On Minimal Trellises of Linear Binary Block Codes

HOC Roman *, FARKAS Peter**, Dr. Sergio Herrera-Garcia ***
* SWH s.r.o.,Stromova 9, 83007 Bratislava, SLOVAKIA, ** Slovak University of Technology, Ilkovicova 3, 812 19 Bratislava, SLOVAKIA, *** CITEDI-IPN Research Center, 2498 Roll Dr. # 757 Otay Mesa, San Diego, CA 92154

Abstract: - This paper deals with the problem of searching and designing of minimal trellises for linear binary block codes (LBBC), from the generator matrices (GM). Minimal trellises could be used for soft decoding, especially in decoders which utilize the so-called Maximum Likelihood criterion, e.g. in decoders based on Viterbi algorithm.

*Key Word*s: - linear block codes, minimal trellis, span, Viterbi algorithm, ML decoding, soft decoding.

1 Introduction

The problem of ML decoding with the help of the trellis was first time tackled in the year 1974 by Bahl et al. in [1]. There was described a BCJR algorithm (acronym from names of authors) which minimizes symbol error probability. The problem of trellis construction arose together with the perspective to use optimal decoding algorithms such as Viterbi algorithm, which minimizes sequence error probability. Many of theories of minimizing trellises have been found for simplifying the decoding complexity. Kschischang and Sorokine in [2] described algorithm for the trellis construction from generator matrix **G** = [*k.n*]. This matrix must be transformed to „trellis-oriented form“ with help of row operations over GF(2). Row vectors must be linearly independent vectors with the minimal possible span and each of them with minimal Hamming distance (*HD* = *d*) from another row vectors. In [2] these row vectors are called „atomic generators“. But authors don’t present method how to find *k* vectors with a really lowest span.

In [3] a method denoted as Greedy Algorithm I, which always generates minimal span vectors (definitions - see II.) is described. However it is based on an exhaustive search through all possible *n*-dimensional vectors and selection of *k* independent vectors with *HD* = *d*, which can generate the LBBC, and with the span as small as possible. This algorithm is unusable in real time for large values of *n*. Modified Greedy Algorithm II is also based on row operations in **G** over GF(2). In [3] column permutations in **G** are mentioned, but author hasn’t used them because of nonexistence of an optimal algorithm. In [4] complexity measures for minimal trellises are described - total number of vertices, edges, mergers and total number of additions and reductions that Viterbi algorithm must make.

We want to extend the set of these algorithms and also describe methods minimizing the span of GMs based not only on row operations, but also on column operations with column vectors of **G**. In section 2 we will describe the LBBC and its **G** matrix, span, span length, trellis and trellis complexity measures. In section 3 we will present the methods using column operations, which leads to trellis simplification, and section 4 contains results for various LBBCs.

2 Linear Block Code and its Trellis

Linear binary (*n*,*k*,*d*) block codes (LBBC) are transmission codes and belong to the group of error correcting codes. The symbols of LBBC are from the set {0,1}. LBBC encoder divides the input bit stream on *k*-tuples and encodes them into *n*-tuples, where *n*>*k*, i.e. a code brings into the bit stream redundant bits. A LBBC could be described using so-called generator matrix **G** with *k* rows and *n* columns. Hence **G** consists of *k* linearly independent vectors (rows) with the length *n*. Every codeword
c = (c_1, c_2, … c_n) could be obtained from a *k*-tuple
u = (u_1, u_2, … u_k)?using multiplication with **G** over GF(2): **c** =?**uG**.

A *trellis* is an oriented finite graph with vertices (nodes) and edges (branches) with labels. A set of vertices *V* is ordered in „depths“ which are represented through index *i*, *i* = 0, 1, …, *n*. There is only one vertex in depth 0 (start) and usually one in depth *n* (end). A set of edges *E* connects the vertices from depth *i*-1 to *i*, *i* = 1, 2, …, *n*. Labels on the edges correspond with bit values of the codewords in time *i*. The trellis must represent all codewords as the paths from start to end.

A *minimal trellis* has a minimal number of vertices | *V* | and a minimal number of edges | *E* | among all trellises that might represent a given LBBC. McEliece has presented in [3] so-called *BCJR trellis*, first time presented in [1]. A BCJR trellis is constructed from the control matrix **H** and is edge-minimal. In [3] it is shown that BCJR trellis can be constructed from (MSGM) GM **G** as well. Both principles of construction are very well described and in detail explained on examples in [3].

