On the BCJR Trellis for Linear Block Codes

Robert J. McEliece, *Fellow, IEEE*

Abstract— **In this semi-tutorial paper, we will investigate the computational complexity of an abstract version of the Viterbi algorithm on a trellis, and show that if the trellis has e edges, the complexity of the Viterbi algortithm is $\Theta(e)$. This result suggests that the "best" trellis representation for a given linear block code is the one with the fewest edges. We will then show that, among all trellises that represent a given code, the original trellis introduced by Bahl, Cocke, Jelinek, and Raviv in 1974, and later rediscovered by Wolf, Massey, and Forney, uniquely minimizes the edge count, as well as several other figures of merit. Following Forney and Kschischang and Sorokine, we will also discuss "trellis-oriented" or "minimal-span" generator matrices, which facilitate the calculation of the size of the BCJR trellis, as well as the actual construction of it.**

Index Terms— **Block code, trellis, Viterbi algorithm, decoding complexity.**

I. Introduction and Summary

In 1974, Bahl, Cocke, Jelinek, and Raviv [3], in a study of optimal bit error probability decoding algorithms, presented, for the first time, a method of representing the words in an arbitrary linear block code by the path labels in a trellis, thus uncovering an important connection between block and convolutional codes. In 1978, Wolf [43] introduced an identical trellis for block codes and showed that it could be used to implement the Viterbi algorithm for maximum-likelihood decoding of an arbitrary block code. Later that same year, Massey [29] made a further study of the problem of representing a block code by a trellis, and gave an alternative construction. For the next ten years, there was relatively little work in this area, but in 1988 Forney [11], in a now celebrated appendix to a paper on coset codes, described what he called "the trellis diagram of a code," which resulted in an explosion of interest in the subject. Of the post-Forney papers, among the most noteworthy are those of Muder [35] and Kschischang and Sorokine [22]. Muder showed that among all trellises representing a given block code, the Forney trellis minimized the number of vertices at each depth. For this reason, Muder called the Forney trellis the "minimal" trellis for the code, and the name has stuck. Kschischang and Sorokine, elaborating on a remark by Forney, developed many of the properties of the important "trellis-oriented" generator matrices for the first time. There have been many other significant contributions to the subject, including [4], [5], [8], [12], [13], [16], [18]–[21], [23], [25]–[27], [40], [42], and [44]. Most recently, in an unexpected turn of events, the theory of "minimal trellises" has been applied successfully to reducing the Viterbi decoding complexity of convolutional codes [33], [38].

In this paper, which is fundamentally tutorial, but which also contains a number of original results, we will take a fresh look at the problem of representing a given linear block code by a trellis. We will begin by studying the computational complexity of a generalized version of the Viterbi algorithm on a trellis, and conclude that this complexity is proportional to the number of edges in the trellis. Motivated by this result, we will then raise the question as to which trellis representing a given binary linear block code $\mathbb{C}$ has the fewest edges. We will show that this question has a surprising and satisfying answer, namely, that among all trellises representing $\mathbb{C}$, the BCJR trellis uniquely minimizes the edge count. Along the way, we will also show that the BCJR trellis is isomorphic to the Forney–Muder "minimal trellis," a historically important fact overlooked by Forney and Muder, but announced by Kot and Leung [21], and proved by Zyablov and Siderenko [44], in 1993. (It has recently been shown by Kschischang and Vardy [24] that the BCJR trellis also minimizes the number of "bifurcations," a number second only to the number of edges in determining the complexity of the Viterbi algorithm.) Pursuing an elliptic remark of Forney's, we will then introduce the class of "trellis-oriented," or as we shall call them, "minimal-span" generator matrices for block codes, and show how these matrices can be used to facilitate both the construction and analysis of the BCJR trellis.

Our approach in Sections III–VII is to begin with the original BCJR definition, and pursue its logical consequences. Along the way, we will derive a number of results, some new, but many already known. We will carefully attribute these results to the original discoverers, but the reader should bear in mind that most of these "known" results were derived for the "minimal" trellis, which was not known at the time to be isomorphic to the BCJR trellis.

Here is a brief outline of the rest of the paper. In Section II, we will define a trellis and present a generalized version of the Viterbi algorithm, whose goal is to compute certain "flows" in the trellis. We shall see that when this general algorithm is specialized appropriately, it can be used for finding the shortest paths through the trellis, or for computing the trellises's path weight enumerator, or for several other purposes. We will present a simple analysis of the generalized Viterbi algorithm, which shows that its computational complexity is $\Theta(e)$, where e is the number of edges in the trellis. We will conclude with a

Manuscript received October 28, 1994; revised December 11, 1995. This work was supported in part by AFOSR under Grant F4960-94-1-005, by a grant from Pacific Bell, and by NSF under Grant NCR-9505975. A portion of the work was also done at the Jet Propulsion Laboratory, California Institute of Technology, under Contract to the National Aeronautics and Space Administration.The material in this paper was presented in part at the IEEE International Symposium on Information Theory, Trondheim, Norway, June 1994.

The author is with the Department of Electrical Engineering, California Institute of Technology, Pasadena, CA 91103 USA.

Publisher Item Identifier S 0018-9448(96)04017-5.

discussion of the relationship of Viterbi's algorithm with other, *similar*, algorithms in the computer science literature.

