Codes and decoding on graphs*

Priti Shankar

Department of Computer Science and Automation, Indian Institute of Science, Bangalore 560 012, India

We look at graphical descriptions of block codes known as trellises, which illustrate connections between algebra and graph theory, and can be used to develop powerful decoding algorithms. Trellises for linear block codes are known to grow exponentially with the code parameters and hence decoding on such objects is often infeasible. Of considerable interest to coding theorists therefore, are more compact descriptions called tail-biting trellises which in some cases can be much smaller than any ordinary trellis for the same code. We derive some interesting properties of tail-biting trellises and present an optimal maximum-likelihood decoding algorithm that performs rather well in terms of decoding complexity even at low signal-to-noise ratios.

OUR model of a communication system consists of a sender and a receiver communicating over an unreliable channel, as illustrated in Figure 1. The channel has an *input alphabet A* and an *output alphabet B*. Symbols from A are transmitted over the channel. In order to transmit information reliably, we transmit *blocks* of symbols rather than individual symbols. Before transmitting the blocks we *encode* them by adding redundant symbols in the hope that the redundancy will enable us to detect and correct errors. The encoding procedure produces a *codeword* from a *message* and it is this codeword that is transmitted. The set of all codewords is called a *code*. At the receiving end, what is obtained is a word r over the alphabet B. The object of the decoder is to obtain the codeword c' which was the most likely one to have been transmitted.

The theory of algebraic codes is founded on the structure and properties of finite fields. Indeed most algebraic decoding algorithms, for instance the Berlekamp–Massey algorithm[1,2], the Euclid[3] and the Guruswami–Sudan list-decoding algorithm[4] depend rather critically on these properties. However, such algorithms require channel outputs to be mapped into field elements, that is, they need *quantization* of the outputs before beginning the decoding process, resulting in loss of valuable information. Coding practioners have therefore tried to explore schemes where raw channel output symbols can be processed as such. One way of doing this is by using trellis representation of codes. Trellises are special kinds of graphs on which decoding algorithms that do not require quantization can be applied. The problem with such decoding schemes is that the algorithms are computationally demanding, in contrast with algebraic decoding whose complexity usually ranges from linear to cubic in the size of the input. Trellis-based algorithms are often probabilistic in nature. Until recently, the most widely used probabilistic decoding algorithm was the Viterbi decoding algorithm[5]. This is an optimal algorithm in the sense that it minimizes the probability of decoding error for a given code. However, the computational complexity of this algorithm is exponential in the code parameters. This is because the Viterbi algorithm operates on a trellis for the code, and the number of nodes (also referred to as the state space of the trellis) in such a graph is often very large, resulting in a high decoding complexity. A tail-biting trellis can be viewed as a superposition of subtrellises for subcodes that share nodes (also called states). The sharing of states gives rise to much smaller graphs and hence reduced decoding complexity. The reduction in the state space in going from an ordinary to a tail-biting trellis can be dramatic. For linear binary trellises, the number of states at any given time index is always a power of 2. So, for example, if the maximum state complexity of a linear trellis was 2^{16} which is approximately 65,000, it could reduce to 2^8, i.e. 256 for a tail-biting trellis for the same code. Thus, if we could obtain an algorithm to decode on the tail-biting trellis, it would be considerably less complex than the Viterbi algorithm on the conventional trellis.

The connections between algebraic and combinatorial representations of block codes are interesting, and we describe some of them here. We also show that one such connection leads to a decoding algorithm that appears quite promising from the point of view of computational complexity.

Background

We give a brief background on subclasses of block codes called linear codes. Readers are referred to the classic texts[6–8].

Let us fix the input alphabet A as the finite field $\mathbb{F}_q$ with q elements. It is customary to define linear codes algebraically as follows:

A linear block code $\mathcal{C}$ of message length k and block length n over a field $\mathbb{F}_q$ is a k-dimensional subspace of an n-dimensional vector space over the field $\mathbb{F}_q$ (such a code is called an (n, k) code).

*Dedicated to Prof. S. Ramaseshan on his 80th birthday.
e-mail: priti@csa.iisc.ernet.in

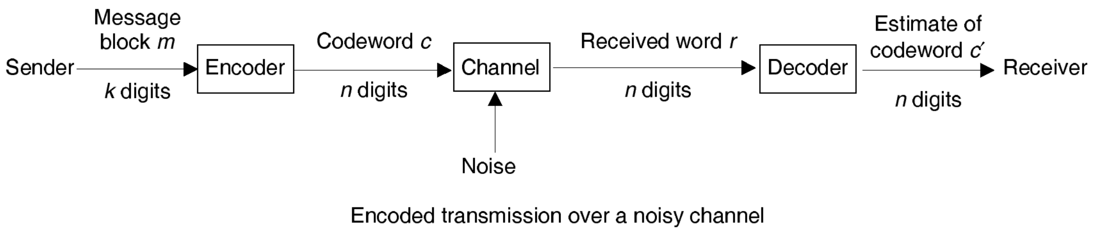

Figure 1. A typical error control scheme.

$$G=\begin{bmatrix}0 & 1 & 1 & 0\\ 1 & 0 & 0 & 1\end{bmatrix}$$

Figure 2. Generator matrix for a (4, 2) linear binary code.

The most common algebraic representation of a linear block code is the generator matrix G. A $k \times n$ matrix G where the rows of G are linearly independent and which generate the subspace corresponding to $\mathcal{C}$ is called a *generator matrix* for $\mathcal{C}$. Figure 2 shows a generator matrix for a (4, 2) linear code over $\mathbb{F}_2$.

An alternate way of specifying the code $\mathcal{C}$ is by the parity check matrix H of $\mathcal{C}$, which enjoys the following property:

$$H\mathbf{c}^T = 0, \forall \mathbf{c} \in \mathcal{C}.$$

Thus, whereas the generator matrix defines the vector subspace which is the code, the parity check matrix defines the orthogonal subspace. The codes generated by $\mathcal{C}$ and H are said to be duals of one another. The dual of $\mathcal{C}$ is denoted by $\mathcal{C}^{\perp}$. For the (4, 2) code that we considered earlier, the parity check matrix is the same as the generator matrix G shown in Figure 2. Such codes are referred to as *self-dual* codes. Each row of the parity check matrix can be thought of as a constraint that the bits of the codeword must satisfy. Thus a codeword is one for which the bits simultaneously satisfy all the constraints imposed by the rows of the parity check matrix.