We have focused on the construction from **G** matrix. So-called *Minimal Span Generator Matrix* (MSGM matrix) [3] must have the following properties:

- The row vectors $\mathbf{g}_1$, $\mathbf{g}_2$, … $\mathbf{g}_k$ of **G** have to be linearly independent
- First nonzero symbol $g_{ij} = 1$ in each row vector $\mathbf{g}_i$, $i = 1, \dots k$ must be placed in different position (column) than first nonzero symbol in another rows $j = 1, 2, \dots n$ and these positions are stored in variables:
  $L_i(\mathbf{g}_i)$; $i = 1, 2, \dots k$ (1)
- Last nonzero symbol $g_{ij} = 1$ in each of row vector $\mathbf{g}_i$, $i = 1, \dots k$ must be placed in different position (column) than last nonzero symbol in another rows $j = 1, 2, \dots n$ and these positions are stored in variables:
  $R_i(\mathbf{g}_i)$; $i = 1, 2, \dots k$ (2)
- If the **G** matrix fulfils last two conditions, we say that it has a *LR property*.
- The *span* of row vectors is defined as:
  $Span(\mathbf{g}_i) = \{L_i(\mathbf{g}_i), L_i(\mathbf{g}_i)+1, \dots, R_i(\mathbf{g}_i)\}$;
  $i = 1, 2, \dots k$ (3)
- Number of row vector elements inside the span is called *spanlength*:
  $Spanlength(\mathbf{g}_i) = |\, Span(\mathbf{g}_i)\,|$ (4)

Span(**G**) is the set of row spans. *Spanlength*(**G**) is equal to the sum of the spanlengths of the row vectors. McEliece (Theorem 6.11) says that „a matrix is MSGM if and only if it has the LR property“. But he has actually considered row operations only (Gaussian elimination, GE) over GF(2) with rows of **G** (see Greedy Algorithm II in [3]). This method doesn't guarantee that Spanlength(G) will be really minimum spanlength. From this aspect is the expression „MSGM“ incorrect.

The trellis complexity could be evaluated using different methods. Some reasonable measures for Viterbi decoding complexity are mentioned in [4]. From the trellis we can obtain the total number of edges $|\,E\,|$ and total number of vertices $|\,V\,|$. We can obtain from the trellis also the total number of simplifications $|\,M\,|$, i.e. path reductions. In binary case a vertex has two incoming edges and one outgoing edge. In binary case:

$|\,M\,| = |\,E\,| - |\,V\,| + 1$ (5)

It means, that we can evaluate $|\,M\,|$ from the known values $|\,E\,|$ and $|\,V\,|$. (In non-binary case if the code is constructed over GF(q), the expression has a denominator equal to q-1). The value $|\,E\,|$ represents the total number of additions required to compute path metrics in the Viterbi algorithm and value $|\,M\,|$ is the number of binary comparisons required to obtain one survived path through in a trellis.

3 The Methods Based on Operations with Column Vectors

It's obvious, that the smaller is *Spanlength*(**G**), the less complex will be the trellis (i.e. the less will be the values of $|\,V\,|$ and $|\,E\,|$). To reduce the value of *Spanlength*(**G**), we considered not only row operations, but also column operations in **G**. The main reason why is because the row vectors usually contain in *Spans* some 0's which can be displaced out of the *Spans* and so reduce *Spanlength*(**G**) via columns permutations. Column exchanges do not affect the LBBC weight spectra, they cause only the rotation of the vector space in which the code is defined. The resulting **G'** matrix describes an equivalent *(n,k,d)* LBBC.

3.1 Method A.

```
/* Span reduction with the help of column operations*/
for($x = 1$; $x = n!$; $x$++) {
Select $n$ column vectors in G and change the column order (make permutations)
Write column vectors to the matrix marked as $\mathbf{G'}_x$
Transform $\mathbf{G'}_x$ matrix to have LR property
// GE, row operations over GF(2)
Calculate Spanlength($\mathbf{G'}_x$)
Store $\mathbf{G'}_x$ and Spanlength($\mathbf{G'}_x$)
}
Find matrix G' with min{Spanlength($\mathbf{G'}_x$)},
$x = 1, 2, \dots n!$
```

Method A transforms the known **G** matrix to all possible $\mathbf{G'}_x$ matrices and selects from them the one: **G'** with LR property and with the minimal span.
But all possible column replacements must be checked out and alike the Greedy Algorithm I, it isn't a real time task for accessible computers.

