2, \cdots, 12 \tag{10}$$

and

$$OA_{ij}(n) = \left[d_{ij}(n) + d_{\bar{i}\bar{j}}(n)\right]/2; \quad n = 1, 2, \cdots, 12 \tag{11}$$

where $\bar{x}$ denotes the binary complement of x. Notice that

$$D_{ij}(n) = -D_{\bar{i}\bar{j}}(n) \tag{12}$$

and

$$OA_{ij}(n) = OA_{\bar{i}\bar{j}}(n). \tag{13}$$

By (12), the normalized distance of a complement codeword $\bar{c}$ is the negative of the normalized distance of c. The offset adjustment compensates for the generally different increments added to the distances, by the normalization process, to different noncomplement codewords. The normalized distance enables us to use, as in [3]–[5], [12], [13], [16], an absolute value operation to make the two-way decision between a codeword and its complement. The absolute value operation is simpler, by definition, than a comparison operation, since it is actually a comparison with a fixed number, namely, zero. In some hardware/VLSI implementations, such as when a sign-magnitude notation is used for describing the values, it is considerably simpler to obtain an absolute value (i.e., assigning zero to the sign bit) than to compare two numbers. This is why in [3]–[5], as well as in this paper, the absolute value operation is preferred over the comparison operation and it is even neglected in the assessment of the total complexity. But the major advantage of using $D_{ij}(n)$ and $OA_{ij}(n)$ becomes clear from the following derivations.

Due to (8), (10), and (11),

$$\begin{aligned} M(s) &= \sum_{n=1}^{12} \left[D_{ij}(n) + OA_{ij}(n)\right] + \Delta^{\min}(c) \cdot P(c) \\ &= \sum_{n=1}^{4} D_{ij}(n) + \sum_{n=5}^{8} D_{ij}(n) + \sum_{n=8}^{12} D_{ij}(n) \\ &\quad + \sum_{n=1}^{4} OA_{ij}(n) + \sum_{n=5}^{8} OA_{ij}(n) + \sum_{n=9}^{12} OA_{ij}(n) \\ &\quad + \Delta^{\min}(c) \cdot P(c). \end{aligned} \tag{14}$$

We shall elaborate now upon the way of computing efficiently $M(s)$ for all s, using (14).

In view of (13), and the structure of the generator matrix G, each $\Sigma_{n=4v-3}^{4v} OA_{ij}(n)$, $v = 1, 2, 3$, is independent of $s_1, s_2, s_3, s_4, s_5, s_6, s_8$, and has at most 8 different values as s (or, equivalently, c) varies. To clarify this important point let us take a closer look, for instance, at the term

$$\sum_{n=1}^{4} OA_{ij}(n) = OA_{c_1c_2}(1) + OA_{c_3c_4}(2) + OA_{c_5c_6}(3) + OA_{c_7c_8}(4).$$

The 8 (i, j) subscripts of the 4 $OA_{ij}(n)$; $n = 1, 2, 3, 4$, constitute c^1, which corresponds to $s^1 = (s_1, s_4, s_6, s_7, s_{10}, s_{11}, s_{12})$. Hence, it is obvious that the term $\Sigma_{n=1}^{4} OA_{ij}(n)$ is independent of s_2, s_3, s_5, s_8, and s_9. Furthermore, due to the first row of G, $\bar{c}^1 = (\bar{c}_1, \bar{c}_2, \cdots, \bar{c}_8)$ corresponds to $(\bar{s}_1, s_4, s_6, s_7, s_{10}, s_{11}, s_{12})$, implying that $OA_{\bar{c}_1\bar{c}_2}(1) + OA_{\bar{c}_3\bar{c}_4}(2) + OA_{\bar{c}_5\bar{c}_6}(3) + OA_{\bar{c}_7\bar{c}_8}(4) = \Sigma_{n=1}^{4} OA_{\bar{i}\bar{j}}(n)$. But, according to (13), $\Sigma_{n=1}^{4} OA_{\bar{i}\bar{j}}(n) = \Sigma_{n=1}^{4} OA_{ij}(n)$ and therefore $\Sigma_{n=1}^{4} OA_{ij}(n)$ is also independent of s_1. Along the same line of thought $(\bar{c}_1, \bar{c}_2, \bar{c}_3, \bar{c}_4, c_5, c_6, c_7, c_8)$ and $(\bar{c}_1, \bar{c}_2, c_3, c_4, \bar{c}_5, \bar{c}_6, c_7, c_8)$ correspond to $(s_1, \bar{s}_4, s_6, s_7, s_{10}, s_{11}, s_{12})$ and $(s_1, s_4, \bar{s}_6, s_7, s_{10}, s_{11}, s_{12})$, respectively. Hence, due to (13), each of $OA_{\bar{c}_1\bar{c}_2}(1) + OA_{\bar{c}_3\bar{c}_4}(2) + OA_{c_5c_6}(3) + OA_{c_7c_8}(4)$ and $OA_{\bar{c}_1\bar{c}_2}(1) + OA_{c_3c_4}(2) + OA_{\bar{c}_5\bar{c}_6}(3) + OA_{c_7c_8}(4)$ is equal to $\Sigma_{n=1}^{4} OA_{ij}(n)$. Thus, the latter is also independent of s_4 and s_6. Altogether, $\Sigma_{n=1}^{4} OA_{ij}(n)$ depends only on s_7, s_{10}, s_{11}, and s_{12} and thus it seems to have 16 different values [which correspond to the 16 values of $(s_7, s_{10}, s_{11}, s_{12})$] as s varies. But, observing that $(\bar{c}_1, \bar{c}_2, \bar{c}_3, \bar{c}_4, \bar{c}_5, \bar{c}_6, c_7, c_8)$ corresponds to $(s_1, s_4, s_6, \bar{s}_7, s_{10}, s_{11}, \bar{s}_{12})$, it is evident that $\Sigma_{n=1}^{4} OA_{ij}(n)$ assumes no more than 8 different values [which correspond to the 8 values of $(s_7 \oplus s_{12}, s_{10}, s_{11})$] as s varies. A similar discussion is applicable to each of the other two terms $\Sigma_{n=5}^{8} OA_{ij}(n)$, and $\Sigma_{n=9}^{12} OA_{ij}(n)$. These values are listed, for each of the three terms (designated by $v = 1, 2, 3$), in Table II in conjunction with Table I (see also, Section III). Using them the total sum $\Sigma_{n=1}^{12} OA_{ij}(n)$, which is assigned at most $2^5 = 32$ different values as s varies [corresponding to the 32 possible values of $(s_7, s_9, s_{10}, s_{11}, s_{12})$], can be efficiently evaluated.