In Section III, we will pose the problem of representing the words in a binary linear block code $\mathbb{C}$ by the paths in a trellis, and see that there are always many trellises that represent $\mathbb{C}$. Motivated by the results in Section II, however, we will argue that the "best" such trellis is the one or ones with the fewest edges, and allege that the BCJR trellis uniquely minimizes the edge count among all trellises representing $\mathbb{C}$. We will then review the BCJR construction, and give a formal proof, apparently the first one, of its correctness.

In Section IV, we will give the basic algebraic and combinatorial analysis of the BCJR trellis, culminating with Theorem 4.6, which gives a formula for the number of vertices and edges at each depth, in terms of the dimensions of the important past and future subcodes of $\mathbb{C}$, which were introduced by Forney. We also present an information-theoretic interpretation (Theorem 4.8) of the vertex dimensions of the BCJR trellis.

In Section V, we will give a proof that the BCJR trellis is the uniquely "minimal" trellis for $\mathbb{C}$, in a number of convincing ways, the most important being that it minimizes the number of edges. As a corollary, we will show that the BCJR trellis is isomorphic to the Forney trellis.

In Section VI, we will present the theory of "minimal-span" generator matrices (MSGM's), which are also called "trellis-oriented" generator matrices. We will show that MSGM's have many useful properties, among them that the important parameters of the BCJR trellis (the number of vertices and edges at each depth, the dimension of the past and future subcodes) can be read directly from them. In many ways MSGM's seem to be the optimal matrix representations for linear codes. As an application, we will show that the "Massey trellis" [29] is isomorphic to the BCJR trellis.

In Section VII, we will describe a general method for using a minimal-span generator matrix for $\mathbb{C}$ to construct the family of "simple linear" trellises for $\mathbb{C}$. We will show that when this method is specialized appropriately, the result is an efficient construction of the BCJR trellis. This method can also be used to constructed the "sectionalized" trellises discussed in [27].

Finally, in Section VIII, we will conclude with some remarks about the "Viterbi decoding complexity" of linear block codes, a subject we introduced in [32].

II. The Viterbi Algorithm for Computing Flows on a Trellis

In this section we will give a careful definition of what we mean by the *Viterbi algorithm on a trellis*, and show that its complexity is $\Theta(e)$, where e is the number of edges in the trellis.[1] We begin with a definition of a trellis, which is the same as the one given by Massey [29] or Muder [35], but couched in the standard terminology for directed graphs given by Stanley [39, sec. 4.7].

A *trellis* $T = (V, E)$ of rank n is a finite-directed graph, with vertex set V and edge set E, in which every vertex is assigned a "depth" in the range $\{0, 1, \cdots, n\}$, each edge connecting a vertex at depth $i-1$ to one at depth i, for some $i = 1, \cdots, n$. Multiple edges between vertices are allowed. The set of vertices at depth i is denoted by V_i, so that $V = \bigcup_{i=0}^{n} V_i$. The set of edges connecting vertices at depth $i-1$ to those at depth i is denoted $E_{i-1,i}$, so that $E = \bigcup_{i=1}^{n} E_{i-1,i}$. There is only one vertex at depth 0, called A, or the *source*, and only one at depth n, called B, or the *sink*. If $e \in E$ is a directed edge connecting the vertices u and v, which we denote by $e: u \rightarrow v$, we call u the *initial vertex*, and v the *final vertex*, of e, and write $\text{init}(e) = u$, $\text{fin}(e) = v$. We denote the number of edges leaving a vertex v by $\rho^+(v)$, and the number of edges entering a vertex v by $\rho^-(v)$, i.e.

$$\rho^+(v) = |\{e: \text{init}(e) = v\}| \tag{2.1}$$

$$\rho^-(v) = |\{e: \text{fin}(e) = v\}|. \tag{2.2}$$

If u and v are vertices, a *path* P *of length* L from u to v is a sequence of L edges: $P = e_1 e_2 \cdots e_L$, such that $\text{init}(e_1) = u$, $\text{fin}(e_L) = v$, and $\text{fin}(e_i) = \text{init}(e_{i+1})$, for $1 = 1, 2, \cdots, L-1$. If P is such a path, we sometimes write $P: u \rightarrow v$ for short. We denote the set of paths from vertices at depth i to vertices at depth j by $E_{i,j}$. We assume that for every vertex $v \neq A, B$, there is at least one path from A to v, and at least one path from v to B.

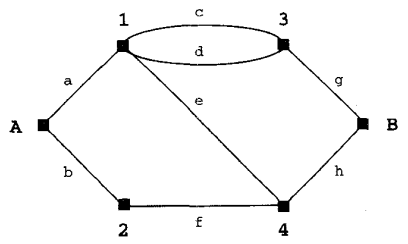

Fig. 1. Trellis of rank 3. The vertex set is $V = \{A, 1, 2, 3, 4, B\}$ and the edge set is $E = \{a, b, c, d, e, f, g, h\}$.

