RESEARCH ARTICLE

ScienceAsia 35 (2009): **95–105**

doi: 10.2306/scienceasia1513-1874.2009.35.095

Applications of Gröbner bases to the structural description and realization of multidimensional convolutional code

Chalie Charoenlarpnopparut

School of Information, Computer and Communication Technology, Sirindhorn International Institute of Technology, Thammasat University, Klong Luang, Pathum Thani 12121, Thailand

e-mail: chalie@siit.tu.ac.th

Received 25 Feb 2008
Accepted 4 Mar 2009

ABSTRACT: Over the past few years, multidimensional convolutional code has become an emerging area of research in the signal processing community. While one-dimensional convolutional code and its variants have been thoroughly understood, the m-D counterpart still lacks unified notation and efficient encoding/decoding implementation. Here, the strong link between the theory of Gröbner bases and m-D convolutional code is explored. Several applications of Gröbner bases to the characterization of m-D convolutional encoders are proposed. Furthermore, the more practical problem of minimal encoder realization is discussed and an algebraic algorithm based on the use of Gröbner bases is provided. From the implementation point of view, the syndrome decoder is currently the only means for decoding m-D convolutional code. A constructive method for computing the syndrome decoding matrix using the theory of syzygy modules is proposed.

KEYWORDS: minimal encoder, syndrome decoding, syzygy, unimodular matrix

INTRODUCTION

Examples of 1-D information are voice signals, audio signals, node voltages, and line currents. By sampling these signals and quantizing the samples, one obtains discrete sequences of binary representations. On the other hand, multidimensional (m-D, $m \geqslant 2$) information may be taken from 2-D images (e.g., MRI film, photos), 3-D images (e.g., holograms, animation of 2-D images, video), and animated 3-D images ($m = 4$). To encode multidimensional information, it is conventional to first transform m-D data arrays into 1-D sequences by means of scanning, and then apply the 1-D encoding technique. This encoding procedure is, in fact, unnatural and ignores the correlation in all other directions except in the scanning direction.

To motivate the importance of multidimensional convolutional code, note that 1-D convolutional code is a generalization of a block code. A block code is a 1-D convolutional code with the delay variable D replaced by zero. Equivalently, 0-D convolutional code is essentially a block code. As a result, a multidimensional convolutional code can be viewed as a generalization of all linear codes.

While 1-D convolutional code and its variants have been thoroughly understood[1–6], the m-D counterpart still lacks unified notation and efficient implementation. In this paper, the partial aim is to emphasize the recent development[7–9] of m-D convolutional code theory as well as to point out the bottleneck and open problems to the multidimensional community. The main goal is to demonstrate the usefulness of the theory of Gröbner bases in the field of convolutional code which has not been done elsewhere.

The theory of Gröbner bases was first introduced in 1965 by Bruno Buchberger in his doctoral dissertation and the name Gröbner was used to honour his thesis advisor. Since then the theory of Gröbner bases has been applied to a wide range of engineering problems[10], e.g., in the field of computer algebra for solving multivariate polynomial equations, integer programming problems, in the systems and control area for solving controller design problems, and in the signal processing area for solving m-D filter bank design and ladder decomposition problems. Gröbner basis theory has also been successfully applied to classical coding theory problems, such as the decoding of cyclic codes[11,12].

This article is organized as follows. First, the basic definition and conventions in m-D system theory are given and briefly explained. In the following section, the properties of m-D convolutional code and its encoder are defined. Then, by using the theory of Gröbner bases, the algorithms for testing

the existence and characterizing those properties are presented. It will be evident that the application of Gröbner bases and its variants will ultimately define a strong link between the m-D convolutional code and system theory. Since the encoder matrix of a 1-D convolutional code can be represented by a rectangular univariate polynomial matrix, in the m-D case, the representation of encoder matrix becomes the multivariate polynomial matrix, where the theory of Gröbner bases for modules[13] is readily applicable. Each row of this generator matrix can be realized (implemented in the form of hardware) separately and thus it has row independence.

In the later part of the paper, the problem of minimal realization[14–16] is considered. Partial solutions by direct sequential search and algebraic reduction using the Gröbner basis for modules are provided. Various algorithms related to counting the number of delay elements are given. In the last section, the decoding process of multidimensional code by means of syndrome decoding is discussed, and with the help of Gröbner bases, the construction procedure of the decoding matrix is explained.

Representation of multidimensional signals

In general, there are two approaches to representing multidimensional signals. In the geometrical approach, an m-D signal is represented by a m-D finite matrix. For example, a 3×3 pixel image I with 16 grey-scale levels (4 bits) can be represented as

$$I = \begin{bmatrix} 1100 & 0101 & 1001 \\ 1000 & 1001 & 0000 \\ 1011 & 0100 & 0010 \end{bmatrix}. \qquad (1)$$

In the algebraical approach, an m-D signal is represented by a m-variate polynomial vector of size $1 \times b$ where b is the number of symbols used for representing each sample. For example, the 2-D image in the previous example ($b = 4, m = 2$) may be represented by

$$u = \begin{bmatrix} 1 + D_2 + D_1^2 + D_1D_2 + D_2^2 \\ 1 + D_1 + D_1D_2^2 \\ D_2^2 + D_1^2D_2^2 \\ D_1 + D_1^2 + D_1D_2 + D_2^2 \end{bmatrix}^{\mathrm{T}}. \qquad (2)$$

Note that in the above example, the ith-row jth-column element of the matrix I in (1) is associated with the multiplying factor $D_1^{j-1}D_2^{i-1}$ and the pth element of the vector u in (2) is taken from the summation of all pth symbols, of all elements of the matrix I multiplied by their associated monomials.

Notation and definition

Let $\mathbb{F} = \mathbb{F}_q$ be the finite field with q elements and $R = \mathbb{F}[D_1, D_2, \ldots, D_m]$ be the ring of m-variate polynomials whose coefficients belong to the field $\mathbb{F}$. Let $n \geqslant 1$ be a positive integer. Then, the free R-module (p.114 in Ref. 13) R^n is the m-variate polynomial row vector space of length n over $\mathbb{F}$,

$$R^n = \left\{ \begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix} | v_i \in R, i = 1, \ldots, n \right\}.$$

Definition 1 An $m-$dimensional convolutional code of length n over $\mathbb{F}$ is an R-submodule $\mathcal{C} \subseteq R^n$. An element $w \in \mathcal{C}$ is called a *codeword*.