A general block code also has a *combinatorial* description in the form of a *trellis*. We borrow from Kschischang and Sorokine[9], the definition of a trellis for a block code.

A trellis T for a block code $\mathcal{C}$ of length n, is an edge-labelled directed graph (V, E, l) with a distinguished root vertex s, having in-degree 0 and a distinguished goal vertex f having out-degree 0, with the following properties:

(i) V is the set of all vertices and all vertices can be reached from the root.
(ii) The goal can be reached from all vertices.
(iii) E is the set of edges and the number of edges traversed in passing from the root to the goal along any path is n. Each edge e has a label $l(e)$ from the input alphabet $\mathbb{F}_q$.
(iv) The set of n-tuples obtained by concatenating the edge labels encountered in traversing all paths from the root to the goal is $\mathcal{C}$.

The length of a path (in edges) from the root to any vertex is unique and is sometimes called the *time index* of the vertex. Accordingly, the vertex set V of the trellis is partitioned into n subclasses $V_0, V_1, \ldots V_{n-1}$, with V_i denoting the vertex set at time index i. (The class V_n is taken to be the same as V_0.) A *subtrellis* of T is a connected subgraph of T containing nodes at every time index i, $0 \leq i \leq n$ and all edges between them. Let S_i be the set of states corresponding to time index i, and $|S_i|$ denote the cardinality of the set S_i. Define $s_i = (\log_q|S_i|)$ and $s_{\max} = \max_i(s_i)$. The *state-complexity profile* of the code is defined as the sequence $(s_0, s_1, \ldots s_{n-1})$. Minimization of $s_{\max}$ is often desirable and $s_{\max}$ is referred to as the *maximum state-complexity*. There are several measures of the size of a trellis. Some of these are $s_{\max}$, the total number of states, the total number of edges and the vector representing the state complexity profile. Minimality of a trellis can be defined with respect to each of these measures[10]. It is well-known that minimal trellises for linear block codes are unique[11,12] and simultaneously satisfy all measures of minimality. Such trellises are known to be *biproper*, which means that no two edges entering a node or leaving a node have the same label. Figure 3 shows a trellis for the linear code in Figure 2.

There are several constructions of minimal conventional trellises from algebraic descriptions of codes. We will focus on two of these, namely the product construction of Kschischang and Sorokine[9], henceforth referred to as the KS construction, and the labelling scheme of Bahl, Cocke, Jelinek and Raviv[13], henceforth referred to as the BCJR construction. The first of these produces a trellis from a generator matrix. The structure of the trellis is critically dependent on the form of the generator matrix which has to satisfy a special property to yield the minimal trellis. The second uses the parity check matrix to generate the trellis and produces the minimal trellis irrespective of the form of the parity check matrix.

We first present the KS construction for constructing a minimal trellis from a generator matrix for the code.

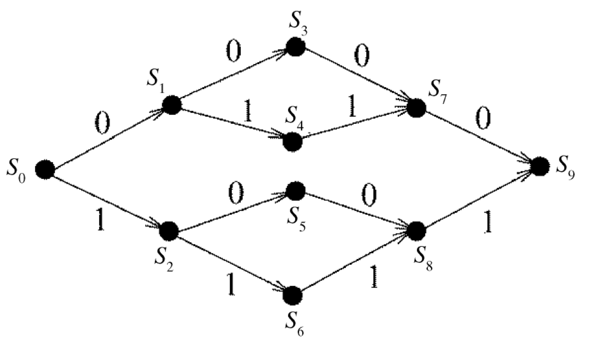

Figure 3. A trellis for the linear block code of Figure 2 with $S_0 = s$ and $S_9 = f$.

Each row of the generator matrix is associated with an *elementary trellis* and the trellis for the whole code is obtained by a *trellis product* operation on the set of elementary trellises for the rows of the generator matrix. Let T_1 and T_2 be the component trellises. We wish to construct the trellis product $T_1.T_2$. The set of vertices of the product trellis at each time index is just the Cartesian product of the vertices of the component trellis. Thus, if i is a time index, $V_i(T_1.T_2) = V_i(T_1) \times V_i(T_2)$. Consider $E_i(T_1) \times E_i(T_2)$ and interpret an element $((v_1,\alpha_1,v_1'),\ (v_2,\alpha_2,v_2'))$ in this product, where v_i, v_i' are vertices and α_1, α_2 edge labels, as the edge $((v_1,v_2),\alpha_1+\alpha_2,(v_1',v_2'))$, where + denotes addition in $\mathbb{F}_q$. If we define the ith section as the set of edges connecting the vertices at time index i to those at time index $i + 1$, then the edge count in the ith section is the product of the edge counts in the ith section of the individual trellises.

Before the product is constructed we put the matrix in *trellis-oriented form* described now. Given a nonzero codeword $\mathbf{c} = (c_1, c_2, \dots c_n)$, *start*($\mathbf{c}$) is the smallest integer i such that c_i is nonzero. Also, *end*($\mathbf{c}$) is the largest integer for which c_i is nonzero. The *linear span* of $\mathbf{c}$ is [*start*($\mathbf{c}$), *end*($\mathbf{c}$)]. By convention, the span of the all 0 codeword $\mathbf{0}$ is the empty span [ ]. The minimal trellis for the binary $(n, 1)$ code generated by a nonzero codeword with span $[a, b]$ is constructed as follows. There is only one path up to $a - 1$ from index 0, and from b to n. From $a - 1$, there are two outgoing branches diverging (corresponding to the two multiples of the codeword), and from $b - 1$ to b, there are two branches converging. For a code over $\mathbb{F}_q$, there will be q outgoing branches and q converging branches. It is easy to see that this is the minimal trellis for the one-dimensional code, and is called the elementary trellis corresponding to the codeword. To generate the minimal trellis for $\mathcal{C}$ we first put the trellis into trellis-oriented form, where for every pair of rows with spans $[a_1, b_1]$, $[a_2, b_2]$, $a_1 \neq b_1$ and $a_2 \neq b_2$. We then construct individual trellises for the k one-dimensional codes as described above, and then form the trellis product. Clearly, the number of states at each time index is always a power of q. We see that the generator matrices