Example 2.1: In Fig. 1 we see a trellis of rank 3, with six vertices and eight edges. The vertex set is $V = \{A, 1, 2, 3, 4, B\}$, with $V_0 = \{A\}$, $V_1 = \{1, 2\}$, $V_2 = \{3, 4\}$, and $V_3 = \{B\}$. The edge set is $E = \{a, b, c, d, e, f, g, h\}$, with $E_{0,1} = \{a, b\}$, $E_{1,2} = \{c, d, e, f\}$, and $E_{2,3} = \{g, h\}$. There are two edges, c and d, connecting vertices 1 and 3, i.e., $\text{init}(c) = \text{init}(d) = 1$ and $\text{fin}(c) = \text{fin}(d) = 3$. We have $\rho^+(A) = 2$, $\rho^-(A) = 0$, $\rho^+(1) = 3$, $\rho^-(1) = 1$, etc. There are four paths from A to B; indeed, $E_{0,3} = \{acg, adg, aeh, bfh\}$.

We also assume each edge in the trellis is *labeled*. The labels come from an algebraic set S which is closed under the operation of two binary operations called "$\cdot$" and "+," which satisfy the following axioms:

The operation "$\cdot$" is associative, and there is an identity element "1" such that $s \cdot 1 = 1 \cdot s = s$ for all $s \in S$. This makes $(S, \cdot)$ a *monoid*(see [9, sec. 4.1]). (2.3)

The operation "+" is associative and commutative, and there is an identity element "0"such that $s + 0 = 0 + s = s$ for all $s \in S$. This makes $(S, +)$ a *commutative monoid*. (2.4)

[1] The notation $f(n) = \Theta(g(n))$ means that there exist positive constants c_1 and c_2 such that $c_1 g(n) \leq f(n) \leq c_2 g(n)$, for all sufficiently large n [7, sec. 2.1].

The distributive law

$$(x+y)\cdot z = (x\cdot z)+(y\cdot z), \quad \text{for all triples } (x,y,z) \text{from } S. \tag{2.5}$$

The triple $(S,\cdot,+)$ is called a *semiring* (see [1, sec. 5.6], or [7, sec. 26.4]). There are several important examples of semirings for our applications (see Examples 2.4–2.7, below).

Let $T=(V,E)$ be a trellis of rank n, such that each edge $e\in E$ is labeled with an element $\lambda(e)$ from a semiring $(S,\cdot,+)$. To indicate that the trellis is labeled, we denote it by $T=(V,E,\lambda)$. With the edges labels given, we now define the label of a path, and the flow between two vertices.

Definition 2.2: The *label of a path* $e_1e_2\cdots e_m$ is defined to be the product ("$\cdot$") $\lambda(e_1)\cdot\lambda(e_2)\cdots\lambda(e_m)$ of the labels of the edges in the path, taken in order. (Order matters, since the operation "$\cdot$" may not be commutative.)

Definition 2.3: If u and v are vertices in a labeled trellis, we define the *flow* from u to v, denoted by $\mu(u,v)$, to be the sum ("+") of the labels on all paths from u to v.[2]

The object of the Viterbi algorithm, when applied to a labeled trellis (V,E,λ), is to compute the flow from the source A to the sink B. This "flow" has different interpretations, depending on the particular semiring from which the labels come. The next four examples illustrate this.

Example 2.4: Let $S=\{0,1\}$, with "$\cdot$" being the Boolean AND operation, and "+" being OR. This is the simplest example of a semiring. If we interpret an edge labeled 0 as being "inactive," and one labeled 1 as "active," then in this case the "flow" $\mu(u,v)$ is 1 if there is a path from u to v, all of whose edges are "active," and otherwise it is 0.

Example 2.5: Let S be the set of nonnegative real numbers, plus the special symbol "∞." Define "$\cdot$" to be ordinary addition [*sic*], with the real number 0 playing the role of the identity required in (2.3). Define "+" be the operation of *taking the minimum*, with the special symbol ∞ playing the role of the identity element required in (2.4), i.e., $\min(s,\infty)=s$ for all real numbers s. It is easy to see that this definition produces a semiring, and if we interpret the label of an edge as its "length," the flow $\mu(u,v)$ is the length of the shortest path from u to v (see Example 2.13). This is the semiring appropriate for "Viterbi decoding," as we will see in Section III.

Example 2.6: Let S be the set of polynomials in one indeterminate x over the ring $\mathbb{Z}$ of ordinary integers, and let "$\cdot$" and "+" be as ordinarily defined. Then if the length of the edge e is denoted $l(e)$, and the edge e is labeled $x^{l(e)}$, with this semiring the flow $\mu(u,v)$ is the generating function for the lengths of the paths from u to v (see Example 2.13, below). This is the semiring appropriate for computing the weight enumerator for a code represented by a trellis, as we shall see in Section III. Similarly, if S is the set of rational functions in x, again with ordinary "$\cdot$" and "+," and if the trellis represents an interconnection of linear time-invariant systems, where the label $\lambda(e)$ is the transfer function between init (e) and fin (e), then $\mu(u,v)$ represents the overall transfer function, or gain, between u and v (see [36, sec. 9.7.2.]).

Example 2.7: Let S be a finite set of "letters," let "$\cdot$" denote string concatenation, and let "+" represent the operation of taking the union of a set of strings. When the Viterbi algorithm is applied in this case, the result (the flow from A to B) is the set of length-n strings over S corresponding to the labels on each of the paths from A to B. We call this set of strings the *language produced by the labeled trellis* (see Example 2.14, below). When the set S is $\{0,1\}$, the language produced by the trellis will be a binary code of length n. In Section III, we shall turn the tables and start with a binary code $\mathbb{C}$ of length n and try to construct a labeled trellis that produces $\mathbb{C}$ as efficiently as possible.