Since R is Noetherian, every m-dimensional convolutional code is finitely generated as an R-module, i.e., there exists a finite set of row vectors $g_1, g_2, \ldots, g_k \in R^n$ such that $\mathcal{C} = \left\{ \Sigma_{i=1}^{k} u_i g_i | u_i \in R \right\}$.

CONVOLUTIONAL ENCODER

In this section, the representation of the convolutional encoder is given in terms of a generator matrix which is an m-D polynomial matrix. This representation is suitable for encoder equivalence testing and the determination of the convolutional decoder.

Generator matrix

As in the 1-D case, a generator matrix of an m-D convolutional code can be viewed as a transfer function matrix of the encoder and it is often the centre of focus in the encoder design and realization stage.

Definition 2 Let G be an $k \times n$ $(k \leqslant n)$ m-variate polynomial matrix constructed row-wise from a set of row vectors $\{g_i\}_{i=1}^{k}$:

$$G = \begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_k \end{bmatrix}.$$

If an m-D convolutional code $\mathcal{C} =$ rowspace(G), then the polynomial matrix $G \in R^{k \times n}$ is called a *generator matrix* of $\mathcal{C}$.

The existence of G is guaranteed since any m-D convolutional code is a finitely generated R-module. Furthermore, if G is of full row rank (i.e., rank(G) $= k$), then the rate of $\mathcal{C}$ is $\frac{k}{n}$. Note that it is desirable for a generator matrix to have full row rank in order to avoid having different input sequences mapped to the same codeword, which may cause confusion for the decoder regarding the actual transmitted message.

Definition 3 If a code $\mathcal{C}$ is free of rate $\frac{k}{n}$, then $\mathcal{C} =$ rowspace(G) for some $G \in R^{k\times n}$ and G is called an *encoder* of the code $\mathcal{C}$.

It is well-known that an R-module can be generated by many different sets of module generators. Analogously, a given m-D convolutional code $\mathcal{C}$ can be generated by more than one encoder. This class of generators that generate the same set of code is defined as a class of *equivalent* generators.

Definition 4 An m-variate polynomial matrix $U \in R^{k\times k}$ is *unimodular* if U is a unit in $R^{k\times k}$. In other words, U is unimodular if and only if $\det(U)$ is a unit in R (i.e., a nonzero element of $\mathbb{F}$).

Proposition 1 [9] *Let $G,\ G_1 \in R^{n\times k}$ be of full row rank. G and G_1 are equivalent encoders if and only if there exists a unimodular matrix $U \in R^{k\times k}$ such that $U \cdot G = G_1$.*

From Proposition 1, given two generator matrices G_1 and $G_2 \in R^{n\times k}$, the following algorithm can be used to determine whether they are equivalent. Furthermore, it can be used to compute the unimodular matrix $U \in R^{k\times k}$ such that $U \cdot G_1 = G_2$.

Algorithm 1 Equivalence test

Step 1: Construct modules M_1 and M_2 generated by the rows of matrices G_1 and G_2 respectively.

Step 2: Compute the (completely) reduced Gröbner bases of M_1 and M_2 with respect to the same ordering by using Buchberger's algorithm for modules (see p.149 in Ref. 13).

Step 3: If the Gröbner bases are identical, then G_1 and G_2 are equivalent. Furthermore, the unimodular matrix U can be constructed by retracing the steps of the division algorithm (see example 3.6.1 in Ref. 13).

Note that an alternative approach is to apply the theory of Gröbner bases to show that every row of G_1 is a member of the module generated by G_2 and vice-versa.

Example 1 Verify that the encoders

$$G_1 = \begin{bmatrix} 1 & 0 & D_1 \\ D_2 & D_2 & 1+D_1D_2 \end{bmatrix},$$

$$G_2 = \begin{bmatrix} 1 & D_1D_2 & 0 \\ 0 & D_2 & 1 \end{bmatrix}$$

are equivalent. *Solution*: Let M_1 and M_2 be the modules generated by all rows of G_1 and G_2 respectively, i.e.,

$$M_1 = \left\langle \begin{bmatrix} 1 & 0 & D_1 \end{bmatrix}, \begin{bmatrix} D_2 & D_2 & 1+D_1D_2 \end{bmatrix} \right\rangle$$
$$M_2 = \left\langle \begin{bmatrix} 1 & D_1D_2 & 0 \end{bmatrix}, \begin{bmatrix} 0 & D_2 & 1 \end{bmatrix} \right\rangle.$$

Computation of Gröbner bases $\mathcal{G}_1, \mathcal{G}_2$ of modules M_1 and M_2 with respect to degree lexicographical ordering yields

$$\mathcal{G}_1 = \mathcal{G}_2 = \left\{ \begin{bmatrix} 0 & D_2 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & D_1 \end{bmatrix} \right\}.$$

Hence the encoders G_1 and G_2 are equivalent. Furthermore, by retracing the steps in the computation of Gröbner bases, one can show that $G_2 = U \cdot G_1$, where

$$U = \begin{bmatrix} 1+D_1D_2 & D_1 \\ D_2 & 1 \end{bmatrix}, \quad \text{and} \quad \det U = 1.$$

Definition 5 A convolutional generator matrix G of size $k \times n, k \leqslant n$ is called *basic* if it has a polynomial right inverse.

From the viewpoint of matrix primeness description, it is equivalent to saying that G is basic if G is zero left prime (ZLP), i.e., all $k \times k$ minors of G are devoid of any common zero. The theory of Gröbner bases for modules can be applied here to compute a right inverse of a polynomial generator matrix G. A more general algorithm and examples on the characterization of the whole class of all possible right inverses can be found in Ref. 17.

Algorithm 2 Computing the right inverse of a given generator matrix

Step 1: Let M be a module generated by all columns of the generator matrix G.

Step 2: Compute the Gröbner basis of the module M with respect to any ordering by using Buchberger's algorithm for modules (see p.149 in Ref. 13).

Step 3: Use the division algorithm (p.145 in Ref. 13) and the Gröbner basis of the module in Step 2 to determine if all column vectors of the identity matrix I_k are members of the module M. If so, the right inverse G^{-1} can be computed by retracing the steps of the division algorithm. Otherwise, the right inverse does not exist.