3.2 Method B.

```
/* Span reduction with help of column operations */
Set processRunTime
Calculate Spanlength(G)
Do {
Select 1st column in G
// random value $i$ from interval 1, 2, … $n$
Select 2nd column in G
// random value $j$ from interval $j = 1, 2, \dots n, j$ ? ?
Change column vectors $i$ and $j$, mark new matrix as G'
Transform G' matrix to have LR property
// GE, row operations over GF(2)
Calculate Spanlength(G')
if(Spanlength(G') < Spanlength(G))
then save G' as new G
Decrement processRunTime
}
While processRunTime has not expired, continue again.
```

Method B needs not to check all possible column permutations. Indices of the column vectors are generated randomly, only two columns are exchanged in one loop and the search can be stopped for example after the time expiration. But we gain this advantage at the cost that the solution is sub optimal. Random generation of column indices brings into the span minimizing process the element of uncertainty, i.e. we cannot ensure that the **G'** is the matrix with minimum span. The result matrix **G'** has LR property, so we can generate the trellis (for example with BCJR algorithm, well described in [3]).

3.3 Method C

Method C is the refinement of **Method B**.

```
In the loop:
Change column vectors i and j, mark new matrix as G'
Transform G' matrix to have LR property
// GE, row operations over GF(2)
Apply "logic" from the left
Apply "logic" from the right
Calculate Spanlength(G')
```

Method C: Principle of the "logic" applied from the left is presented in Fig.1. It is sometimes possible to find two appropriate column vectors in the LR matrix with the same vector activity. This *activity* means that column vector elements (bits) lie inside of the *Span*(**G**).

1st property: One bit of the 1st column vector starts the row span (bit value is 1), in fig.1 the 3rd bit in the 4th column vector starts the 3rd row *Span*($\mathbf{g}_3$).

2nd property: the vector element in another column, with the same index as the bit in 1st vector has value 0.

If we have found two vectors, which satisfy these conditions, after the column vector exchange the bit 0 will lie outside of the *Span*(**G**). Applying the "logic" from the left and from the right through all *2k* column vectors satisfying 1st vector property we can speed up the span reduction process.

.. .. ..

$$\begin{array}{ccccccc c}
 & & & ? & & ? & & \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots \\
0 & 0 & 0 & \mathbf{1} & 1 & \mathbf{0} & 0 & \dots \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & \dots \\
0 & 1 & 1 & 0 & 1 & 1 & 1 & \dots \\
0 & 0 & 1 & 1 & 0 & 1 & 1 & \dots \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & \dots
\end{array}
\sim
\begin{array}{ccccccc c}
 & & & ? & & ? & & \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots \\
0 & 0 & 0 & \mathbf{0} & 1 & \mathbf{1} & 0 & \dots \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & \dots \\
0 & 1 & 1 & 1 & 1 & 0 & 1 & \dots \\
0 & 0 & 1 & 1 & 0 & 1 & 1 & \dots \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & \dots
\end{array}$$

Fig.1: "Logic" applied from the left of the **G** matrix

LBBC [*n*,*k*,*d*]	[23,7,9]	[27,5,13]	[27,10,9](1st)	[36,15,10]	[13,5,5]	[27,10,9](2nd)
RO:*Spanlength*(**G**)	119	113	170	314	42	180
CE:*Spanlength*(**G**)	90	92	153	282	33	149
Span reduction	24,4 %	18,6 %	10,0 %	10,2 %	21,4 %	17,2 %
RO : | *V* |	1534	606	5630	185598	150	10238
CE : | *V* |	426	350	2990	46942	76	2414
reduction of | *V* |	72,2 %	42,2 %	46,9 %	74,7 %	49,3 %	76,4 %
RO : | *E* |	1660	636	6396	218364	180	11260
CE : | *E* |	520	380	3756	63324	98	3052
reduction of | *E* |	68,7 %	40,3 %	41,3 %	71,0 %	45,6 %	72,9 %
RO : | *M* |	127	31	767	32767	31	1023
CE : | *M* |	95	31	767	16383	23	639
reduction of | *M* |	25,2 %	0 %	0 %	50,0 %	25,8 %	37,5 %
RO : type	syst.	syst.	syst.	syst.	syst.	syst.
CE : type	non-syst.	non-syst.	non-syst.	non-syst.	non-syst.	non-syst.