Here is a pseudocode description of the Viterbi algorithm ([10], [31, sec. 6.6], [41]). To simplify the notation, from now on, $\mu(x)$ will be used to denote the flow from A to x. As will be seen, the Viterbi algorithm successively computes $\mu(x)$ for all $x\in V_1,V_2,\cdots,V_n$, and finally returns the value of $\mu(B)$, which is the flow from A to B.

```
/* The Viterbi Algorithm on the
                                Trellis (V,E,λ) */
{
μ(A) = 1;
for (i = 1 to n) {
  for (v ∈ V_i)
*   μ(v) = Σ_{e: fin(e)=v} μ(init(e)) · λ(e);
  }
output  μ(B);
}
```

Example 2.8: If we apply the Viterbi algorithm to the labeled trellis in Fig. 1, we find, successively, that

$$\begin{aligned}
\mu(A) &= 1 \quad \text{(initialization)}\\
\mu(1) &= \mu(A)\cdot a = 1\cdot a = a\\
\mu(2) &= \mu(A)\cdot b = 1\cdot b = b\\
\mu(3) &= (\mu(1)\cdot c)+(\mu(1)\cdot d) = (a\cdot c)+(a\cdot d)\\
\mu(4) &= (\mu(1)\cdot e)+(\mu(2)\cdot f) = (a\cdot e)+(b\cdot f)\\
\mu(B) &= (\mu(3)\cdot g)+(\mu(4)\cdot h)\\
&= (((a\cdot c)+(a\cdot d))\cdot g)+(((a\cdot e)+(b\cdot f))\cdot h)\\
&= (a\cdot c\cdot g)+(a\cdot d\cdot g)+(a\cdot e\cdot h)+(b\cdot f\cdot h).
\end{aligned}$$

In this case, at least, the value computed by the Viterbi algorithm for $\mu(B)$ is the sum of the labels of the (four) paths from A to B. The next theorem proves that this is always true.

Theorem 2.9: The Viterbi algorithm correctly computes the flows $\mu(v)$, for all $v\in V$.

Proof: The proof is by induction on depth (v). For depth $(v)=1$, it follows from the definition of a trellis that all paths from A to v must consist of just one edge e, with init $(e)=A$ and fin $(e)=v$. Thus the true value of $\mu(v)$ is the sum of the labels on all edges joining A to v. (Recall that multiple edges between vertices are allowed.) On the other hand, when the algorithm computes $\mu(v)$ on line $*$, the value

[2] In [1, sec. 5.6], the analogous quantity is called the "cost" of going from u to v, and in [7, sec. 26.4], it is called the "summary" of all path labels from u to v.

it assigns to it is (because of the initialization $\mu(A) = 1$)

$$\begin{aligned}\mu(v) &= \sum_{e:\text{fin}(e)=v} \mu(\text{init}(e)) \cdot \lambda(e) \\ &= \sum_{e:A\to v} 1 \cdot \lambda(e) \\ &= \sum_{e:A\to v} \lambda(e)\end{aligned}$$

which is, as required, the sum of the labels on all edges joining A to v. Thus the algorithm works correctly for all vertices v with $\text{depth}(v) = 1$.

Now we assume that the assertion is true for all vertices at depth i or less, and consider a vertex v at depth $i+1$. When the algorithm computes $\mu(v)$ on line $*$, the value it assigns to it is

$$\mu(v) = \sum_{e:\text{fin}(e)=v} \mu(\text{init}(e)) \cdot \lambda(e). \tag{2.6}$$

But $\text{depth}(\text{init}(e)) = i$ and so by the induction hypothesis

$$\mu(\text{init}(e)) = \sum_{P:A\to\text{init}(e)} \lambda(P). \tag{2.7}$$

Combining (2.6) and (2.7), and using the commutativity of "+" and the distributive law (2.5), we have

$$\begin{aligned}\mu(v) &= \sum_{e:\text{fin}(e)=v} \sum_{P:A\to\text{init}(e)} \lambda(P)\cdot\lambda(e) \\ &= \sum_{e:\text{fin}(e)=v} \sum_{P:A\to\text{init}(e)} \lambda(Pe).\end{aligned} \tag{2.8}$$

But every path from A to v must be of the form Pe, where P is a path from A to a vertex u with $\text{depth}(u) = i$, $\text{init}(e) = u$, $\text{fin}(e) = v$. Thus by (2.8), $\mu(v)$ is correctly computed by the algorithm. □

The next theorem says that the computational complexity of the Viterbi algorithm is proportional to the number of edges in the trellis.

Theorem 2.10: The Viterbi algorithm requires $\Theta(|E|)$ arithmetic operations, i.e., "multiplications" and "additions."