Example 2 Consider a binary encoder matrix,

$$G = \begin{bmatrix} 1 & 0 & D_1 \\ D_2 & D_1 & 1+D_1D_2 \end{bmatrix}.$$

By using the SINGULAR program [18] for computing the Gröbner basis, the syzygy module and implementing the long division algorithm [17] one can obtain the parameterized right inverse in the form:

$$Q = \begin{bmatrix} 1+D_1D_2 & D_1 \\ 0 & 0 \\ D_2 & 1 \end{bmatrix} + \begin{bmatrix} D_1^2 \\ 1 \\ D_1 \end{bmatrix} \begin{bmatrix} a & b \end{bmatrix}$$

where $a, b \in \{0, 1\}$. Since G has a polynomial right inverse, i.e., $GQ = I_2$, the encoder matrix G is basic.

Encoding procedure

In this section, a 2-D convolutional encoding process is illustrated by an example. Let us consider a 2-D convolutional code, of rate $\frac{2}{3}$ whose generator matrix is given by

$$G = \begin{bmatrix} 1 & D_1 & D_1D_2 \\ 0 & D_2 & D_1+1 \end{bmatrix}$$

and let I be the 2-D binary input taken from a 3×3 pixel, 4-colour image

$$I = \begin{bmatrix} 01 & 10 & 10 \\ 11 & 00 & 00 \\ 01 & 10 & 01 \end{bmatrix}$$

whose algebraic representation is given as

$$u = \begin{bmatrix} D_1+D_2+D_1^2+D_1D_2^2 & 1+D_2+D_2^2+D_1^2D_2^2 \end{bmatrix}.$$

The output v of an encoder G can be obtained by vector matrix multiplication:

$$v = u \cdot G = \begin{bmatrix} D_1+D_2+D_1^2+D_1D_2^2 \\ \begin{pmatrix} D_2+D_1^2+D_1D_2+D_2^2 \\ +D_1^3+D_2^3+D_1^2D_2^2+D_1^2D_2^3 \end{pmatrix} \\ \begin{pmatrix} 1+D_1+D_2+D_1D_2+D_2^2+D_1^2D_2 \\ +D_1^2D_2^2+D_1^3D_2+D_1^3D_2^2+D_1^2D_2^3 \end{pmatrix} \end{bmatrix}^{\mathrm{T}}.$$

The polynomial output vector y can be represented in the geometrical form as

$$I_{\mathrm{out}} = \begin{bmatrix} 001 & 101 & 110 & 010 \\ 111 & 011 & 001 & 001 \\ 011 & 100 & 011 & 001 \\ 010 & 000 & 011 & 000 \end{bmatrix}.$$

Observation 1: If k is the number of symbols (bits) of each input pixel, and the generator matrix is of size $k \times n$, the number of symbols (bits) of each encoded pixel is n and is independent of the number of input pixels. On the other hand, the size of the encoded image (2-D array) depends directly on the row-wise maximal total degree of the generator matrix.

Observation 2: In the case when the size of the input is large (e.g., 256×256 pixels), the total degree of the input vector would grow significantly. However, this is not a problem in the implementation stage since the realization of the encoder is done by using an array shift register, not polynomial multiplication.

Observation 3: One assumption used here is that the number of rows of the generator matrix G must be the same as the number of symbols (bits) representing each pixel. When this is not the case, one needs to regroup the m-D data symbols to suit the encoder matrix.

SYNDROME DECODER AND DUAL CODE

The decoding of m-D convolutional code can be performed in a straightforward manner by means of syndrome decoding. However, unlike the soft-decision scheme, e.g., Viterbi decoding in 1-D, this hard-decision decoding method may not be optimal in the maximum-likelihood sense nor the maximum a priori sense [5]. In this section, the basic definition and existence criteria of the syndrome decoder are given along with the algorithm for computing the syndrome decoder matrix.

Definition 6 Let $\mathcal{C}$ be an R-submodule of the free R-module, R^n. An *image representation* of $\mathcal{C}$ is a matrix $G \in R^{l \times n}$ such that $\mathcal{C} = \mathrm{rowspace}(G)$. A *kernel representation* of $\mathcal{C}$ is a matrix $H \in R^{n \times p}$, where p is some positive integer, such that $\mathcal{C} = \ker(H) = \{w \in R^n | w \cdot H = \mathbf{0}\}$.

If a sequence $w \in \mathcal{C}$, then $w = u \cdot G$ for some $u \in R^k$, and so $w \cdot H = u \cdot G \cdot H = \mathbf{0}, \ \forall\, u$. This implies $GH = \mathbf{0}$.

Given an m-D convolutional code $\mathcal{C}$ with the associated generator matrix G, the kernel representation (i.e., the matrix H), if it exists, can be constructed by computing the syzygies of the module generated by all columns of the generator matrix G (p.161, Ref. 13).

Example 3 Let G be the minor left prime matrix

$$G = \begin{bmatrix} 1 & D_1 & D_1D_2 \\ 0 & D_2 & D_1+1 \end{bmatrix}.$$

The kernel representation of G is computed by using the SINGULAR program [18] command lines

```
>ring r=2,(x,y),(c,dp);
>module M=[1,0],[x,y],[xy,x+1];
>module H=syz(M);
>H;
H[1]=[x2+xy2+x,x+1,y]
```

to obtain $H = \begin{bmatrix} D_1 + D_1^2 + D_1D_2^2 & D_1+1 & D_2 \end{bmatrix}^{\mathrm{T}}$.

By the properties of the matrix H and syzygy, all codewords in the set $\mathcal{C} = \mathrm{rowspace}(G)$ satisfy the condition

$$w \cdot H = 0, \quad \forall\, w \in \mathcal{C}.$$

This condition provides a *parity check*, which can be applied to a received sequence for testing whether it is a codeword. The matrix H is often called the *parity check matrix* or the *syndrome decoder* of $\mathcal{C}$ and the corresponding codes are called *finite convolutional code*. Note that not every convolutional code admits

a syndrome decoder (i.e., the parity check matrices of some m-D convolutional codes do not exist)[19,20]. The following proposition characterizes the existence of a syndrome decoder.

Proposition 2 [8,9] *A free modular code $\mathcal{C}$ admits a syndrome decoder if and only if $\mathcal{C}$ has a minor prime generator matrix G.*

(An m-variate polynomial matrix G is minor prime[21], if all maximum-size minors (major determinants) are devoid of common factor other than units.) For a Laurent polynomial ring, the existence condition[8] is relaxed to the condition that the generator matrix must be left factor prime.