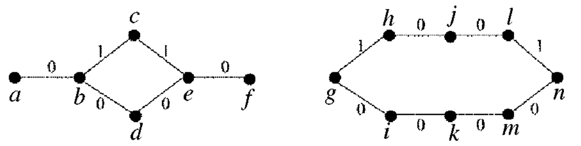

Figure 4. Elementary trellises for the rows of the generator matrix in Figure 2.

displayed earlier are already in trellis-oriented form. The elementary trellises for the two rows of the generator matrix in Figure 2 are shown in Figure 4. The product of these two elementary trellises yields the trellis in Figure 3.

We now summarize the BCJR construction of a trellis from a parity check matrix for a code. The interesting property of this construction is that it generates the minimal trellis *irrespective* of the form of the parity check matrix. We refer to the labelling of vertices generated by this construction as the BCJR labelling of the trellis. It is well known[10] that the set of vectors that are labels at each time index form a vector space whose dimension is the state-complexity at that time index. Let H be the parity check matrix for a (n, k) linear block code over $\mathbb{F}_q$, and let $\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_n$ be the columns of H. Each codeword $\mathbf{c} = (c_1, \dots, c_n)$ of the code gives rise to a sequence of states $\{\mathbf{s}_i\}_{i=0}^{n}$, each state being labelled by an $(n - k) \times 1$ vector as follows:

$$\mathbf{s}_i = \begin{cases} 0 & \text{if } i = 0 \\ \mathbf{s}_{i-1} + c_i\mathbf{h}_i & \text{otherwise.} \end{cases}$$

An interesting property that can be proved using the BCJR labelling scheme for trellises is the following:

Theorem 1. Let $\mathcal{C}$ be an (n, k) linear block code. Then the state complexity profiles of the minimal trellises for both $\mathcal{C}$ and $\mathcal{C}^{\perp}$ are the same.

Though the minimal trellises for a linear code and its dual have the same state complexity profiles, the edge complexity profiles can be very different. This difference is useful in certain decoding algorithms[14]. The BCJR labelling was used to introduce a useful decoding algorithm often called the forward–backward algorithm[13]. In contrast to the Viterbi algorithm, the BCJR algorithm performs maximum *a posteriori* symbol detection. The labelling scheme allows one to prove several important results for linear trellises, for example, Theorem 1.

We now introduce tail-biting trellises in a setting that leads to the decoding algorithm presented later. As we have mentioned earlier, every linear code has a unique minimal biproper trellis; so this is our starting point. Our objective is to describe an operation which we term *subtrellis overlaying*, that yields a smaller trellis. Reduction in the size of a trellis is a step in the direction of reducing decoder complexity.

One way of constructing tail-biting trellises is to partition the code $\mathcal{C}$ into disjoint subcodes, and 'overlay' the subtrellises corresponding to these subcodes to get a smaller, 'shared' trellis. An example will illustrate the procedure.

Example 1: Let $\mathcal{C}$ be the linear (4, 2) code defined by the generator matrix in Figure 2. $\mathcal{C}$ consists of the set of codewords {0000, 0110, 1001, 1111} and is described by the minimal trellis in Figure 3. The state-complexity profile of the code is (0, 1, 2, 1). Now partition $\mathcal{C}$ into subcodes $\mathcal{C}_1$ and $\mathcal{C}_2$ as follows:

$$\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2; \quad \mathcal{C}_1 = \{0000, 0110\}; \quad \mathcal{C}_2 = \{1001, 1111\},$$

with minimal trellises shown in Figure 5 *a* and *b*, respectively.

The next step is the 'overlaying' of the subtrellises as follows. There are as many states as time index 1 and time index n as partitions of $\mathcal{C}$. States (s_2, s_2'), (s_3, s_3'), (s_1, s_1'), (s_4, s_4') are superimposed to obtain the trellis in Figure 6.

Note that overlaying may increase the state-cardinality at some time indices (other than 0 and n), and decrease it at others. Codewords are represented by (s_0^i, s_f^i) paths in the overlayed trellis, where s_0^i and s_f^i are the start and final states of subtrellis i. Thus paths from s_0 to s_5 and from s_0' to s_5' represent codewords in the overlayed trellis of Figure 6. Overlaying causes subtrellises for subcodes to 'share' states. Note that the shared trellis is also biproper, with $s_{\max} = 1$ and state-complexity profile (1, 0, 1, 0).

The small example above illustrates two important points. First, it is possible to get a trellis with a smaller number of states to define essentially the same code as the original trellis, with the new trellis having several start and final states, and with a restricted definition of paths corresponding to codewords. Secondly, the new trellis is obtained by the superposition of smaller trellises so that some states are shared. It is shown[15,16] that overlaying of trellises to obtain tail-biting trellises requires the decomposition of the code $\mathcal{C}$ into a subgroup and its cosets. The subgroup and its cosets correspond to structurally identical subtrellises and overlaying is advantageous, if and only if it is possible to choose coset leaders satisfying certain conditions. It is shown[17] that for a given linear code it is possible to reduce the maximum state complexity to the square root of the corresponding complexity in the minimal conventional trellis using techniques equivalent to overlaying.

Two questions that naturally arise at this point are, first, whether there are KV and BCJR-like constructions for tail-biting trellises and secondly, if there is a counterpart to Theorem 1 for tail-biting trellises. Koetter and Vardy[18–20] in a series of articles have extended the notion of a span and shown that a product construction exists for tail-biting trellises. In the next section we develop a framework for a modified BCJR construction and show that there is an affirmative answer to the second question.