Proof: The execution of line $*$ in the algorithm requires $\rho^-(v)$ "multiplications" and $\rho^-(v) - 1$ "additions," where $\rho^-(v)$ is defined in (2.2). Thus the total number of "multiplications" required by the algorithm is

$$\texttt{multiplications} = \sum_{i=1}^{n} \sum_{v\in V_i} \rho^-(v) \tag{2.9}$$

and the total number of additions is

$$\begin{aligned}\texttt{additions} &= \sum_{i=1}^{n} \sum_{v\in V_i} (\rho^-(v) - 1) \\ &= \sum_{i=1}^{n} \sum_{v\in V_i} \rho^-(v) - \sum_{i=1}^{n} \sum_{v\in V_i} 1.\end{aligned} \tag{2.10}$$

Now every edge in E is counted exactly once in the sum in (2.9), since if $e : u \to v$, then $\text{fin}(e) \in V_i$ for exactly one value of $i \in \{1, 2, \cdots, n\}$. Thus the sum in (2.9) is $|E|$. The second sum in (2.10) is $|V| - 1$, since every vertex except A is in $\bigcup_{i=1}^{n} V_i$. Thus from (2.9) and (2.10), we have

$$\texttt{multiplications} = |E| \tag{2.11}$$

$$\texttt{additions} = |E| - |V| + 1 \tag{2.12}$$

so that the total number of "arithmetic operations" required by the algorithm is $2|E| - |V| + 1 \le 2|E|$. We have $|V| \ge 1$, and $|E| - |V| + 1 \ge 0$ (since the trellis is connected), so that the total number of operations required is bounded above by $2|E|$ and bounded below by $|E|$. □

The quantity $|E| - |V| + 1$ appearing in (2.12) has a natural combinatorial significance: it represents the total number of "bifurcations" in the trellis. Here a "simple bifurcation" is a vertex v with $\rho^+(v) = 2$, and in general, a vertex v with $\rho^+(v) = \rho$ is counted as $\rho - 1$ bifurcations. With this definition, the total number of bifurcations in the trellis is given by the double sum in (2.10), which as we have seen is equal to $|E| - |V| + 1$. For example, the trellis in Fig. 1 has $|E| - |V| + 1 = 8 - 6 + 1 = 3$, and indeed that trellis has three bifurcations, one at vertex A and two at vertex 1.

In the next four examples, we will see how the Viterbi algorithm operates on the trellis of Fig. 1 when the labels come from the four types of semigroups described in Examples 2.4–2.7.

Example 2.11: Let us apply the Viterbi algorithm to the trellis of Fig. 1, using the semiring from Example 2.4, with the following set of Boolean labels:

e:	a	b	c	d	e	f	g	h
$\lambda(e)$:	1	0	1	1	1	1	0	1

Then if we follow the steps in Example 2.8, replacing "+" with OR, and "·" with AND, we find successively that $\mu(A) = 1$ (initialization), $\mu(1) = 1$, $\mu(2) = 0$, $\mu(3) = 1$, $\mu(4) = 1$, and finally, $\mu(B) = 1$. Thus the Viterbi algorithm concludes that $\mu(B) = 1$, which means (see Example 2.4) that there is at least one "active" path from A to B. Indeed, $P = aeh$ is such a path.

Example 2.12: Let us apply the Viterbi algorithm to the trellis of Fig. 1, using the semiring from Example 2.5, with the labels, to be interpreted as "edge lengths," as follows:

e:	a	b	c	d	e	f	g	h
$\lambda(e)$:	1	0	0	1	0	2	2	1

If we follow the steps in Example 2.8, making the appropriate changes, we find successively that $\mu(A) = 0$ (initialization), $\mu(1) = 1$, $\mu(2) = 0$, $\mu(3) = 1$, $\mu(4) = 1$, and finally, $\mu(B) = 2$. The Viterbi algorithm concludes that $\mu(B)$, i.e., the length of the shortest path from A to B, is 2. It is easy to verify by inspection that this path is aeh.

Example 2.13: This time let us use the semiring $\mathbb{Z}[x]$, i.e., the ring of polynomials in the indeterminate x with integer coefficients, as in Example 2.6, and let the labels in Fig. 1 be as follows:

e:	a	b	c	d	e	f	g	h
$\lambda(e)$:	x	1	1	x	1	x^2	x^2	x

Note that the labels in this case are all of the form x^l, where l is the edge length from Example 2.12. Once again, following the outline in Example 2.8, making the appropriate changes,

we find successively that $\mu(A) = 1$ (initialization), $\mu(1) = x$, $\mu(2) = 1$, $\mu(3) = x + x^2$, $\mu(4) = x + x^2$, and finally, $\mu(B) = x^2 + 2x^3 + x^4$. In this case $\mu(B) = x^2 + 2x^3 + x^4$ represents the generating function for the paths from A to B, enumerated by length. Thus there is one path of length 2, two paths of length 3, and one path of length 4.

Example 2.14: Finally, let us use the semiring of Example 2.7, using (with slight abuse of notation), the set of "letters" $\{a, b, c, d, e, f, g, h\}$ as labels for the corresponding edges. Then the computation in Example 2.8 shows that the set of strings "generated" by the trellis, i.e., the language produced by T, is $\{acg, adg, aeh, afh\}$.