Example 4 Consider a 3-D convolutional code $\mathcal{C}$ with a corresponding generator matrix:

$$G = \begin{bmatrix} D_1 & 0 & D_2 \\ 0 & D_1 & D_3 \end{bmatrix}.$$

The major determinants of G contain the common factor D_1. As a result, G is not minor prime. In p.16 of Ref. 9, it has been shown that G is in fact left factor prime. By using the SINGULAR program, $H = [\ D_2 \quad D_3 \quad D_1\ ]^{\mathrm{T}}$ is the syzygy of the module generated by all columns of G. However, H is not a syndrome decoder of $\mathcal{C}$ since $v = [\ D_3 \quad D_2 \quad 0\ ]$ satisfies $v \cdot H = 0$, but it does not belong to the set $\mathcal{C}$ (i.e., there does not exist a row vector $u \in R^2$ that produces $u \cdot G = [\ D_3 \quad D_2 \quad 0\ ]$).

Definition 7 The *orthogonal module* of M is denoted by $M^{\perp}$ and is given by

$$M^{\perp} \triangleq \left\{ w \in R^n | wv^t = 0,\ \forall v \in M \right\}.$$

For an m-D convolutional code $\mathcal{C}$, the *orthogonal code* of $\mathcal{C}$ is denoted by the module $\mathcal{C}^{\perp}$. In the case where the $\mathcal{C}$ admits a syndrome decoder (i.e., has a kernel representation), the orthogonal code $\mathcal{C}^{\perp}$ is called the *dual code* of $\mathcal{C}$.

REALIZATION OF CONVOLUTIONAL ENCODER

The implementation of multidimensional convolutional code requires the use of various dimensional delays (shift registers), $D_1, D_2, \ldots$ where an example of the canonical form realization is shown in Fig. 1. In this section, the focus is on the minimal realization of a 2-D convolutional code. The higher dimension cases can be tackled in a similar fashion.

To realize the encoding process efficiently in the form of hardware, it is necessary to minimize the number of delays needed. In the 1-D case, one objective is to single out the generator matrix, from all equivalent ones, that has the minimum overall constraint length. However, a meaningful definition of constraint length, in a higher $(m \geqslant 2)$ dimension case, has not been given anywhere. A good candidate for this parameter should directly indicate the total number of delays in the implementation of a desired encoder. Given a generator matrix $G \in R^{k \times n}$, we wish to identify among all equivalent generator matrices, the one whose realization uses the minimum number of delay elements. A practical procedure towards the solution of this problem is only possible by direct searches via a computer program.

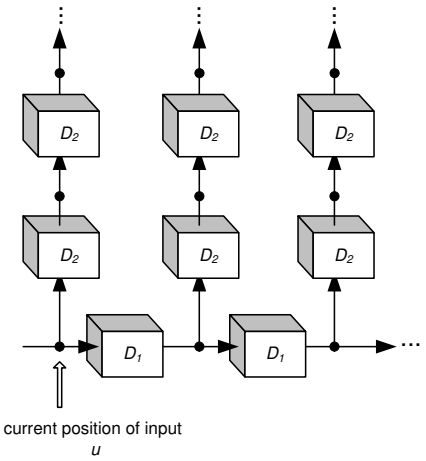

Fig. 1 Example of a 2-D shift register array where u_i is the input and D_1, D_2 are the horizontal and vertical delays, respectively.

One approach to solve the problem stated above is to perform a search over the space of all possible equivalent generator matrices. Since an equivalent generator matrix can be generated by multiplying a unimodular matrix by the given generator matrix, there are an infinite number of elements to search from. As a result, a finite scope for searching must be set in such a way that the minimal basic encoder should be found most of the time. Here we limit the searching space by limiting the total degree of each element in the unimodular matrix to be less than a certain integer chosen according to the available search time.

Number of delay elements

Since there is only one kind of delay in one-dimensional convolutional encoder realization, it is straightforward to count the total delays[22]. However,

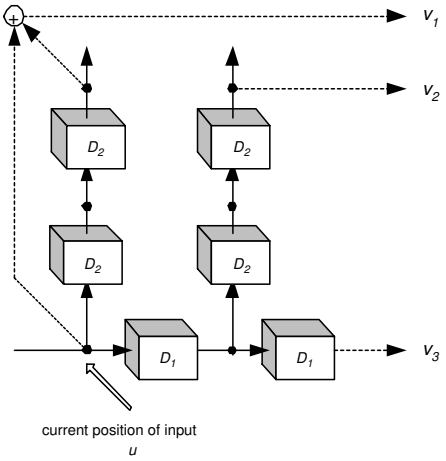

Fig. 2 Direct realization of the encoder matrix G_1.

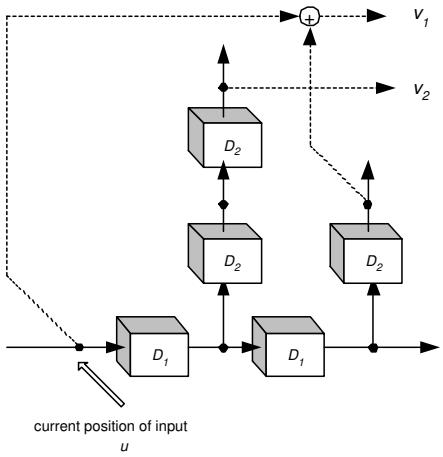

Fig. 3 Direct realization of the encoder matrix G_3.

in 2-D, the counting procedure is nontrivial as illustrated by the following example.

Example 5 Let $G_1 = [1 + D_2^2 \;\; D_1D_2^2 \;\; D_1^2]$. It is not so obvious that two D_1's and four D_2's may be used. The realization diagram in Fig. 2 clearly indicates how many delays are needed.

Given a realization diagram, the total number of the delays required can be easily counted. However, one of the goals here is to find an algorithm for quickly counting the number of delays needed without drawing the diagram.

For the monomial case, the total number of delays is counted from its total degree.

Example 6 Given a generating matrix $G_2 = [D_1^2D_2^2]$, to realize G_2, two D_1's and two D_2's are required. In other words, the total number of delays required are the same as the total degree of the monomial.

Counting delays for a polynomial row vector G

For illustrative purposes, let G_3 be a (1×2) generator matrix $G_3 = [D_1^2D_2 + 1 \;\; D_1D_2^2]$. The associated realization diagram is given in Fig. 3. From Fig. 3, it is clear that to realize G_3, two D_1 and three D_2 are required.