Tab.1: Comparison of the trellis complexity reduction

Legend:	RO - Results obtained with McElliece's method (Row Operations only)
	CE - Results obtained using Column Exchanges too (Method C)
	type - if the trellis of **G'** matrix describes systematic LBBC too or non-systematic only

4 Results

We've chosen for implementation the C language and we used as the workstation standard PC with Pentium II 700 MHz processor. Some matrices are published in [7] and [8], some of them are published on the Internet.

In the table there are presented the results for miscellaneous LBBCs. We can see that relative small *Span* reduction leads to the eminently high total vertex and edge set reduction. If we reduce $|V|$ and $|E|$, we can reduce $|M|$ as well (see expression (5), but not in every case, see e.g. [27,5,13]).

G' matrices obtained using column operations too, describe non-systematic LBBCs. It is possible to find **G'** in systematic form, but the trellis reduction will not be so high.

For [27,10,9] we reduced total number of additions computed for every received codeword to 27,1 % from previous value (100 % - 72,9 %) and total number of comparisons to obtain one survived path through the trellis to 62,5 % from previous value (100 % - 37,5 %).

On the [13,5,5] example we will show the trellis complexity reduction graphically. It is obvious that the 2nd trellis in Fig. 2 is less complex as the trellis constructed from the **G'**, presented in Fig. 1, obtained with row operations.

The 1st trellis has complexity measures
$|V| = 150$, $|E| = 180$, $|M| = 31$.

The 2nd trellis has complexity measures
$|V| = 76$, $|E| = 98$, $|M| = 23$, i.e. $|V|$ is reduced to 50,7 %, $|E|$ to 54,4 % and $|M|$ to 74,2 %.

5 Conclusions

The main contribution of the new methods, which apply column permutations in **G** matrix of LBBC in order to reduce the overall *Spanlength*(**G**), is the trellis complexity reduction.

The less complex is the trellis of the LBBC, the simpler is the decoding process; i.e. the less operations (additions and comparisons) must be performed. We have not found exact mathematical expressions for minimizing bounds. We have used sub-optimal methods which has newer the less led to the massive trellis complexity reduction.

6 Acknowledgment

The authors want to express their thanks to SWH Siemens Company Slovakia, CITEDI-IPN Research Center, CONACYT Mexico as well as Scientific Grant Agency of Ministry of Education of Slovak Republic and Slovak Academy of Sciences (Grant VEGA 1/7615/20) for supporting this work.

7 References

[1] L.R. Bahl, J. Cocke, F. Jelinek, J. Raviv: „Optimal decoding of linear codes for minimizing symbol error rate.“ IEEE Trans. on Inf. Theory, vol.20, No.2, pp.284-287, March 1974.
[2] F.R. Kschischang, V. Sorokine: „On the trellis structure of block codes.“ IEEE Trans. on Inform. Theory, vol.41, No.6, pp.1924-1937, November 1995.
[3] R.J. McEliece: „On the BCJR trellis for linear block codes.“ IEEE Trans. on Inform. Theory, vol.42, No.4, pp.1072-1092, July 1996.
[4] A.B. Kiely, S.J. Dolinar, R.J. McEliece, L.L. Ekroot, W. Lin: „Trellis decoding complexity of linear block codes.“ IEEE Trans. on Inform. Theory, vol.42, No.6, pp.1687-1697, November 1996.
[5] G.B. Horn, F.R. Kschischang: „On the intractability of permuting a block code to minimize trellis complexity.“ IEEE Trans. on Inform. Theory, vol.42, No.6, pp.2042-2048, November 1996.
[6] R. Hoc, P. Farkas: „Hladanie minimalnej mriezky linearneho blokoveho kodu.“ Bratislava, SK: Telekomunikacie '97, 3rd international conference, June 1997.
[7] P. Farkas, W. Juling: „Five new best [27,10,9] codes.“ Hirzel-Verlag Stuttgart, AEÜ, vol.48, No.2, 1994.
[8] P. Farkas, A.S. Smirnov, J.V. Sotskov: „Linearne kody opravujuce mnohonasobne chyby.“ Informacne systemy, vol.14, No.5, pp. 540, 1986.