We conclude this section with some remarks concerning the relationship between Viterbi's algorithm, and other, similar, algorithms that appear in the computer science literature. First, it is often asserted that the Viterbi algorithm is a "dynamic programming" solution to the problem of computing flows in a trellis (dynamic programming is discussed in [7, ch. 16]). This is true, but it should be borne in mind that dynamic programming is a *methodology*, not an algorithm, and there is no evidence that Viterbi was aware of this methodology in 1967 when he invented his algorithm [41]. Still, it is fair to say that a bright present-day computer science student, familiar with dynamic programming, and asked to produce an algorithm for finding flows in a trellis, would be likely to re-invent the Viterbi algorithm.

The closest match to an existing algorithm is usually considered to be *Dijkstra's algorithm* [1, sec. 5.10], [7, sec. 25.2], but there are some important differences. Dijkstra's algorithm finds the shortest paths from a given initial vertex v_0 to all other vertices in an arbitrary finite directed graph. However, Dijkstra's algorithm, when applied to a trellis (with the initial vertex being the source) is not as efficient as Viterbi's algorithm, since its running time is $O(|V|^2)$, not $O(|E|)$. (The problem is that Dijkstra's algorithm has not been "tailored" to the regular structure of the trellis.) Furthermore, as pointed out in [1, sec. 5.10], Dijkstra's algorithm does not lend itself to the "semiring" generalization. The semiring generalization is, however, available for an algorithm that computes the flows between all pairs of vertices in an arbitrary directed graph [1, sec. 5.6], [7, sec. 26.4], but the complexity of this algorithm is $O(|V|^3)$, and there does not appear to be any way to significantly simplify this algorithm, if only the flows from one particular vertex are required. Another close match is an algorithm which finds the single-source shortest paths in a *directed acyclic graph* (dag), as described in [7, sec. 25.4]. Its complexity is $\Theta(|V| + |E|)$, which is better than Dijkstra's algorithm, but still not as good as Viterbi's algorithm, since a trellis is a very special kind of dag, which obviates the "topological sort" which is necessary in the dag algorithm. Also, the dag algorithm does not appear to lend itself to the semiring generalization. The moral here is that Viterbi's algorithm is *an algorithm on a trellis*; nontrellis algorithms, when specialized to trellises, are not as efficient as Viterbi's algorithm. Conversely, it is not fair to say that Viterbi's algorithm applies to structures more general than trellises (such as dags or arbitrary digraphs), since highly efficient algorithms are already available for such problems.

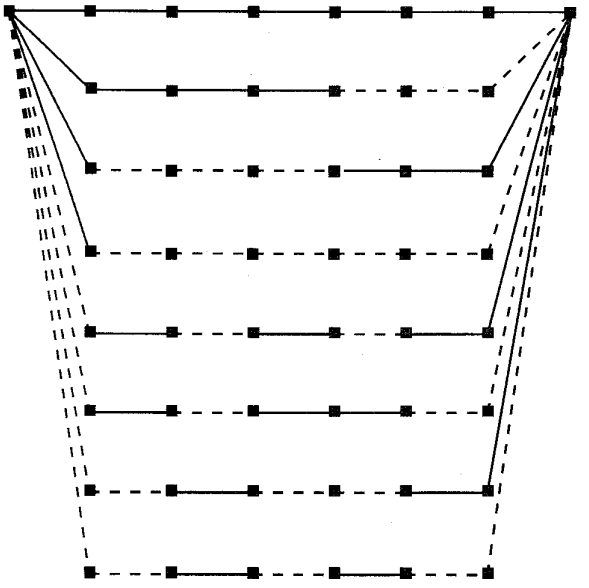

Fig. 2. A trellis representing the code in Example 3.1, the SOI trellis [37]. The edge count is $|E| = 56$; also, $|V| = 50$, and $|E| - |V| + 1 = 7$. (In the notation of Section VII, this is the $\{[1, 7], [1, 7], [1, 7]\}$ trellis for $\mathbb{C}$ with respect to the MSGM given in Example 7.1.

III. The BCJR Trellis for a Linear Block Code—Definition

In Section II, we discussed general labeled trellises and the general Viterbi algorithm. In this section, we will apply those results to the problem of finding "good" trellis representations for binary linear block codes.

Thus let $\mathbb{C}$ be a fixed (n, k, d) binary linear block code, and let $T = (V, E, \lambda)$ be a labeled trellis of rank n, with labels from the set $S = \{0, 1\}$, with the structure of the "language semiring" of Example 2.7. We say T *represents* $\mathbb{C}$ if the language produced by T is identical to the code $\mathbb{C}$. In other words, if we associate a length-n binary word with every path from A to B in the trellis by concatenating the edge labels on the path, and if the set of such "trellis path" words is identical to the set of codewords in $\mathbb{C}$, we say that T represents $\mathbb{C}$.

Example 3.1: Consider the $(7, 3, 3)$ block code defined by the generator matrix

$$G = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \end{pmatrix}. \tag{3.1}$$

This code has eight codewords of length 7. It can be represented by many different trellises, and in Figs. 2–5 we see four such trellises. (For convenience, in these figures, a solid edge is to be considered labeled 0, and a dashed edge, labeled 1.) In Section VII, we will reveal how we found these four trellises, but for now the reader can verify directly that in each case, the eight labeled paths from the source to the sink correspond to the eight codewords in C.