The following algorithm can be used to find the total number of delays for a polynomial row vector G without drawing a realization diagram.

Algorithm 3 Counting delays for a polynomial row vector G

Step 1: Let $G(1, j)$ be the polynomial in the jth column. To find out the number of D_1, compare all polynomials in G and find D_1 with the highest degree (treating D_2 as constant). The number of D_1 required is equal to the maximum degree of all polynomials: $\mathrm{del}_{D_1}(G) = \max_j\{\deg_{D_1} G(1, j)\}$.

Step 2: The number of delays, D_2, is the sum over r_1 of all the maximum degrees r_2 of monomials in the form $D_1^{r_1}D_2^{r_2}$ of all $G(1, j), r_1 = 0, 1, 2, \ldots$.

Step 3: The total number of delays needed can be found by simply summing all the delays: $\mathrm{del}(G) = \mathrm{del}_{D_1}(G) + \mathrm{del}_{D_2}(G)$.

Example 7 Let

$$G = [D_2 \;\; D_1D_2 \;\; 1 + D_1^2D_2]. \tag{3}$$

Step 1: Clearly, $\deg_{D_1} G(1, 1) = 0, \deg_{D_1} G(1, 2) = 1$, and $\deg_{D_1} G(1, 3) = 2$. Therefore, $\mathrm{del}_{D_1}(G) = \max_j\{\deg_{D_1} G(1, j)\} = 2$.

Step 2: For $r_1 = 0, G(1, 1) = D_1^0D_2^1$, so one D_2 is needed. For $r_1 = 1, G(2, 1) = D_1^1D_2^1$, so another D_2 is needed. Finally, $r_1 = 2, G(1, 2) = 1 + D_1^2D_2$, so another D_2 is needed. Therefore, $\mathrm{del}_{D_2}(G) = 1 + 1 + 1 = 3$ and $\mathrm{del}(G) = 5$.

Note that the total number of delays could be different if the roles of D_1 and D_2 were interchanged in the algorithm. For example, in this example only three delays (one D_2 and two D_1's) are required. Also note that the presence of '1' in the elements of G is completely irrelevant to the total number of delays.

Counting delays for a polynomial matrix

To facilitate the understanding, the algorithm will be presented in parallel with an example.

Example 8 Let

$$G = \begin{bmatrix} 1+D_1 & D_1D_2 & D_1^2 \\ D_2^2 & 1 & 1+D_2 \end{bmatrix}. \qquad (4)$$

Algorithm 4 Counting the total number of delays for a given generator matrix G

Step 1: Count one row at a time using Step 1 of Algorithm 3. The total number of delays D_1 needed is

$$\deg_{D_1} G = \sum_i \max_j \{\deg_{D_1} G(i,j)\}.$$

In this example, $G(1,3) = D_1^2$ and there is no D_1 in the second row. Hence only two D_1's are needed.

Step 2: Count the number of D_2's row-wise by using Step 2 in Algorithm 3.
In the example, in the first row the number of D_2 delays required is only one, due to the term D_1D_2. In the second row, the number of D_2 delays required is two, due to the monomials D_2^2. Hence three delays D_2 are required for implementing G.

Step 3: Compute the total number of delays required by adding results from the two previous steps.
For the generator matrix G in (4), five (two D_1's and three D_2's) delays are needed.

Alternatively, the above algorithm may be reconsidered by using the notation of support which can be used directly in a computer program.

Support of a generator matrix

A support S of a bivariate polynomial row-vector $G = [g_1(D_1, D_2)\ \ g_2(D_1, D_2)\ \ \cdots\ \ g_k(D_1, D_2)]$ is defined as a $p \times q$ binary matrix S whose element $S(r,t)$ is 1 if there exists $D_1^{r-1}D_2^{t-1}$ as a monomial in at least one of the polynomials g_j, $j = 1, 2, \ldots, k$ and 0 otherwise. For example, (3) defines a map onto a support matrix in Fig. 4.

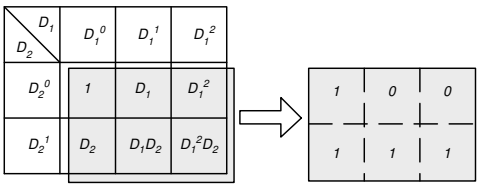

Fig. 4 Mapping of a generator encoding matrix onto a support matrix using (3).

For the purpose of mathematical representation and computation, the support will be represented in a matrix form. Each row of G can be represented by one support matrix. The ith row and jth column of support matrix corresponds to the presence of the monomial $D_1^iD_2^j$, $i, j = 0, 1, 2, \ldots$ in the row of generator matrix G. To avoid confusion, elements of G are generally polynomials, each of which consists of the sum of one or more monomials.

The rest of the matrix will be filled with 0's. For example, the support of (3) is

$$S = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}.$$

The next algorithm summarizes the method to generate the support of a generator matrix.

Algorithm 5 Construction of a support

Again, the generator matrix in (3) will be used to illustrate the method.

Step 1: For each monomial in the ith row G, put a '1' in the support matrix, S_i, in the associated position. For example, $G(1,3) = 1 + D_1^2D_2$. Therefore, $S(0,0) = 1$ because of the monomial $1 = D_1^0D_2^0$ in $G(1,3)$ and $S(1,0) = 1$ because of the D_2 in $G(1,1)$.

Step 2: If the same monomial is encountered more than once, then the later found monomials are ignored. The presence of the second monomial does *not* cancel any entries in the support.

Step 3: Finally, the size of the matrix can be checked for verification. The number of columns is equal to the $\max_{i,j}\{\deg_{D_1} G(i,j)\} + 1$ and the number of rows is $\max_{i,j}\{\deg_{D_2} G(i,j)\} + 1$.

Note that for a given generator matrix G, there are as many support matrices S as the number of rows in G.

Example 9 Let

$$G = \begin{bmatrix} 1 + D_1D_2 + D_1^3 & D_1 + D_2^2 \\ D_1^3 + D_2 + D_1D_2^3 & 1 + D_1^2 + D_1^2D_2^2 \end{bmatrix}.$$

The supports of G are

$$S_1 = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}, \quad S_2 = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}.$$

Algorithm 6 Counting delays for a generator matrix using support

Step 1: Represent each row g_i of the $n \times k$ generator matrix G by the corresponding support matrix S_i for $i = 1$ to n.