Each $\Sigma_{n=4v-3}^{4v} D_{ij}(n)$, $v = 1, 2, 3$, depends only on one of s_1, s_2, and s_3, namely, s_v, and has at most $2^7 = 128$ different values (corresponding to the 128 possible values of s^v) as s varies. By (12), $\Sigma_{n=4v-3}^{4v} D_{ij}(n) = -\Sigma_{n=4v-3}^{4v} D_{\bar{i}\bar{j}}(n)$. Hence, each $|\Sigma_{n=4v-3}^{4v} D_{ij}(n)|$ is independent of s_1, s_2, and s_3, and assumes 64 different values at most, corresponding to the 64 possible values of $(s_4, s_6, s_7, s_{10}, s_{11}, s_{12})$ for $v = 1$, $(s_4 \oplus s_5, s_6 \oplus s_8, s_7 \oplus s_9, s_{10}, s_{11}, s_{12})$ for $v = 2$, and $(s_5, s_8, s_9, s_{10}, s_{11}, s_{12})$ for $v = 3$. These 64 values are listed, for each of $v = 1$, 2, and 3, in Table III, which consists of eight parts, each of which is arranged in Gray code ordering of signs. This way of ordering, previously used in [5] for similar purpose, is a means for efficient computation of all the $\Sigma_{n=4v-3}^{4v} D_{ij}(n)$ involved.

TABLE I
SUMS OF PAIRS OF $OA_{ij}(n)$

q	$x_q(v)$	$y_q(v)$
1	$OA_{00}(4v-3) + OA_{00}(4v-2)$	$OA_{00}(4v-1) + OA_{00}(4v)$
2	$OA_{10}(4v-3) + OA_{10}(4v-2)$	$OA_{10}(4v-1) + OA_{10}(4v)$
3	$OA_{10}(4v-3) + OA_{00}(4v-2)$	$OA_{10}(4v-1) + OA_{00}(4v)$
4	$OA_{00}(4v-3) + OA_{10}(4v-2)$	$OA_{00}(4v-1) + OA_{10}(4v)$

TABLE II
8 POSSIBLE VALUES OF $\Sigma_{n=4v-3}^{4v} OA_{ij}(n)$

q	v=1: s_7 v=2: $s_7 \oplus s_9$ v=3: s_9	s_{10} s_{11} s_{12}	$z_q(v)$
1	1	0 0 1	$x_1(v) + y_1(v)$
	0	0 0 0	
2	1	0 0 0	$x_2(v) + y_2(v)$
	0	0 0 1	
3	1	0 1 1	$x_2(v) + y_1(v)$
	0	0 1 0	
4	1	0 1 0	$x_1(v) + y_2(v)$
	0	0 1 1	
5	1	1 0 1	$x_3(v) + y_3(v)$
	0	1 0 0	
6	1	1 0 0	$x_4(v) + y_4(v)$
	0	1 0 1	
7	1	1 1 1	$x_4(v) + y_3(v)$
	0	1 1 0	
8	1	1 1 0	$x_3(v) + y_4(v)$
	0	1 1 1	

The reduction from 128 values to only 64 complement pairs of $\Sigma_{n=4v-3}^{4v} D_{ij}(n)$, $v = 1, 2, 3$, and to 32 values of $\Sigma_{n=1}^{12} OA_{ij}(n)$ has a major influence on the computational and memory complexities of the proposed decoder. This is the main reason why it is rewarding to define the normalized distance and the offset adjustment. These ideas appear also in [4], but have not been expressed explicitly as done here by (10)–(14). Moreover, the observations that each $\Sigma_{n=4v-3}^{4v} OA_{ij}(n)$, $v = 1, 2, 3$ assumes 8 values at most and that $\Sigma_{n=1}^{12} OA_{ij}(n)$ has no more than 32 values, are new.