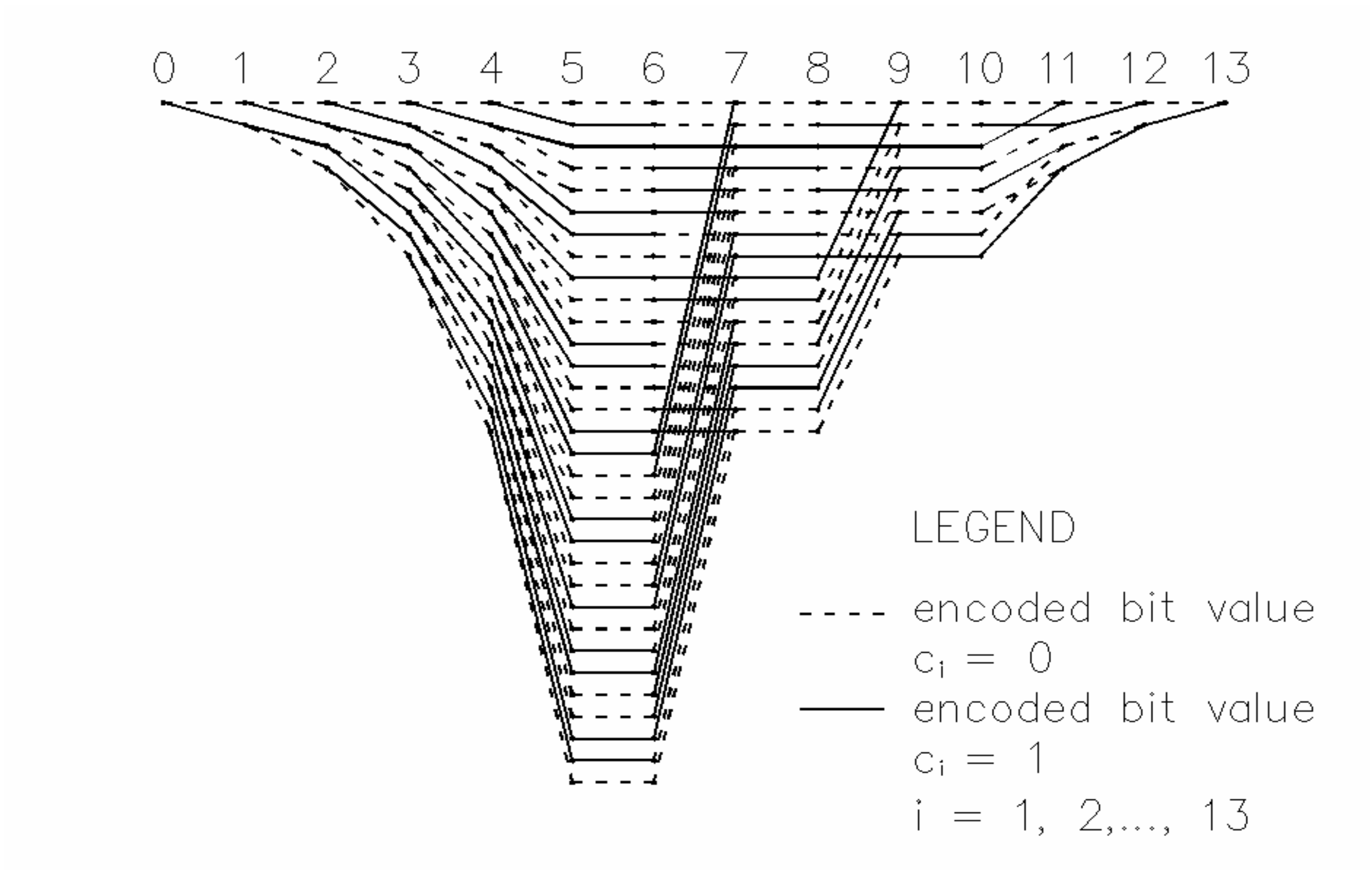
0
1
2
3
4
5
6
7
8
9
10
11
12
13
LEGEND
---- encoded bit value $c_i = 0$
—— encoded bit value $c_i = 1$
$i = 1, 2,\ldots, 13$