Step 2: If the size of the support matrix S_i is $p_i + 1 \times q_i + 1$, then the number of D_1 delays for realizing g_i is computed by $\mathrm{del}_{D_1}(g_i) = q_i$.

Step 3: Remove the first row of S_i, thereby making it a $(p-1)\times q$ matrix.

Step 4: This is the important step. Consider one column of S_i at a time and identify the row number (the top row is counted as row number 1) of the last row (of that column) that contains '1'. Add these numbers up for all columns. The result is the number $\mathrm{del}_{D_2}(g_i)$ of D_2 delays required.

Step 5: Compute the total number N of delays for G by adding up the number of D_1 delays and D_2 delays required for each row of G. That is,

$$N=\sum_{i=1}^{n}\mathrm{del}_{D_1}(g_i)+\mathrm{del}_{D_2}(g_i).$$

Minimal Encoder

In this subsection, let first define the term minimal and propose two approaches to produce a minimal encoder. It is well-known that a given m-D convolutional code can be generated by many equivalent encoders, each of which is characterized by a generator matrix. The class of equivalent generator matrices can be parameterized by unimodular multiplication:

$$\mathcal{S}(G_0)=\left\{G\in R^{k\times n}\middle|\ G=UG_0\right\}$$

where U is a unimodular matrix and $G_0\in R^{k\times n}$. Here, a unimodular matrix is defined as a square multivariate polynomial matrix whose determinant is a unit element in the coefficient field $\mathbb{F}$, i.e., one.

$$U \text{ is unimodular} \longleftrightarrow |U|=\det(U)=1. \quad (5)$$

Examples of unimodular matrices are:

$$U_1=\begin{bmatrix} D_1D_2+1 & D_1 \\ D_2 & 1 \end{bmatrix},$$

$$U_2=\begin{bmatrix} D_1 & 1 \\ 1+D_1^2+D_1D_2 & D_1+D_2 \end{bmatrix}.$$

Given a generator matrix, to find all possible equivalent generator matrices, as many unimodular matrices as possible must first be found. That is, given $G_0\in R^{k\times n}$, to find a G which is equivalent to G_0, one needs to construct $U\in R^{k\times k}$. One way to find as many U as possible for a particular $R^{k\times k}$ is to consider a *sequential* search through as many polynomial matrices as possible. Unimodular matrices may be generated by systematically arranging possible combinations of polynomial entries of the matrix.

First, the given generator matrix is entered as an input. Then its number of required delays is counted and stored. Next, multiply the given generator matrix by a stored unimodular matrix to produce a different equivalent generator matrix G. Then the number of delays for this new equivalent matrix is computed and compared with the original number. The G that produces a smaller number of delays will be stored. The process is repeated with new equivalent matrices until all unimodular matrices in storage are used. Finally, the stored generator matrix G is the one that required the smallest number of delays found. Clearly, the search result depends on the searching space. Since theoretically the searching space is infinitely countable, the minimal encoder is not guaranteed to be found. However, from the practical point of view, one can start the search by generating a large number of unimodular matrices. From our experimental testing, we found that this approach does provide a quick and suboptimal result most of the time. Next, the detailed implementation is explained.

Consider the two-dimensional case. Let

$$U=\begin{bmatrix} P_1 & P_2 \\ P_3 & P_4 \end{bmatrix} \quad (6)$$

where $P_i, i=1,2,3,4$ is a 2-D polynomial. Now, the goal is to find a set of $\{P_i\}_{i=1}^4$ such that U becomes a unimodular matrix. For instance, P_1 may be in the form

$$P_1=a_1+a_2D_1+a_3D_1^2+a_4D_2+a_5D_1D_2 + a_6D_1^2D_2+a_7D_2^2+a_8D_1D_2^2+a_9D_1^2D_2^2$$

where $a_i\in\{0,1\}$. The concept of support is particularly helpful to visualize this polynomial P_1. P_1 can be written in the form of a support as

$$S_{P_1}=\begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix}. \quad (7)$$

A similar treatment can be applied for other P_k, $k=2,3,4$ with a_i replaced by b_i, c_i, and d_i, respectively.

An example is given to illustrate how to generate a minimal delay encoding matrix. The example was worked out with the help of computer software to generate all possible 2×2 unimodular matrices with maximum degree of each element less than or equal to $(2,2)$.

Example 10 Suppose

$$G=\begin{bmatrix} D_1D_2 & D_2^2 & 1+D_1^2 \\ D_1(1+D_2^2) & D_2+D_2^3 & 1+D_1+D_1^2D_2 \end{bmatrix}. \quad (8)$$

This requires four D_1's and nine D_2's. Let the input of the function be the support of each row of (8). After

the search, the minimal delay generator matrix of (8) will be

$$G_{\min} = \begin{bmatrix} D_1 & D_2 & 1+D_1 \\ D_1D_2 & D_2^2 & D_1^2 \end{bmatrix} \qquad (9)$$

which requires three D_1's and only four D_2's, assuming that the standard realization structure set up earlier is used, and the unimodular matrix $U_{\min}$ used to produce $G_{\min}$ is

$$U_{\min} = \begin{bmatrix} D_2 & 1 \\ 1 & 0 \end{bmatrix}.$$

In this sequential search approach, the performance of the search is not optimal and largely depends on the size of the searching space (i.e., the number of unimodular matrices used). Furthermore, for a large generator matrix, the sequential approach will suffer from the higher complexity. The general problem to find the optimal solution is still open. The proposed standard realization structure is a fixed structure and the minimum number of delays is not ensured (as detailed in next section).

Finding minimal delay encoder by algebraic approach

In this section, a more general approach to identify the minimal encoder is proposed. Given a generator matrix G, one can observe that the number of delays is directly related to the maximum total degree of each multivariate polynomial row vector in G. This fact is especially true when a strictly rectangular array of delays D_1, D_2 is used. That is, the number of delays can be identified by the sum of the row-maximum degree product.

In the 1-D case, there exists a well-known algebraic algorithm for computing the minimal convolutional encoder[3,22]. The 1-D algorithm is based on the Smith-form description. The direct generalization of this algebraic approach is not possible as there is no generalization of the Smith-form description in m-D. Furthermore, the number of delay elements in the realization of m-D convolutional encoder cannot be computed by adding the row-wise constraint length (constraint length is undefined in m-D convolutional code). The attempt here is to determine an alternative m-D approach similar to the algorithm given in[3,5,6,22,23].