Finally, we deduce by (9)–(14) that

$$\mathfrak{M}(s') = -\left|\sum_{n=1}^{4} D_{ij}(n)\right| - \left|\sum_{n=5}^{8} D_{ij}(n)\right| - \left|\sum_{n=9}^{12} D_{ij}(n)\right| + \sum_{n=1}^{12} OA_{ij}(n) + \nabla^{\min}(\boldsymbol{c}_m) \cdot P(\boldsymbol{c}_m) \quad (15)$$

where

$$\nabla^{\min}(\boldsymbol{c}) = \min\left\{\nabla^{\min}(\boldsymbol{c}^v);\ v = 1, 2, 3\right\}, \quad (16)$$

and the definitions of $\boldsymbol{c}_m$ and $\nabla^{\min}(\boldsymbol{c}^v)$ are the following. $\nabla^{\min}(\boldsymbol{c})$ reflects the fact that while performing the absolute value operations in (15), we are choosing between $\boldsymbol{c}^v$ and $\bar{\boldsymbol{c}}^v$, $v = 1, 2, 3$ according to the minimum between $\Sigma_{n=4v-3}^{4v} D_{ij}(n)$ and $-\Sigma_{n=4v-3}^{4v} D_{ij}(n)$, and we are also choosing between the subtotal parity $P(\boldsymbol{c}^v)$ and the subtotal parity $P(\bar{\boldsymbol{c}}^v)$. Denote by $\boldsymbol{c}_m^v$ that value of $\boldsymbol{c}^v$ for which $\Sigma_{n=4v-3}^{4v} D_{ij}(n) \le 0$, and let $\boldsymbol{c}_m = (\boldsymbol{c}_m^1, \boldsymbol{c}_m^2, \boldsymbol{c}_m^3)$. Denote $\Delta^{\min}(\boldsymbol{c}^v) = \min\{\Delta_{ij}(n),\ n = 4v-3, \cdots, 4v\}$ for $v = 1, 2, 3$, and let