If the code $\mathbb{C}$ is being used on a discrete memoryless channel with transition probabilities $p(y \mid x)$, where $x \in \{0, 1\}$, and y is an element of the channel output alphabet, then

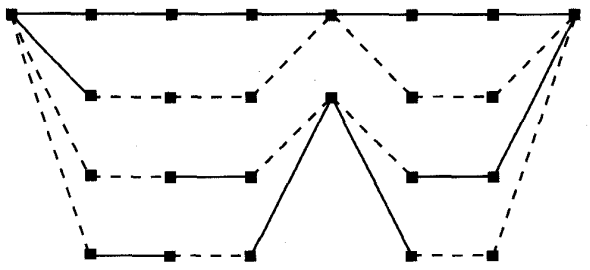

Fig. 3. Another trellis for the code of Example 3.1. The edge count is $|E| = 28$; also, $|V| = 24$, and $|E| - |V| + 1 = 5$. (This is the $\{[1,7],[1,4],[5,7]\}$ trellis for $\mathbb{C}$ with respect to the MSGM given in Example 7.1.)

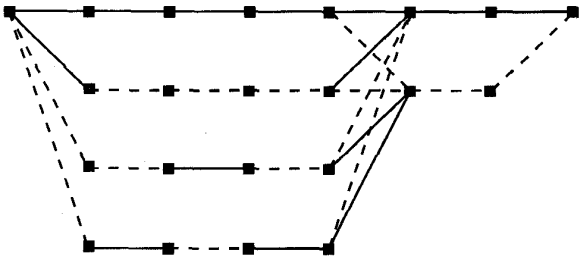

Fig. 4. Yet another one. The edge count is $|E| = 28$; also, $|V| = 22$, and $|E| - |V| + 1 = 7$. (This is the $\{[1,5],[1,5],[5,7]\}$ trellis for $\mathbb{C}$ with respect to the MSGM given in Example 7.1.)

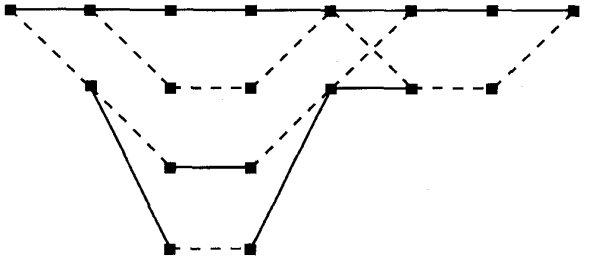

Fig. 5. Still another one. The edge count is $|E| = 22$; also, $|V| = 18$, and $|E| - |V| + 1 = 5$. (This is the $\{[1,5],[2,4],[5,7]\}$ trellis for $\mathbb{C}$ with respect to the MSGM given in Example 7.1.)

any trellis representing $\mathbb{C}$ can be used for Viterbi decoding, using the semiring of Example 2.5. It works like this. If $\boldsymbol{R} = (R_1, R_2, \cdots, R_n)$ is a received noisy version of one of the codewords, and if each edge $e \in E_{i-1,i}$ is re-labeled with the "log-likelihood" quantity $-\log p(R_i \mid \lambda(e))$, then the codeword corresponding to the "shortest path" from A to B in the trellis will be the maximum-likelihood choice for the transmitted codeword. See [43, sec. III], for a more detailed description of this.

Similarly, we can use a trellis representing $\mathbb{C}$ to calculate the weight enumerator for $\mathbb{C}$, using the semiring of Example 2.6. For this application, each edge e in the trellis should be re-labeled $x^{\lambda(e)}$. Then the total "flow" from A to B will be the generating function for the weights of the codewords, i.e., the code's weight enumerator, as explained in Example 2.13.

In either application (Viterbi decoding or weight enumerator calculation), because of Theorem 2.10, we will wish to find, among all trellises that represent $\mathbb{C}$, the one or ones with the *fewest edges*. Surprisingly, it turns out that there is always (up to isomorphism) a unique edge-minimal trellis that represents $\mathbb{C}$. This trellis structure was first discovered by Bahl, Cocke, Jelinek, and Raviv in 1974 [3], and Wolf in 1978 [43], but later isomorphic versions of it were discovered and analyzed by Massey [29], Forney [11], and Muder [35]. We now review this important construction. We will begin with what we shall call the "BCJR ur-trellis."

The BCJR ur-trellis is based on an $r \times n$ parity-check matrix H for $\mathbb{C}$, where $r = n - k$ is the redundancy of the code. We will assume that

$$H = (\boldsymbol{h}_1, \cdots, \boldsymbol{h}_n)$$

where $\boldsymbol{h}_1 \cdots, \boldsymbol{h}_n$ are the n columns of H. The code $\mathbb{C}$ then consists of all vectors $\boldsymbol{C} = (C_1, \cdots, C_n)$ such that

$$H\boldsymbol{C}^T = C_1\boldsymbol{h}_1 + \cdots + C_n\boldsymbol{h}_n = 0. \tag{3.2}$$

The vertex set for the BCJR ur-trellis consists of 2^r vertices at depth i for $i = 0, 1, \cdots, n$. For convenience we will assume that each of the vertices at depth i is identified with a binary vector of length r, which is called the *state* of the vertex. Thus there are $2^r \times (n+1)$ vertices, each identified uniquely by a (state, depth) pair.