By inspection, it can be inferred that the total number of delay elements required for realizing a given convolutional code depends greatly on the maximum total degree of each row vector of its generator matrix G. Using this fact, one can remove the unnecessary delays during the implementation by first finding a lower row-wise maximum total degree equivalent generator matrix $G' = UG$, where U is the unimodular matrix, and use it for the realization instead.

There are two approaches proposed for finding the minimal delay encoder. The first is based on the sequential search over the finitely many set of sample space, whose members are the equivalent generator matrices obtained by multiplying the given generator matrix by unimodular matrices. The result of this approach directly depends on the number of unimodular matrices used for forming the searching space. The second approach here is based on the usage of Gröbner basis to perform an algebraic row-operation of the generator matrix to reduce row-wise the maximum total degree. This approach has a significant advantage over the previous method, in that it provides a fast and systematic way to the suboptimal solution. The term *suboptimal* is used here due to the fact that there may exist a generator matrix with larger total degree but requiring a smaller number of delay elements than the one with minimum total degree. When the encoder is realized in the direct form[15], the total degree is directly (but not proportionally) related to the number of delays. The algebraic approach to find the minimal realization where the optimal solution is guaranteed may not exist due to the nonlinear relationship between the well-ordering degree and the total number of delays.

Consider a generator matrix G, whose rows are $g_1, g_2, \ldots, g_k \in R^n$. Multiplying G by a unimodular matrix U_1 to get another equivalent generator matrix $G_1 = U_1G$ is essentially performing a row operation on G. The multiplying factor U_1 is constructed by using a Gröbner basis with respect to a proper choice of degree ordering such that the maximum total degree of a selected row in G' is less than that of G.

Algorithm 7 Minimization of the row-wise maximum degree (w.r.t. degree reverse lexicographical ordering)

Given an $k \times n$ generator matrix G whose rows are denoted by $g_1, g_2, \ldots, g_k \in R^n$,

Step 0: Initialize $i = 1$, where i is the row number to be reduced.

Step 1: Define a module M generated by all row vectors except the ith row.

Step 2: Compute the Gröbner basis S of the module M w.r.t. degree lexicographical ordering and *term over position* (TOP) ordering (see p.142 of Ref. 13).

Step 3: Reduce the ith row w.r.t. the Gröbner basis S. The leading monomial of the reduced ith row will

have a smaller or equal degree.

Step 4: Form the updated generator matrix G' by replacing the ith row of G by the reduced ith row obtained in Step 3. This generator matrix is equivalent to the given generator matrix.

Step 5: If $i < k$, let $i = i + 1$ and go to Step 1. Otherwise, the algorithm is completed.

The following points should be noted. First, in Step 2 of Algorithm 7, the degree lexicographical ordering is used because the aim here is to reduce the maximum total degree. In the degree lexicographical ordering, the monomial with the maximum total degree is the leading monomial and will be reduced first in the reduction process. Second, the algorithm can be reapplied to the resulting generator matrix to further reduce the maximum degree (w.r.t. degree lexicographical and TOP orderings). And finally, each row of the updated generator matrix is reduced with respect to the module generated by the remaining rows. As a result, the final generator matrix after repeatedly applying Algorithm 7 will converge to the minimum total degree among all possible equivalent generator matrices.

Example 11 Consider a binary 2-D 3×4 generator matrix $G \in \mathbb{F}_2[D_1, D_2]$,

$$G = \begin{bmatrix} G_{11} & G_{12} & G_{13} & G_{14} \\ G_{21} & G_{22} & G_{23} & G_{24} \\ 1 & D_1 & D_2 + D_1D_2^3 & D_1 + D_2^2 \end{bmatrix}$$

where $G_{11} = 1+D_1^2+D_1^2D_2$, $G_{12} = 1+D_1+D_1^3D_2$, $G_{13} = D_2 + D_1^2D_2^2 + D_1^3D_2^4$, $G_{14} = D_1 + D_2^2 + D_1^3D_2 + D_1^2D_2^3$, $G_{21} = 1 + D_1 + D_2 + D_1D_2 + D_1^2 + D_1^3$, $G_{22} = 1 + D_1^2 + D_1D_2$, $G_{23} = 1 + D_2 + D_1D_2 + D_2^2 + D_1D_2^4$, and $G_{24} = D_2 + D_1D_2 + D_1^2 + D_2^2 + D_1D_2^2 + D_2^3$. This generator matrix G requires 28 delay elements, detailed in Table 1.

After a passing through the first iteration of Algorithm 7, by using SINGULAR, one obtains an equivalent generator matrix G', shown in (10) which requires only 13 delay elements for realization using the canonical form. The subsequent iteration results in the same generator matrix G' and so the algorithm converges to G'.

Table 1 Numbers of delays required for implementation of G and G'.

Row No.	G			G'		
	D_1's	D_2's	Sum	D_1's	D_2's	Sum
1	3	9	12	2	2	4
2	3	7	10	1	2	3
3	1	5	6	1	5	6
Total	7	21	**28**	4	9	**13**

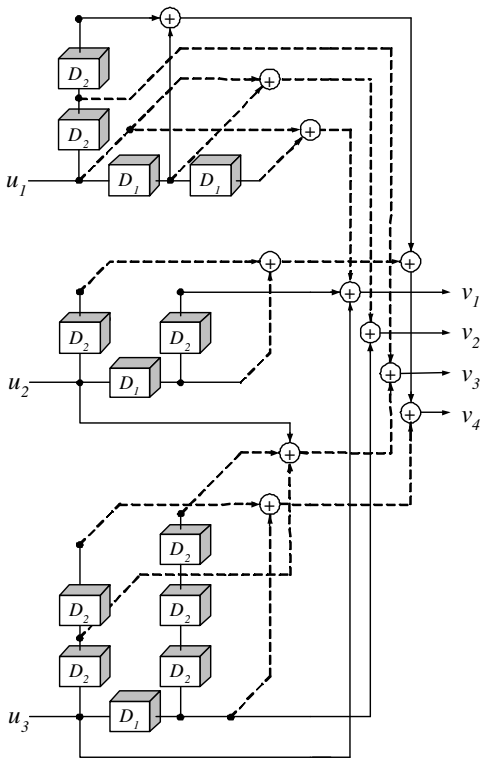

Fig. 5 Realization of the convolutional code with the generator matrix G'.

$$G' = UG = \begin{bmatrix} 1+D_1^2 & 1+D_1 & D_2 & D_1 + D_2^2 \\ D_1D_2 & 0 & 1 & D_1+D_2 \\ 1 & D_1 & D_2+D_1D_2^3 & D_1 + D_2^2 \end{bmatrix}$$

where

$$U = \begin{bmatrix} 1 & 0 & D_1^2D_2 \\ 1+D_1 & 1 & D_2 \\ 0 & 0 & 1 \end{bmatrix}$$

is a unimodular matrix. The realization of the 2-D convolutional encoder with the generator matrix G' is shown in Fig. 5.