TABLE III
64 POSSIBLE VALUES OF $\pm\left|\Sigma_{n=4v-3}^{4v} D_{ij}(n)\right|$

	v=1: s_4 s_6 s_7 s_{10} s_{11} s_{12} v=2: $s_4 \oplus s_5$ $s_5 \oplus s_8$ $s_7 \oplus s_9$ v=3: s_5 s_8 s_9	$z_q'(v)$
1	0 0 0 0 0	$D_{00}(4v-3) + D_{00}(4v-2) + D_{00}(4v-1) + D_{00}(4v)$
2	0 0 1 0 0 1	$-[D_{00}(4v-3) + D_{00}(4v-2) + D_{00}(4v-1) - D_{00}(4v)]$
3	1 0 0 0 0 0	$-[D_{00}(4v-3) + D_{00}(4v-2) - D_{00}(4v-1) - D_{00}(4v)]$
4	1 0 1 0 0 1	$-[D_{00}(4v-3) + D_{00}(4v-2) - D_{00}(4v-1) + D_{00}(4v)]$
5	1 1 0 0 0 0	$D_{00}(4v-3) - D_{00}(4v-2) + D_{00}(4v-1) + D_{00}(4v)$
6	1 1 1 0 0 1	$-[D_{00}(4v-3) - D_{00}(4v-2) - D_{00}(4v-1) - D_{00}(4v)]$
7	0 1 0 0 0 0	$-[D_{00}(4v-3) - D_{00}(4v-2) + D_{00}(4v-1) - D_{00}(4v)]$
8	0 1 1 0 0 1	$-[D_{00}(4v-3) - D_{00}(4v-2) + D_{00}(4v-1) + D_{00}(4v)]$
9	0 0 1 0 0 0	$D_{10}(4v-3) + D_{10}(4v-2) + D_{10}(4v-1) + D_{10}(4v)$
10	0 0 0 0 0 1	$-[D_{10}(4v-3) + D_{10}(4v-2) + D_{10}(4v-1) - D_{10}(4v)]$
11	1 0 1 0 0 0	$-[D_{10}(4v-3) + D_{10}(4v-2) - D_{10}(4v-1) - D_{10}(4v)]$
12	1 0 0 0 0 1	$-[D_{10}(4v-3) + D_{10}(4v-2) - D_{10}(4v-1) + D_{10}(4v)]$
13	1 1 1 0 0 0	$D_{10}(4v-3) - D_{10}(4v-2) + D_{10}(4v-1) + D_{10}(4v)$
14	1 1 0 0 0 1	$-[D_{10}(4v-3) - D_{10}(4v-2) - D_{10}(4v-1) - D_{10}(4v)]$
15	0 1 1 0 0 0	$-[D_{10}(4v-3) - D_{10}(4v-2) + D_{10}(4v-1) - D_{10}(4v)]$
16	0 1 0 0 0 1	$-[D_{10}(4v-3) - D_{10}(4v-2) + D_{10}(4v-1) + D_{10}(4v)]$
17	1 1 0 0 1 0	$D_{10}(4v-3) + D_{10}(4v-2) + D_{00}(4v-1) + D_{00}(4v)$
⋮		⋮
25	1 1 1 0 1 0	$D_{00}(4v-3) + D_{00}(4v-2) + D_{10}(4v-1) + D_{10}(4v)$
⋮		⋮
33	1 0 0 1 0 0	$D_{10}(4v-3) + D_{00}(4v-2) + D_{10}(4v-1) + D_{00}(4v)$
⋮		⋮
41	1 0 1 1 0 0	$D_{00}(4v-3) + D_{10}(4v-2) + D_{00}(4v-1) + D_{10}(4v)$
⋮		⋮
49	0 1 0 1 1 0	$D_{00}(4v-3) + D_{10}(4v-2) + D_{10}(4v-1) + D_{00}(4v)$
⋮		⋮
57	0 1 1 1 1 0	$D_{10}(4v-3) + D_{00}(4v-2) + D_{00}(4v-1) + D_{10}(4v)$
58	0 1 0 1 1 1	$-[D_{10}(4v-3) + D_{00}(4v-2) + D_{00}(4v-1) - D_{10}(4v)]$
59	0 0 1 1 1 0	$-[D_{10}(4v-3) + D_{00}(4v-2) - D_{00}(4v-1) - D_{10}(4v)]$
60	0 0 0 1 1 1	$-[D_{10}(4v-3) + D_{00}(4v-2) - D_{00}(4v-1) + D_{10}(4v)]$
61	1 0 1 1 1 0	$D_{10}(4v-3) - D_{00}(4v-2) - D_{00}(4v-1) + D_{10}(4v)$
62	1 0 0 1 1 1	$-[D_{10}(4v-3) - D_{00}(4v-2) - D_{00}(4v-1) - D_{10}(4v)]$
63	1 1 1 1 1 0	$-[D_{10}(4v-3) - D_{00}(4v-2) + D_{00}(4v-1) - D_{10}(4v)]$
64	1 1 0 1 1 1	$-[D_{10}(4v-3) - D_{00}(4v-2) + D_{00}(4v-1) + D_{10}(4v)]$

$$\nabla^{\min}(c^v) = \begin{cases} \min\left\{\Delta^{\min}(c^v),\ 2\left|\sum_{n=4v-3}^{4v} D_{ij}(n)\right| + \Delta^{\min}(\bar{c}^v)\right\}; \\ \quad \text{if } P(c^v) = P(\bar{c}^v) \\ \min\left\{\Delta^{\min}(c^v),\ 2\left|\sum_{n=4v-3}^{4v} D_{ij}(n)\right|\right\}; \\ \quad \text{if } P(c^v) \neq P(\bar{c}^v). \end{cases} \tag{17}$$

The meaning of (17) is that, whenever $P(c^v) = P(\bar{c}^v)$ for a chosen c^v, the penalty incurred by inverting the parity is the minimum between (a) the penalty for inverting the parity associated with c^v and (b) the penalty for replacing the chosen c^v by $\bar{c}^v$ plus the penalty for inverting the parity associated with $\bar{c}^v$. When $P(c^v) \neq P(\bar{c}^v)$ we refrain from adding the penalty for inverting the parity associated with $\bar{c}^v$ because, by changing the chosen c^v to be $\bar{c}^v$, the parity is inverted at the same instant. Equations (15)–(17) imply the maximum likelihood decoding algorithm stated and discussed in the next section. The ideas appearing in [4] have some similarity with those expressed by (15)–(17), but there they are related to trellis-based decoding of the Leech lattice. Furthermore, it seems that the fixed structure of the trellis (i.e., its independence of the received vector) prevents a full utilization of these ideas, especially with respect to the reduction of the average decoding complexity, as opposed to what is achieved by the decoder proposed here.

III. The Decoding Algorithm

The decoding algorithm has two stages, a precomputation stage and a main stage. The precomputation stage is composed of three major steps and the main stage is composed of two major steps, as follows.

Decoding Algorithm

1) Precomputation Stage:

a) For each of the 12 sets of the 8 $d_{ijk}(n)$, $n = 1, 2, \cdots, 12$, compute:

a.1) the set of 4 $d_{ij}(n)$ and the corresponding set of 4 $\Delta_{ij}(n)$, and

a.2) the set of 2 $D_{ij}(n)$ and the corresponding set of 2 $OA_{ij}(n)$.

b) For each $v = 1, 2, 3$, corresponding to the three vertical sections of G, compute:

b.1) the 8 possible values of $\Sigma_{n=4v-3}^{4v} OA_{ij}(n)$,

b.2) the 64 possible values of $|\Sigma_{n=4v-3}^{4v} D_{ij}(n)|$,

b.3) the corresponding 64 values of $\nabla^{\min}(c_m^v)$, and

b.4) the 64 subtotal parities $P(c_m^v)$.