Fig. 1: trellis of [13,5,5] obtained with row operations

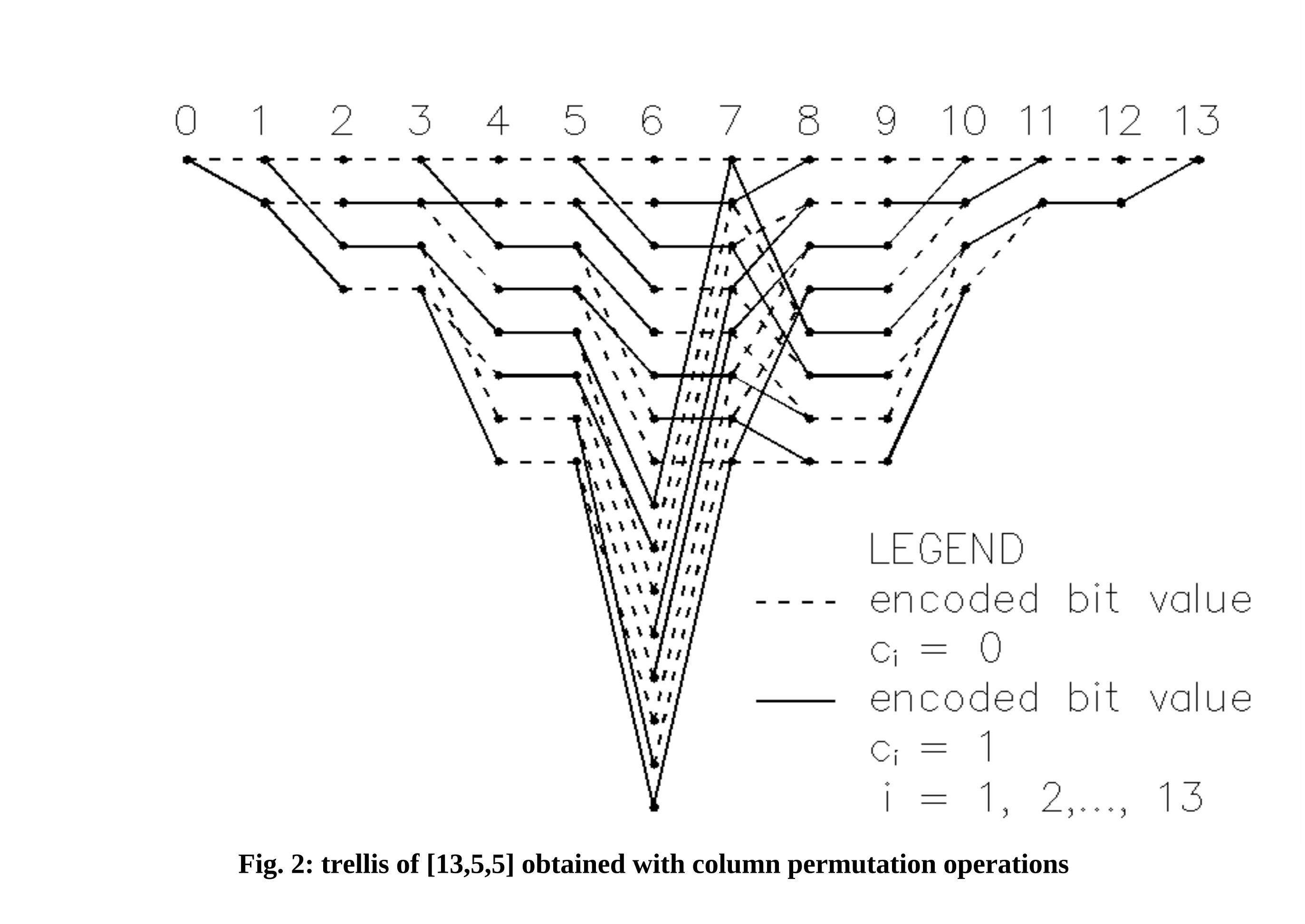
0
1
2
3
4
5
6
7
8
9
10
11
12
13
LEGEND
---- encoded bit value $c_i = 0$
—— encoded bit value $c_i = 1$
$i = 1, 2,\ldots, 13$

Fig. 2: trellis of [13,5,5] obtained with column permutation operations