CONCLUSIONS

It has been shown that there exists a strong relationship between the theories of multidimensional systems and those of m-D convolutional code. In the system point of view, the focus is on the input and output relationship and how the system responds to a different input. On the other hand, the coding side only concentrates on the set of output (codewords)

and its properties. The set of all m-D convolutional codewords generated by a generator matrix G can be modelled as an R-submodule, where many mathematical tools such as Gröbner bases, module theory and algebraic geometry can be applied.

The minimal encoder realization is explored and the algorithm for finding a suboptimal solution is given. Although the solution obtained from the sequential search may not be minimal in some cases, with the current computing technology, it is possible to extend the searching space so that the optimal solution is found every time. Another algorithm for computing the minimal encoder based on some algebraic theories is in the development process and will be reported in the near future.

There are still many areas of m-D convolutional codes which are rather unexplored, e.g., the design of m-D convolutional code, performance measurement, and soft decoding. There has been a study[9] on the weight and distance bound of m-D convolutional code which can be further investigated and may lead to the completion of optimal m-D convolutional code design procedure. Some progress has been made for the special class of m-D convolutional code design, namely the unit memory codes (the ones whose generator matrix contains only first or zero degree polynomials).

Acknowledgements: This work was supported by the Thailand Research Fund (TRG 4580063). The author would like to thank Professors N. K. Bose, Zhiping Lin, and H. A. Park for their valuable comments and discussions.

REFERENCES

1. Forney G Jr (1970) Convolutional codes I: Algebraic structure. *IEEE Trans Inform Theor* **IT-16**, 720–38.
2. Forney G Jr (1973) The Viterbi algorithm. *Proc IEEE* **61**, 268–78.
3. Johannesson R, Wan Z (1993) A linear algebra approach to minimal convolutional encoders. *IEEE Trans Inform Theor* **39**, 1219–33.
4. Johannesson R, Wan Z (1998) Some structural properties of convolutional codes over rings. *IEEE Trans Inform Theor* **44**, 839–45.
5. Mahapakulchai S (2002) MAP source-controlled channel decoding for image transmission using CPFSK and ring convolutional codes. PhD thesis, Pennsylvania State Univ.
6. Mahapakulchai S, Van Dyck RE (2004) Design of ring convolutional trellis codes for MAP decoding of MPEG-4 imagery. *IEEE Trans Comm* **52**, 1033–7.
7. Fornasini E, Valcher M (1994) Algebraic aspects of 2D convolutional codes. *IEEE Trans Inform Theor* **IT-40**, 1068–82.
8. Valcher M, Fornasini E (1994) On 2-D finite support convolutional codes: An algebraic approach. *Multidim Syst Signal Process* **5**, 231–43.
9. Wiener P (1998) Multidimensional convolutional codes. PhD thesis, Univ of Notre Dame.
10. Buchberger B (1985) Gröbner bases: An algorithmic method in polynomial ideal theory. In: Bose N (ed) *Multidimensional Systems Theory: Progress, Directions, and Open Problems*, Reidel, Dordrecht, pp 184–232.
11. Fitzpatrick P (1995) On the key equation. *IEEE Trans Inform Theor* **41**, 1290–302.
12. Orsini E, Sala M (2005) Correcting errors and erasures via the syndrome variety. *J Pure Appl Algebra* **200**, 191–226.
13. Adams W, Loustaunau P (1994) *An Introduction to Gröbner Bases*, American Mathematical Society.
14. Charoenlarpnopparut C, Tantaratana S (2003) Multidimensional convolutional code: Progresses and bottlenecks. In: Proceedings of the 2003 IEEE International Symposium on Circuits and Systems, Bangkok, Thailand, vol 3, pp 686–9.
15. Chaoenlarpnopparut C, Wongsura S, Davies A (2003) Direct realization of a 2-d convolutional encoder with lesser number of delay elements. In: Proceedings of the 3rd International Symposium on Communications and Information Technologies, Thailand, vol 1, pp 258–62.
16. Charoenlarpnopparut C, Tantaratana S (2004) Algebraic approach to reduce the number of delay elements in the realization of multidimensional convolutional code. In: Proceedings of the IEEE International Midwest Symposium on Circuits and Systems, Hiroshima, Japan, vol 2, pp 529–32.
17. Charoenlarpnopparut C (2000) Gröbner bases in multidimensional systems and signal processing. PhD thesis, Pennsylvania State Univ.
18. Greuel GM, Pfister G, Schoenemann H (2001) *SINGULAR 2.0. A Computer Algebra System for Polynomial Computations*, Centre for Computer Algebra, Univ of Kaiserslautern. [www.singular.uni-kl.de].
19. Charoenlarpnopparut C, Jangisarakul P (2008) Constructive algorithms for determining inverses and syndrome matrices of multidimensional convolutional encoders using the Gröbner basis approach. In: Proceedings of the 14th Asia-Pacific Conference on Communications, Tokyo, Japan.
20. Charoenlarpnopparut C, Bose N (2001) Gröbner bases for problem solving in multidimensional systems. *Multidim Syst Signal Process* **12**, 365–76.
21. Charoenlarpnopparut C, Bose N (1999) Multidimensional FIR filter bank design using Gröbner bases. *IEEE Trans Circ Syst II* **46**, 1475–86.
22. Johannesson R, Zigangirov KS (1999) *Fundamentals of Convolutional Coding*, IEEE Press, New York, USA.
23. Massey J, Mittelholzer T (1989) Convolutional codes over rings. In: Proceedings of the 4th Joint Swedish-Soviet International Workshop on Information Theory, Gotland, Sweden, pp 14–8.