Store the 64 signs of $\Sigma_{n=4v-3}^{4v} D_{ij}(n)$, denoted by $\operatorname{sgn}(c^v)$, before taking the absolute value.

c) Compute the 32 possible values of $\Sigma_{n=1}^{12} OA_{ij}(n)$.

2) Main Stage:

d) For each of the $2^9 = 512$ source-words s' compute $\mathfrak{M}(s')$ according to (15). Find that s', denoted $s'_{\min}$, which minimizes $\mathfrak{M}(s')$, and obtain the corresponding $c'_{\min}$. Subsequently, using $\operatorname{sgn}(c^v)$; $v = 1, 2, 3$ at $c'_{\min}$, get $c_{\min}$.

e) Use $c_{\min}$ to determine the best A sequence in the same way as it was determined by the straightforward decoding rule of (5) and (6). (If the overall parity-check fails, the easy way to get the lth of the k indexes, which must be inverted, is to recompute $\Delta^{\min}(c_{\min})$.)

For the purpose of estimating the computational and memory complexity of the algorithm, we shall count the number of arithmetic operations and the number of memory words required by each step. These figures are intended for comparison only. The actual computation time and memory space are highly implementation-dependent. Following [3]–[5] we count an addition, a subtraction and a comparison of two words as one operation, regardless of the word length. Also, as done in [3]–[5], we ignore the complexity of computation of an absolute value, evaluation of a parity check, and access-time to memory. We shall also assume, as in [3], that all the distances $d_{ijk}(n)$ or, alternatively, their normalized versions $D_{ij}(n)$ along with $OA_{ij}(n)$ and $\Delta_{ij}(n)$ are precomputed and available. All these values can be computed directly from the geometry of the signal constellation (like that of Fig. 3) of the symbols used for transmission (see also, [4]). The determination of these distances depends on the implementation of the signaling scheme, and since it has a complexity of the order of a small multiple of 96 [3], we shall not include, as in [3] and [4], the complexity of this step, step a) of the algorithm, in the total complexity. (A rough estimate for the worst case of this "small multiple" of 96 is about 500 operations, which has to be added to the total complexity of the Λ_{24} decoder given in the sequel.) When assessing the memory requirement, we consider any constant or variable that has to be stored, as one word with a fixed number of bits, but neglect one-bit variables like parity and sign.

For performing step b.1) efficiently, it is rewarding to compute first all possible values of sum of pairs of $OA_{ij}(n)$, i.e., $OA_{ij}(4v-3) + OA_{gh}(4v-2)$ and $OA_{ij}(4v-1) + OA_{gh}(4v)$, $v = 1, 2, 3$, denoted in Table I by $x_q(v)$ and $y_q(v)$, respectively. Then, with the aid of Table II, compute the 8 possible values of $\Sigma_{n=4v-3}^{4v} OA_{ij}(n)$, denoted by $z_q(v)$. (The idea of computing first the sum of 2-tuples and the 4-tuples appears also in [3] and [4].) Thus, step b.1) requires, for each v, $4 + 4 + 8 = 16$ additions.

For step b.2) we utilize tables similar to Tables I and II, but now for computing the 8 "leaders" of sums of 4-tuples of $D_{ij}(n)$, namely, items 1, 9, 17, 25, 33, 41, 49, and 57 in Table III. This requires 16 additions. By exploiting the Gray-code ordering in Table III, we determine the seven other values of $\pm|\Sigma_{n=4v-3}^{4v} D_{ij}(n)|$ per leader. The leaders and those terms whose sign is changed are typed in Table III by boldface letters. This computation requires a total of $7 \cdot 8 = 56$ additions. Thus step b.2) requires altogether $16 + 56 = 72$ additions for each v.

The best way to compute $\Delta^{\min}(c_m^v)$ in step b.3) is the following. Create an ordered list of the $4 \cdot 4 = 16$ $\Delta_{ij}(n)$, $n = 4v-3, \cdots, 4v$, by $\Sigma_{i=1}^{16} \lceil \log(3i/4) \rceil = 46$ comparison operations [15]. With the aid of this

ordered list we get immediately all the required $\Delta^{\min}(\boldsymbol{c}^v)$ for (17). Then, using (17), determine each of the 64 values of $\nabla^{\min}(\boldsymbol{c}_m^v)$ by at least one comparison [if $P(\boldsymbol{c}^v) \neq P(\bar{\boldsymbol{c}}^v)$], and at most one comparison and one addition [if $P(\boldsymbol{c}^v) = P(\bar{\boldsymbol{c}}^v)$]. Thus, step b.3) requires $46 + 64 = 110$ comparisons and an extra 64 additions if $P(\boldsymbol{c}^v) = P(\bar{\boldsymbol{c}}^v)$, or a total at most 174 operations. Thus the total complexity of the step b) is $3 \cdot (16 + 72 + 174) = 786$ operations at most, and $3 \cdot (16 + 72 + 110) = 594$ operations at least.