We first define a *circular span*[18] of a codeword. Let $\mathbf{c} = (c_1, c_2 \ldots c_n)$ be a codeword such that c_i and c_j are nonzero, $i < j$ and all components between i and j are zero.

Then $[j, i]$ is said to be a *circular span* of the codeword. Note that while the linear span of a codeword is unique, a circular span is not, as it depends on the consecutive run of zeros chosen. Given a codeword and a circular span $[j, i]$, there is a unique elementary trellis corresponding to it. If the code symbols are from $\mathbb{F}_q$, then the trellis has q states from index 0 to index $i-1$, one state from index i to index j, and q states from index $j+1$ to index n. If the start states are numbered from 1 to q and final states likewise, only i to i paths are codewords. Any linear tail-biting trellis can be constructed from a generator matrix whose rows can be partitioned into two sets, those which are taken to have linear span, and those taken to have circular span[18]. The tail-biting trellis is then formed as a product of the elementary trellises corresponding to these rows. We call this trellis a KSKV trellis, as it uses a modified KS construction due to Koetter and Vardy[18]. We will represent such a generator matrix as $G = [\frac{G_l}{G_c}]$, where G_l is the submatrix consisting of rows with linear span, and G_c the submatrix of rows with circular span. Koetter and Vardy[19] have shown that for several definitions of minimality, including $s_{\max}$-minimality, the spans chosen for the rows of the generator matrix must satisfy the condition that no two of them start at the same position and no two of them end at the same position. If this condition is satisfied, then G is said to be in *minimal span form*. Thus the problem of constructing a minimal tail-biting trellis is then reduced to finding a basis

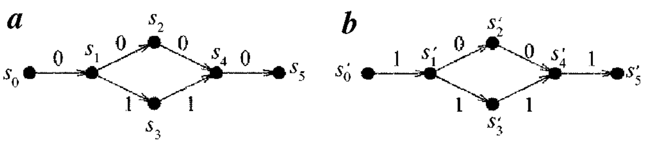

Figure 5. Minimal trellises for (*a*) $\mathcal{C}_1$ = {0000, 0110}, and (*b*) $\mathcal{C}_2$ = {1001, 1111}.

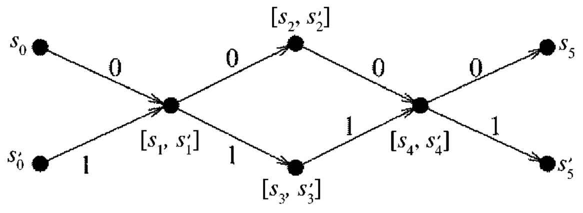

Figure 6. Trellis obtained by overlaying trellises in Figure 5 *a* and *b*.

for the subspace constituting the code, and a choice of spans, such that the corresponding elementary trellises yield a product that corresponds to a minimal tail-biting trellis under the appropriate definition of minimality.

A trellis T is *non-mergeable* if there exist vertices in the same vertex class of T that can be replaced by a single vertex, while retaining the edges incident on the original vertices, without modifying $\mathcal{C}(T)$. Koetter and Vardy[19] have shown that if a linear trellis is non-mergeable, then it is also biproper.

Obtaining dual tail-biting trellises

Let $\mathcal{C}$ be an (n, k) linear code over $\mathbb{F}_q$ with generator matrix G with row set $\{\mathbf{g}_1, \mathbf{g}_2, \dots, \mathbf{g}_k\}$. Let submatrix G_l contain the vectors of linear span and G_c contain the vectors of circular span. Let $H = [\mathbf{h}_1, \mathbf{h}_2 \dots \mathbf{h}_n]$ be the parity check matrix, where $\mathbf{h}_i$ denotes column i. The algorithm BCJR–TBT[21] constructs a non-mergeable linear tail-biting trellis T, given G and H.

Informally speaking, the algorithm after constructing the BCJR-labelled trellis for the subcode consisting of the rows of linear span, adds states and edges for linear combinations of each row $\mathbf{g}_c$ with circular span, in turn, by offsetting BCJR-like label computations with a state vector that is formed by beginning the label computations for $\mathbf{g}_c$ at the start of the circular span and proceeding in a circular order, rather than beginning at time index 0 and proceeding in linear order, as is performed for rows of linear span. The intermediate generator matrix which is formed by including one row of circular span at a time is called G_{int}. Let $\langle g_i \rangle$ denote the subspace generated by g_i.

Algorithm BCJR–TBT

```
Input:   The matrices $G$ and $H$.
Output:  A non-mergeable linear tail-biting trellis $T = (V, E, \mathbb{F}_q)$ representing $\mathcal{C}$.
Initialization:  $G_{\text{int}} = G_l$. Let $\{\mathbf{d_x}\}_{\mathbf{x} \in \mathcal{C}}$ as follows:
    $\mathbf{d_x} = \begin{cases} \sum_{j=a}^{n} x_j \mathbf{h}_j & \text{if } \mathbf{x} \in \langle \mathbf{g}_i \rangle, \mathbf{g}_i \text{ is a row of } G_c \text{ with circular span } [a,b] \\ 0 & \text{otherwise.} \end{cases}$
Step 1:  Construct the BCJR-labelled trellis for the subcode generated by the submatrix $G_l$, but using the matrix $H$ instead of the parity check matrix for the code $G_l$. Let $V_0, V_1 \dots V_n$ be the vertex sets created and $E_1, E_2, \dots E_n$ be the edge sets created.
Step 2:  for each row vector $\mathbf{g}$ of $G_c$
    for each $\mathbf{x} \in \langle \mathbf{g} \rangle$, $\mathbf{y}$ in the rowspace of $G_{\text{int}}$.
    {
    Let $\mathbf{z}$ denote the codeword $\mathbf{x} + \mathbf{y}$.
    let $\mathbf{d_z} = \mathbf{d_x} + \mathbf{d_y}$.
    $V_0 = V_n = V_0 \cup \{\mathbf{d_z}\}$.
    $V_i = V_i \cup \left\{ \mathbf{d_z} + \sum_{j=1}^{i} z_j \mathbf{h}_j \right\}, 1 \le i < n.$
    There is an edge $e = (\mathbf{u}, z_i, \mathbf{v}) \in E_i$, $\mathbf{u} \in V_{i-1}$, $\mathbf{v} \in V_i$, $1 \le i \le n$, iff $\mathbf{d_z} + \sum_{j=1}^{i-1} z_j \mathbf{h}_j = \mathbf{u}$ and
    $\mathbf{d_z} + \sum_{j=1}^{i} z_j \mathbf{h}_j = \mathbf{v}$.
    }
    $G_{\text{int}} = G_{\text{int}} + \mathbf{g}$.
```

The following properties of the resulting trellis T are proved in Nori and Shankar[22].

(i) The trellis T is linear and non-mergeable, and represents $\mathcal{C}$.

(ii) The trellis T is isomorphic to the KSKV trellis, iff G is in minimal-span form and results in a non-mergeable product trellis.

From the properties above and from the result of Koetter and Vardy[19] that all linear tail-biting trellises arise from product constructions, we conclude that all non-mergeable linear trellises can be obtained by using the algorithm BCJR–TBT on an appropriate generator matrix.

Example 2: Consider the (7, 4) Hamming code defined by the parity check matrix H and the generator matrix G (annotated with spans):

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} \begin{matrix} (1,6] \\ (6,2] \\ (3,7] \\ (7,5] \end{matrix}$$

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

The BCJR–TBT construction for this choice of spans is associated with the rows of G and is illustrated in Figure 7.

Note that this choice of spans is not good from the point of view of minimizing the value of the maximum state complexity ($s_{\max}$) of the tail-biting trellis. It has been chosen to illustrate that the BCJR–TBT is dependent on the form of the generator and parity check matrices chosen, unlike the trellis produced by the BCJR construction for the conventional case.

We will now describe the algorithm BCJR–TBT$^{\perp}$ that takes G and H as inputs and computes a non-mergeable linear tail-biting trellis $T^{\perp}$ for the dual code $\mathcal{C}^{\perp}$ (ref. 21).

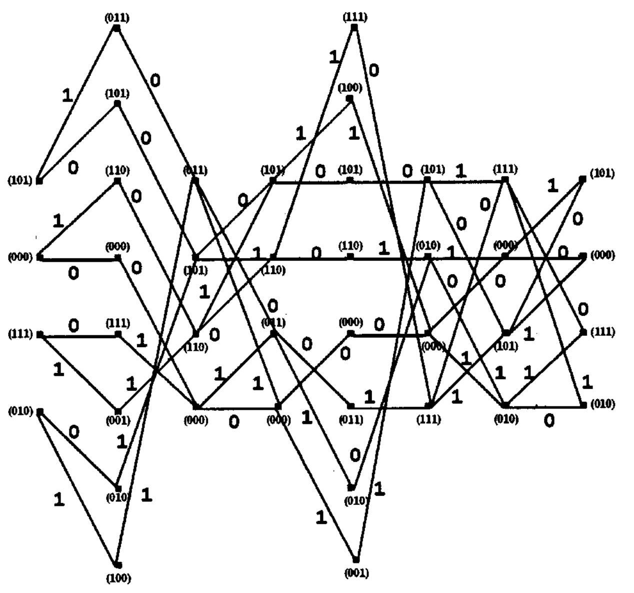

Figure 7. BCJR–TBT for given spans.

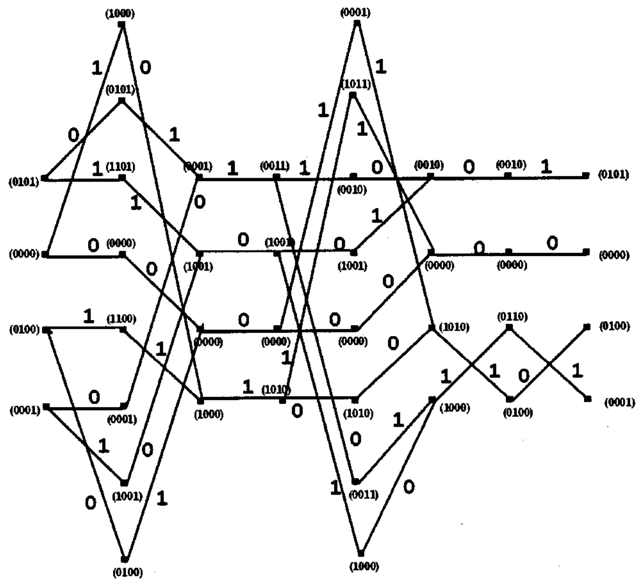

Figure 8. BCJR–TBT$^{\perp}$ for given spans.

Algorithm BCJR–TBT$^{\perp}$

```
Input: The matrices $G$ and $H$.
Output: A non-mergeable tail-biting trellis $T^{\perp} = (V, E, \mathbb{F}_q)$ representing $\mathcal{C}^{\perp}$.
Initialization: $V_i|_{0 \le i \le n} = E_i|_{1 \le i \le n} = \phi$.
  for each $\mathbf{y} = (y_1, y_2, \ldots, y_n) \in \mathcal{C}^{\perp}$.
  {
    let $\mathbf{d}^T = (d_1\ d_2 \ldots d_k)$ s.t. $d_i = \begin{cases} 0 & \text{if } 1 \le i \le l \\ \sum_{j=a}^{n} y_j g_{i,j} & \text{otherwise,} \end{cases}$
    where $\mathbf{g}_i \in G$ has circular span $(a, b]$.
    $V_0 = V_n = V_0 \cup \{\mathbf{d}\}$.
    $V_i = V_i \cup \left\{ \mathbf{d} + \sum_{j=1}^{i} y_j (g_{j,1}, g_{j,2} \ldots g_{j,k})^T \right\}$.
    There is an edge $e = (\mathbf{u}, z_i, \mathbf{v}) \in E_i$, $\mathbf{u} \in V_{i-1}$,
      $\mathbf{v} \in V_i$, $1 \le i \le n$, iff
    $\mathbf{d} + \sum_{j=1}^{i} y_j (g_{j,1}, g_{j,2}, \ldots, g_{j,k})^T = \mathbf{u}$, and
    $\mathbf{d} + \sum_{j=1}^{i} y_j (g_{j,1}, g_{j,2}, \ldots, g_{j,k})^T = \mathbf{v}$.
  }
```

Example 3: The BCJR–TBT$^{\perp}$ construction for the Hamming code from Example 2 with choice of spans $\mathcal{S}$ for the generator matrix is illustrated in Figure 8. It can be seen that the dual trellis has the same state-complexity profile as the primal trellis in Figure 7.