Step c) requires $2 \cdot 32 = 64$ additions. Thus the complexity of the precomputation stage totals 850 operations at most and 658 operations at least.

Instead of computing each of the 512 possible $\mathfrak{M}(\boldsymbol{s}')$, as prescribed by step d) of the algorithm, it is more efficient to postpone the summation of the term $\Sigma_{n=1}^{12} OA_{ij}(n)$ in (15) until the minimization over s_4, s_5, s_6, and s_8 is completed. Accordingly, 2 additions are required for a given $\boldsymbol{s}'$, provided that $P(\boldsymbol{c}_m) = 0$. Otherwise, 2 additional comparisons are required for computing $\nabla^{\min}(\boldsymbol{c}_m)$, according to (16), and one extra addition is required in (15). For minimizing $\mathfrak{M}(\boldsymbol{s}')$ 511 comparisons plus 32 additions are required, the latter for adding the term $\Sigma_{n=1}^{12} OA_{ij}(n)$ after each minimization over s_4, s_5, s_6, and s_8. Thus, step d) requires at least $512 \cdot 2 + 32 = 1056$ additions plus 511 comparisons, a total of 1567 operations, and at most $512 \cdot 3 + 32 = 1568$ additions plus $512 \cdot 2 + 511 = 1535$ comparisons, or a total of 3103 operations at most. Step e) requires at most 11 operations for computing $\Delta^{\min}(\boldsymbol{c}_{\min})$ by using (7). In summary, given the normalized distances, the proposed decoder requires for decoding the best of the A and B sequences $2 \cdot (3103 + 850 + 11) + 1 = 7929$ operations at most, $2 \cdot (1567 + 658) + 1 = 4451$ operations at least, and about 6200 operations on the average. Compare these figures to 55968 operations in [5], 15167 operations in [3], and about 10000 operations in [4].

Regarding the memory complexity, we obtain the following figures. For exploiting the structure of Table III in the precomputation stage, we have to compute and store in memory at least 1/4 of this table for each of $v = 1, 2$ and one entry for $v = 3$. Furthermore, we have to compute and store 1/4 of the 32 values of $\Sigma_{n=1}^{12} OA_{ij}(n)$. This requires $16 + 16 + 1 + 8 = 41$ memory words. Accordingly, we have to store $16 + 16 + 1 = 33$ values of $\nabla^{\min}(\boldsymbol{c}_m)$ and the three ordered-lists which occupy $3 \cdot 16 = 48$ memory words. Thus, a total of at least 122 memory words are required (excluding the 96 memory words required for storing the normalized distances). In comparison, the trellis decoders in [3] and [4] seem to require at least a small multiple of the number of states in the trellis, which is 256. The memory space for storing the tables of the decoder in [5] is even larger. Thus, a gain factor of more than 2 is achieved by the proposed decoder.

IV. Further Improvements and Conclusions

The main idea of Section III is to perform an independent partial minimization over s_1, s_2, and s_3, by exploiting the structure of the first three rows of G. This idea can be carried further through utilization of the first five rows of G, as follows. Let $s_e = s_4 \oplus s_5$, and assume for a while that s_4, s_e, and s_5 can be chosen independently for minimizing $\mathfrak{M}(\boldsymbol{s}')$. This assumption is compensated by the introduction of an additional parity constraint that has to be checked, namely,

$$s_4 \oplus s_e \oplus s_5 = 0. \tag{17}$$

Accordingly, we split each of the three terms $\Sigma_{n=4v-3}^{4v} D_{ij}(n)$, $v = 1, 2, 3$, in (15), and obtain the six terms $\Sigma_{n=2u-1}^{2u} D_{ij}(n)$, $u = 1, 2, \cdots, 6$. Subsequently, we minimize $\mathfrak{M}(\boldsymbol{s}')$ over s_4, s_e, s_5 by performing an absolute value. Denote this minimum by

$$\underline{\mathfrak{M}}(\boldsymbol{s}'') = \min_{s_4, s_e, s_5} \left\{ \mathfrak{M}(\boldsymbol{s}') \right\} \tag{18}$$

where $\boldsymbol{s}'' = (s_6, s_7, \cdots, s_{12})$. But now, when evaluating $\underline{\mathfrak{M}}(\boldsymbol{s}'')$ in the main stage for each of the $2^7 = 128$ possible values of $\boldsymbol{s}''$, two parity constraints have to be checked. Consequently, three, different $\nabla^{\min}(\boldsymbol{c}_m)$ have to be computed in case one of the two, or both, parity checks fail. Thus, additional precomputations are required. Careful investigation of the proposed idea leads to the conclusion that for decoding the best A sequence approximately 200 extra operations are required in the precomputation stage while the complexity of the main stage is reduced by at least 1100 operations in the worst case. Thus, a total saving of at least $2 \cdot (1100 - 200) = 1800$ operations is achieved. Therefore, the total complexity of the improved Leech lattice decoder is at most ~ 6100 operations and about 5000 on the average. This yields gain factors of almost 10 over [5], 2.5 over [3] and about 2 over [4].

Following the same line of thought, one might be tempted to exploit the structure of additional rows of G. Promising candidates are the 6th and the 8th rows, in addition to the first five rows. However, it turns out that for this modification of the algorithm, the increase in the precomputations overweighs the decrease in the complexity of the main stage. This phenomenon is attributable to the sharp increase in the number of different $\nabla^{\min}(\boldsymbol{c}_m)$ that exist in case of three parity checks, namely $2^3 - 1 = 7$ per item in Table III, instead of $2^2 - 1 = 3$. Nonetheless, for other codes [16] and lattices it may be worthwhile to work with more than two parity constraints.

It is interesting to compare the complexities of the decoder proposed for the Leech lattice and decoders for the state of the art trellis-coded modulation schemes with the same effective coding gain. Although the nominal coding gain of the Leech lattice is 6.02 dB [3], the effective coding gain is not so high since the error coefficient is very large. More explicitly, the error probabilities for coded systems on Gaussian channels are typically of the form $P(E) = K \cdot \exp(-E)$ where E is governed by the nominal coding gain, and the error coefficient K is of the order of the number of coded sequences at minimum distance from an average transmitted sequence [2] (see also, [8, ch. 3 Sections 1.3 and 1.4]). Thus, the number of nearest

TABLE IV
COMPARISON TABLE OF TRELLIS CODES

Code	Dim.	States	$\tilde{N}_D$	γ (db)	$\tilde{N}_0$	γ_{eff} (db)
Wei	16	64	352	5.64	796	4.12
Ungerboeck	1	64	384	5.44	72	4.61
Ungerboeck	2	64	456	5.44	56	4.68
Calderbank–Sloane	2	64	464	5.44	48	4.72
Calderbank–Sloane	4	64	512	6.02	828	4.48
Wei	8	64	584	6.02	316	4.76
Wei	16	64	600	5.64	412	4.31
Ungerboeck	1	128	768	6.02	132	5.01
Ungerboeck	2	128	902	6.02	344	4.74
Calderbank–Sloane	2	128	912	6.02	228	4.85

neighbors N_0 (called also the kissing number [8]) is frequently used for approximating of the error coefficient. For Λ_{24} the normalized N_0, denoted by $\tilde{N}_0$ and equal to $N_0/12$ (normalized to two dimensions), is $196560/12 = 16380$ [8], [9]. Forney [9] uses the following rule of thumb as a measure of the effective coding gain of trellis codes: an increase by a factor of two in the error coefficient reduces the coding gain by about 0.2 dB (at error rates of the order of 10^{-6}). But, as stated in [9], this measure is only approximately valid for moderately low error rates, and generally does not take into account neighbors other than the nearest. For lattice codes, including Λ_{24}, an effective coding gain is not given in [9], because the aforementioned rule of thumb is questionable when the number of nearest neighbors is very large. Therefore, for comparison we list in Table IV some trellis codes with about the same normalized (to two dimensions) decoding complexity $\tilde{N}_D$ and about the same nominal coding gain γ that Λ_{24} has, along with their normalized error coefficients and their effective coding gain γ_{eff}. For Λ_{24} $\tilde{N}_D \cong 6000/12 = 500$ operations at most, and about 400 on the average, including the refinements. The material is accumulated from the tables of [9] and the results about decoding complexity stated there; the latter seem to be based on the same assumptions used here for estimating the decoding complexity. The codes in Table IV are ordered according to their increasing decoding complexity (other parameters of these codes may also be found in [9]).

One of the Wei's schemes, which are highly suited for implementation, is used in a Codex 19.2 kbit/s modem, as stated in [9]. Another Codex 19.2 kbit/s modem [4], which has a performance comparable to that of the previous one [10], employs a Leech lattice code. According to Table IV the decoding complexities of both schemes are also about the same. This example does not support the common "folk theorem" about the preferability of convolutional codes over block codes.

In [4], two Analog Devices ADSP2100 digital signal processors were used for programming the Leech lattice encoder–decoder. At 19.2 kbits/s information speed 70 percent of the processing capability of the two processors was utilized, and 45 percent of the capability was consumed by the decoder of the Leech lattice. Because the complexity of our Leech lattice decoder is only slightly more than one half of that of [4] (and thus may require only about 25 percent of the two ADSP2100 processing capability), it seems that such a Leech lattice modem can be implemented by a single digital signal processor. Furthermore, regarding the implementation of the improved decoder in a VLSI chip, it is conjectured [18] that an information speed of the order of a few million symbols per second can be achieved, assuming 1.25 micron CMOS technology with about 100 000 transistors. Thus, the Leech lattice is a promising candidate for coded modulation.