The following results are proved in Nori and Shankar[22].

(i) The trellis $T^{\perp}$ is a non-mergeable linear trellis that represents $\mathcal{C}^{\perp}$.

(ii) The state complexity profiles of T and $T^{\perp}$ are identical.

We thus have the following theorem:

Theorem 2. Let T be a non-mergeable linear trellis, either conventional or tail-biting, for a linear code $\mathcal{C}$. Then there exists a non-mergeable linear dual trellis $T^{\perp}$ for $\mathcal{C}^{\perp}$, such that the state-complexity profile of $T^{\perp}$ is identical to the state-complexity profile of T.

Finally, we know that for tail-biting trellises there are several measures of minimality. If any of these definitions requires the trellis to be non-mergeable, it immediately follows that there exist under that definition of minimality, minimal trellises for a code and its dual with identical state-complexity profiles.

Decoding on tail-biting trellises

We now propose an optimal decoding algorithm[15] that makes critical use of the fact that tail-biting trellises can be viewed as the superposition of subtrellises that share states at certain time indices. We first describe two algorithms on which the optimal algorithm is based, the Viterbi algorithm mentioned earlier and the A^* algorithm, well-known in the artificial intelligence community.

The Viterbi algorithm

Assume that a cocdeword $\mathbf{c}$ of length n is transmitted over the unreliable channel with input alphabet $A = \mathbb{F}_q$ and output alphabet B. Define

$f(./x): B \mapsto [0, 1]\ \forall x \in \mathbb{F}_q$.

If B is a discrete set, the channel is completely characterized by q known probability mass functions; if B is continuous, say $B = \mathbb{R}$, then $f(./x)$ are continuous probability density functions. To simplify the notation, assume B is discrete. The quantity $f(y/x)$ is the probability of receiving symbol y given that symbol x was transmitted. Let $\mathbf{y}$ be the received vector. The optimal decoding strategy is one that finds a codeword $\mathbf{c} = (c_1, c_2, \dots c_n)$ that maximizes $Pr\{\mathbf{y}/\mathbf{c}\}$. Assume that all codewords are equally likely to be transmitted and that each symbol transmitted over the channel is affected independently by the channel noise. Then if $\mathbf{y} = (y_1, y_2, \dots y_n)$ we have

$$Pr\{\mathbf{y}/\mathbf{c}\} = \prod_{i=1}^{n} f(y_i / c_i).$$

Maximizing this probability is equivalent to minimizing the sum

$$\sum_{i=1}^{n} -\log(f(y_i / c_i))$$

over all codewords. If we relabel each edge e in the trellis for the code $\mathcal{C}$ by $l'(e) = -\log(f(y_i/c_i))$, then finding a codeword that maximizes the probability above is the same as finding a shortest path from s to f through the trellis where the edge lengths are given by the function $l'(.)$. This is exactly what the Viterbi algorithm computes. Since the trellis is a regular layered graph, the algorithm proceeds level-by-level, computing a *survivor* at each node; this is a shortest path to the node from the source. For each branch b, leaving a node at level i, the algorithm updates the survivor at that node by adding the cost of the branch to the value of the survivor. For each node at level $i + 1$, it compares the values of the path cost for each branch entering the node and chooses the one with minimum value. There will thus be only one survivor at the goal vertex, and this corresponds to the decoded codeword. For an overlayed trellis we are interested only in paths that go from s_i to f_i, $0 \le i \le t$, where t is the number of overlayed subtrellises.

The A algorithm*

The A^* algorithm is well-known in the literature on artificial intelligence[23]. The A^* algorithm uses, in addition to the path length from the source to the node u, an estimate $h(u, f)$ of the shortest path length from the node u to the goal node in guiding the search. Let $L_T(u, f)$ be the shortest path length from u to f in T. Let $h(u, f)$ be any lower bound such that $h(u, f) \le L_T(u, f)$, and such that $h(u, f)$ satisfies the following inequality, i.e. for u a predecessor of v, $l(u, v) + h(v, f) \ge h(u, f)$. If both the above conditions are satisfied, then the algorithm A^*, on termination, is guaranteed to output a shortest path from s to f. The algorithm is given below.

*Algorithm A**

Input: A trellis $T = (V, E, l')$, where V is the set of vertices, E is the set of edges and $l'(u, v) \ge 0$ is a length for edge (u, v) in E, a source vertex s and a destination vertex f, and an estimate $h(u, f)$ for the shortest path from u to f for each vertex $u \in V$.

Output: The shortest path from s to f.

```
/*p(u) is the cost of the current shortest path from s to u
and P(u) is a current shortest path from s to u */
begin
  $S \leftarrow \varnothing$, $\bar{S} \leftarrow \{s\}$, $p(s) \leftarrow 0$, $P(u) \leftarrow (\,)$, $\forall u \in V$,
    $p(u) = +\infty$, $\forall u \neq s$;
  repeat
    Let $u$ be the vertex in $\bar{S}$ with minimum value of
      $p(u) + h(u, f)$.
    $S \leftarrow S \cup \{u\}$;   $\bar{S} \leftarrow \bar{S} \setminus \{u\}$;
    if $u = f$ then return $P(f)$;
    for each $(u, v) \in E$ do
      if $v \notin S$ then
        begin
          $p(v) \leftarrow \min(p(u) + l'(u, v), previous\ (p(v)))$;
          if $p(v) \neq previous\ (p(v))$ then append $(u, v)$ to
            $P(u)$ to give $P(v)$;
          $(\bar{S}) \leftarrow (\bar{S}) \cup \{v\}$;
        end
    forever
end
```

Our algorithm is a two-phase algorithm that uses a Viterbi trial in the first phase to deliver estimates which are used by an A^* algorithm in the second phase. It has been proved in Shankar *et al.*[15] that these estimates satisfy the properties which guarantee that when the A^* algorithm terminates, it indeed delivers the shortest codeword path in the tail-biting trellis.