ACKNOWLEDGMENT

The authors are grateful to G. D. Forney, Jr., for many helpful comments, and for preprints of his papers, to G. R. Lang and F. M. Longstaff for preprints of their algorithm, and to the referees for their suggestions that helped to improve the presentation of this paper.

REFERENCES

[1] J. Leech, "Notes on sphere packing," *Can. J. Math*, vol. 19, pp. 251–267, 1967.
[2] G. D. Forney, Jr., R. G. Gallager, G. R. Lang, F. M. Longstaff, and S. U. Qureshi, "Efficient modulation for band-limited channels," *IEEE J. Select. Area Commun.*, vol. SAC-2, pp. 632–647, 1984.
[3] G. D. Forney, Jr., "Coset codes II: Binary lattices and related codes," *IEEE Trans. Inform. Theory*, vol. 34, pp. 1152–1187, 1988.
[4] G. R. Lang and F. W. Longstaff, "A Leech lattice modem," see this issue, pp. 968–973.
[5] J. H. Conway and N. J. A. Sloane, "Soft decoding techniques for codes and lattices, including the Golay code and the Leech lattice," *IEEE Trans. Inform. Theory*, vol. IT-32, pp. 41–50, 1986.
[6] ——, "On the Voronoi regions of certain lattices," *SIAM J. Algebraic Discrete Methods*, vol. 5, pp. 294–305, 1984.
[7] G. Ungerboeck, "Channel coding with multilevel/phase signals," *IEEE Trans. Inform. Theory*, vol. IT-28, pp. 55–67, 1982.
[8] J. H. Conway and N. J. A. Sloane, *Sphere Packing, Lattices and Groups*. New York: Springer-Verlag, 1988.
[9] G. D. Forney, Jr., "Coset codes I: Introduction and geometrical classification," *IEEE Trans. Inform. Theory*, vol. 34, pp. 1123–1151, 1988.
[10] ——, "A bounded-distance decoding algorithm for the Leech lattice, with generalizations," preprint.
[11] J. Leech and N. J. A. Sloane, "Sphere packing and error correcting codes," *Can. J. Math.*, vol. 23, pp. 718–745, 1971.
[12] Y. Be'ery and J. Snyders, "Optimal soft decision block decoders based on the Hadamard transform," *IEEE Trans. Inform. Theory*, vol. IT-32, pp. 355–364, 1986.
[13] ——, "Soft decoding of the Golay codes," presented at the IEEE Inform. Theory Workshop, Bellagio, Italy, June 1987.
[14] N. J. A. Sloane, "Tables of sphere packings and spherical codes," *IEEE Trans. Inform. Theory*, vol. IT-27, pp. 327–338, 1981.
[15] D. E. Knuth, *The Art of Computer Programming, Vol. 3*. Reading, MA: 1973, pp. 181–190.
[16] J. Snyders and Y. Be'ery, "Maximum likelihood soft decoding of binary block codes and decoders for the Golay codes," *IEEE Trans. Inform. Theory*, to be published.
[17] R. A. Silverman and M. Balser, "Coding for a constant data rate source," *IRE Trans. Inform. Theory*, vol. PG IT-4, pp. 50–63, 1954.
[18] B. Shahar and Y. Be'ery, "VLSI Architectures for Golay and Leech decoders," Tel-Aviv Univ., Research Rep., 1988.

Yair Be'ery was born in Tel-Aviv, Israel. He received the B.Sc., M.Sc., and Ph.D. degrees in electrical engineering from Tel-Aviv University, Tel Aviv, Israel, in 1979, 1979, and 1985, respectively.

He is currently with the Department of Electronic Communications, Control, and Computer Systems, Tel-Aviv University, Tel-Aviv, Israel. From 1979 to 1985 he was involved in research and development of digital signal processing systems and design of VLSI architectures for special purpose processors. He received the Eliyahu Golomb Award from the Ministry of Defense, Israel, in 1984, and the Rothschild Fellowship for postdotoral studies at the Rensselaer Polytechnic Institute, Troy, NY, in 1986. His research interests include error control coding, VLSI architectures and systolic arrays, and adaptive algorithms for speech and data transmission.

Boaz Shahar was born in Haifa, Israel. He received the B.Sc. degree in electrical engineering from Tel-Aviv University, Tel-Aviv, Israel, in 1984. He is presently studying towards the M.Sc. degree at Tel-Aviv University.

From 1984 to 1988 he was involved in developing the NS32532 microprocessor and VLSI testing methodologies at National Semiconductor (Israel). His current research interest is VLSI implementation of decoders for error control codes.

Jakov Snyders received the B.Sc., M.Sc., and D.Sc. degrees in electrical engineering from the Technion—Israel Institute of Technology, Haifa, in 1966, 1969, and 1972, respectively.

Since 1974 he has been with Tel-Aviv University, Tel-Aviv Israel. He has held visiting positions at the University of Toronto, University of California—Los Angeles, McGill University, Montreal, P.Q., Canada, and the Rensselaer Polytechnic Institute, Troy, N.Y. His research interests include error control coding, spread spectrum and multiple user communication, and stochastic processes and systems.