We first define the notation we will be using. Let T_B be the tail-biting trellis and $T_{\mathcal{C}}$ the conventional trellis for the code under consideration. Let $s_1, s_2, \dots s_t$ be the set of start states and $f_1, f_2, \dots f_t$ be the set of final nodes of the tail-biting trellis. The t sub-trellises are denoted by $T_1, T_2, \dots T_t$, with sub-trellis T_j corresponding to sub-code $\mathcal{C}_j$. The estimate or lower bound for a shortest path from a node u in a sub-trellis to the final node f_j in that sub-trellis is denoted by $h(u, f_j)$. A Viterbi trial on the tail-biting trellis finds shortest paths to the destination nodes $f_1, f_2, \dots f_t$ from *any* of the source nodes $s_1, s_2, \dots s_t$. A *survivor* at any intermediate node u is a shortest path from a source node s_i to u. A *winning* path or a *winner* at node f_j is a survivor at node f_j and its cost is denoted by $h(s_j, f_j)$, as it turns out to be a lower bound on the cost of a shortest path from s_j to f_j. (Note that such a path need not start at s_j.) Let us term the codeword that would be the final survivor if Viterbi decoding was performed on subtrellis T_j alone, as the *T_j-codeword survivor*. We term a survivor on the tail-biting trellis, which corresponds to an $s_i - f_i$ path, a *codeword survivor*, and one that corresponds to an $s_i - f_j$ path, $i \neq j$, a *non-codeword survivor*.

The two-phase algorithm

The two-phase algorithm uses a structure called a *heap*. For purposes of this discussion, a heap stores a set of elements which can be linearly ordered and can deliver the minimal element in constant time. The size of the heap is the number of elements it stores. Elements can be added to the heap and the cost of restructuring the heap so that it can deliver the minimal element in constant time is logarithmic in the size of the heap.

The two phases of the algorithm are described below.

Phase 1:
Execute a Viterbi decoding algorithm on the tail-biting trellis, and obtain survivors at each node. It is easy to see that a survivor at a node u has a cost which is a lower bound on the cost of the least cost path from s_j to u in an $s_j - f_j$ path passing through u, $1 \leq j \leq t$. If there exists a value of k for which an $s_k - f_k$ path is an overall winner, then this is the shortest path in the original trellis T_C. If this happens, decoding is complete. If no such $s_k - f_k$ path exists, go to Phase 2.

Phase 2:
(i) Consider only subtrellises T_j such that the winning path at T_j is an $s_i - f_j$ path with $i \neq j$ (i.e. at some intermediate node a prefix of the $s_j - f_j$ path was 'knocked out' by a shorter path originating at s_i), and such that there is no $s_k - f_k$ path with smaller cost. Let us call such trellises *residual trellises*. Let the estimate $h(s_j, f_j)$ associated with the node s_j be the cost of the survivor at f_j obtained in the first phase.
(ii) Create a heap of r elements where r is the number of residual trellises, with current estimate $h(s_j, f_j)$ with minimum value as the top element. Let j be the index of the subtrellis with the minimum value of the estimate. Remove the minimum element corresponding to T_j from the heap and run the A^* algorithm on trellis T_j (called the *current trellis*). For a node u, take $h(u, f_j)$ to be $h(s_j, f_j) - cost(survivor(u))$ where $cost(survivor(u))$ is the cost of the survivor obtained in the first phase. The quantity $h(u, f_j)$ satisfies the two properties required of the estimator in the A^* algorithm.
(iii) At each step, compare $p(u) + h(u, f_j)$ in the current subtrellis with the top value in the heap. If at any step the former exceeds the latter (associated with subtrellis, say, T_k), then make T_k the current subtrellis. Insert the current value of $p(u) + h(u, f_j)$ in the heap (after deleting the minimum element) and run the A^* algorithm on T_k either from start node s_k (if T_k was not visited earlier) or from the node which it last expanded in T_k. Stop when the goal vertex is reached in the current subtrellis.

In the best case (if the algorithm needs to execute Phase 2 at all), the search will be restricted to a single residual subtrellis.

We have run simulations of the algorithm on several tail-biting trellises for block codes[24] on an additive white Gaussian noise (AWGN) channel. Below, we give the parameters for the well-known[3,9] Golay code for the conventional as well as the tail-biting trellis given in Calderbank *et al.*[25].

Extended (24, 12) binary Golay code
Tail-biting trellis:
Number of branches = 384; Number of states = 192;
State complexity profile = (4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4).

Conventional 12-section trellis:
Number of branches = 2728; Number of states = 1065;
State complexity profile = (0, 2, 4, 6, 6, 8, 8, 8, 6, 6, 4, 2).

Figure 9 shows the number of nodes at which computations are performed using our algorithm as a function of signal-to-noise ratio (SNR), the latter being a measure of how noisy a channel is (the higher the SNR, the more reliable the channel). The algorithm displays a curious property. The average number of nodes examined is always less than twice the number of nodes in the tail-biting trellis, thereby showing that effectively the average computation is less than two rounds of the tail-biting trellises (with some additional low overheads incurred by the sorting and heap operations). Typically, most approximate algorithms iterate around the tail-biting trellis for a

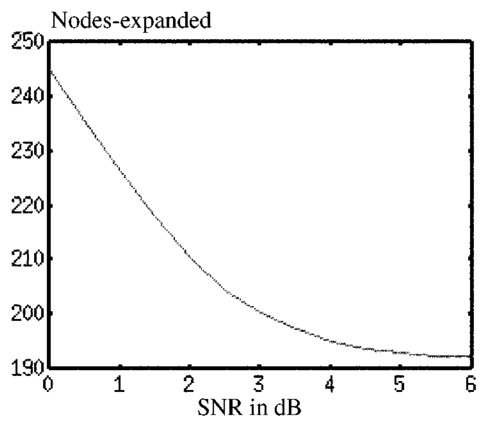

Figure 9. Number of nodes examined in the two-phase algorithm for the (24, 12) Golay code.

finite number of iterations, and it is observed that the decoder eventually locks onto a path closest to the received codeword[26]. The number of such iterations is typically around 4 or 5. Here, we see that after an amount of computation, which on the average is less than two rounds of the tail-biting trellises, we get the optimal decoded codeword. Detailed simulation results have been reported in Shankar *et al.*[27].

Conclusion

Connections between algebraic and combinatorial views of linear block codes have been described. It is shown that viewing a tail-biting trellis as a coset decomposition of the group representing the code with respect to a subgroup satisfying special properties, leads to a decoding algorithm that appears promising from the viewpoint of computational complexity. Two problems that merit further study are the complexity of the problem of finding the $s_{\max}$-minimal tail-biting trellis, and a theoretical study of the performance of the two-phase decoding algorithm.

1. Berlekamp, E. W., *Algebraic Coding Theory*, McGraw Hill, New York, 1968.
2. Massey, J. L., Shift-register synthesis and BCH decoding. *IEEE Trans. Inf. Theory*, 1969, **15**, 122–127.
3. Sugiyama, Y., Kasahara, M., Hirasawa, S. and Namekawa, T., A method for solving the key equation for decoding Goppa codes. *Inf. Control*, 1975, **27**, 87–99.
4. Guruswami, V. and Sudan, M., Improved decoding of Reed–Solomon codes and algebraic–geometric codes. *IEEE Trans. Inf. Theory*, 1999, **45**, 1757–1767.
5. Viterbi, A. J., Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Trans. Inf. Theory*, 1967, **13**, 260–269.
6. McEliece, R. J., *The Theory of Information and Coding*, Encyclopedia of Mathematics and its Applications, Addison Wesley, 1977.
7. MacWilliams, F. J. and Sloane, N. J. A., *The Theory of Error Correcting Codes*, North-Holland, Amsterdam, 1981.
8. Blahut, Richard, E., *Algebraic Codes for Data Transmission*, Cambridge University Press, 2003.
9. Kschischang, F. R. and Sorokine, V., On the trellis structure of block codes. *IEEE Trans. Inf. Theory*, 1995, **41**, 1924–1937.
10. Vardy, A., Trellis structure of codes. In *Handbook of Coding Theory* (eds Pless, V. S. and Huffman, W. C.), Elsevier, 1998.
11. Muder, D. J., Minimal trellises for block codes. *IEEE Trans. Inf. Theory*, 1988, **34**, 1049–1053.
12. McEliece, R. J., On the BCJR trellis for linear block codes. *IEEE Trans. Inf. Theory*, 1996, **42**, 1072–1092.
13. Bahl, L. R., Cocke, J., Jelinek, F. and Raviv, J., Optimal decoding of linear codes for minimizing symbol error rate. *IEEE Trans. Inf. Theory*, 1974, **20**, 284–287.
14. Berkmann, J. and Weiss, C., On dualizing trellis-based APP decoding algorithms. *IEEE Trans. Commun.*, 2002, **50**, 1743–1757.
15. Shankar, P., Dasgupta, A., Deshmukh, K. and Rajan, B. S., On viewing block codes as finite automata. *Theor. Comput. Sci.*, 2003, **290**, 1775–1795.
16. Shany, Y. and Be'ery, Y., Linear tail-biting trellises, the square-root bound, and applications for Reed–Muller codes. *IEEE Trans. Inf. Theory*, 2000, **46**, 1514–1523.
17. Wiberg, N., Codes and decoding on general graphs. Ph D thesis, Department of Electrical Engineering, Linkoping University, Linkoping, 1996.
18. Koetter, R. and Vardy, A., Construction of minimal tail-biting trellises. *Proceedings of the IEEE International Workshop on Information Theory*, Killarney, Ireland, 1998, pp. 72–74.
19. Koetter, R. and Vardy, A., On the theory of linear trellises. In *Information, Coding and Mathematics* (ed. Blaum, M.), Kluwer, Boston, 2002.
20. Koetter, R. and Vardy, A., The structure of tail-biting trellises: minimality and basic principles. *IEEE Trans. Inf. Theory*, 2002 (submitted).
21. Nori, A. and Shankar, P., A BCJR-like labelling scheme for tail-biting trellises. In *Proceedings 2003 IEEE International Symposium on Information Theory*, IEEE Press, 2003.
22. Nori, A. and Shankar, P., Forty-first Annual Allerton Conference on Communication Control and Computing, Allerton, October 2003 (to be presented).
23. Hart, P. E., Nilsson, N. J. and Raphael, B., A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Solid-State Circuits*, 1968, **SSC-4**, 100–107.
24. Shankar, P., Kumar, P. N. A., Sasidharan, K. and Rajan, B. S., ML decoding of block codes on their tail-biting trellises. In *Proceedings 2001 IEEE International Symposium on Information Theory*, IEEE Press, 2001, p. 291.
25. Calderbank, A. R., Forney, Jr., G. D. and Vardy, A., Minimal tail-biting trellises: The Golay code and more. *IEEE Trans. Inf. Theory*, 1999, **45**, 1435–1455.
26. Aji, S., Horn, G., McEliece, R. J. and Xu, M., Iterative min-sum decoding of tail-biting codes. In *Proceedings of the IEEE International Workshop on Information Theory*, Killarney, Ireland, 1998, pp. 68–69.
27. Shankar, P., Kumar, P. N. A., Sasidharan, K. and Rajan, B. S., Optimal maximum-likelihood decoding on tail-biting trellises. Technical Report IISc-CSA-PS-03-01, Department of Computer Science and Automation, Indian Institute of Science, Bangalore. (http://drona.csa.iisc.ernet.in/~priti).

Received 28 August 2003