The *edges* of the BCJR ur-trellis are produced by the codewords of $\mathbb{C}$. If $\boldsymbol{C} = (C_1, \cdots, C_n)$ is a codeword, there are n corresponding labeled edges in the trellis, $e_1, \cdots, e_n$, which form a path of length n, defined as follows:

$$\begin{aligned} \text{init}\,(e_i) &= C_1\boldsymbol{h}_1 + \cdots + C_{i-1}\boldsymbol{h}_{i-1} \\ \text{fin}\,(e_i) &= C_1\boldsymbol{h}_1 + \cdots + C_{i-1}\boldsymbol{h}_{i-1} + C_i\boldsymbol{h}_i \\ \lambda(e_i) &= C_i \end{aligned} \tag{3.3}$$

for $i = 1, 2, \cdots, n$. In (3.3), when $i = 1$, $\text{init}\,(e_1)$ is defined to be 0. Thus

$$\text{init}\,(e) = 0, \qquad \text{for all } e \in E_{0,1}. \tag{3.4}$$

Every code path $e_1 \cdots e_n$ ends at state 0, i.e., has $\text{fin}\,(e_n) = 0$, since from (3.3) and (3.2), with $i = n$ we have

$$\text{fin}\,(e_n) = C_1\boldsymbol{h}_1 + \cdots + C_n\boldsymbol{h}_n = 0.$$

Thus

$$\text{fin}\,(e) = 0, \qquad \text{for all } e \in E_{n-1,n}. \tag{3.5}$$

It can happen that different codewords will produce common edges, i.e., edges with the same values of $\text{init}\,(e)$, $\text{fin}\,(e)$, and $\lambda(e)$. Such "shared" edges are only counted once in the trellis. It is this sharing of edges that makes the BCJR trellis an efficient graphical representation of the code.

Example 3.2: To illustrate the BCJR ur-trellis construction, consider again the $(7, 3, 3)$ code with generator matrix given by (3.1). One possible parity-check matrix for this code is

$$H = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}. \tag{3.6}$$

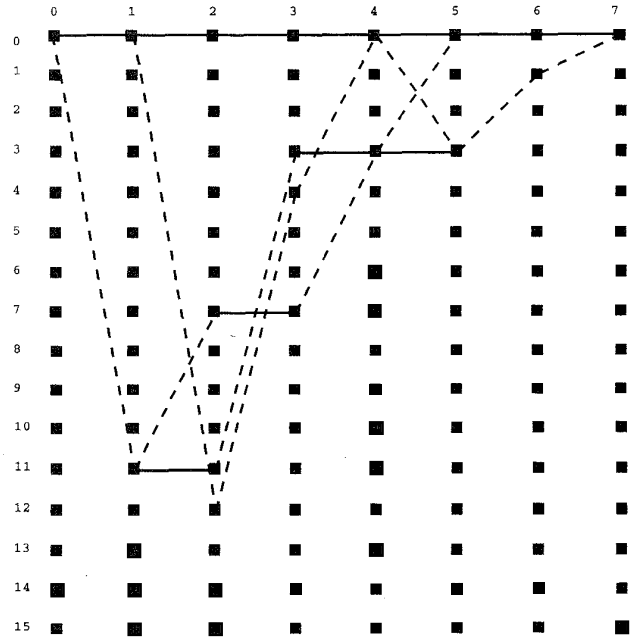

Fig. 6. The BCJR ur-trellis for the $(7, 3, 3)$ code with generator matrix given by (3.1) and parity-check matrix given by (3.6).

In Fig. 6, we see the corresponding BCJR ur-trellis. (For convenience, in Fig. 6 the vertices are labeled with the decimal equivalents of their binary representation.) For example, consider the codeword corresponding to the first row of G, i.e., $\boldsymbol{C} = (1010011)$. According to (3.3), the first edge in the corresponding trellis path is defined by $\text{init}(e_1) = 0$, $\text{fin}(e_1) = C_1 \cdot \boldsymbol{h}_1 = 1 \cdot (1011) = 11$, and $\lambda(e_1) = C_1 = 1$. In Fig. 6, we indicate this by joining state 0 at depth 0 to state 11 at depth 1 with a dashed edge. Similarly, the second edge e_2 for the codeword (1010011) has $\text{init}(e_2) = C_1 \cdot \boldsymbol{h}_1 = 1 \cdot (1011) = 11$, $\text{fin}(e_2) = C_1 \cdot \boldsymbol{h}_1 + C_2 \cdot \boldsymbol{h}_2 = (1011) = 11$, and $\lambda(e_2) = C_2 = 0$. This is indicated in Fig. 6 by a solid edge connecting state 11 at depth 1 to state 11 at depth 3.

If we continue in this way, calculating $\text{init}(e)$, $\text{fin}(e)$, and $\lambda(e)$, for all eight edges on all eight codewords, we arrive at the trellis of Fig. 6. This trellis has only 22 distinct edges, rather than the expected $8 \times 8 = 64$, because many of the edges are shared between several codewords. For example, all codewords with $C_1 = 1$ share the edge connecting the state 0 at depth 0 to state 11 at depth 1.

The ur-trellis of Fig. 6 has many "unused" vertices, i.e., vertices through which no edge passes, and so, according to the definition given in Section II, it is technically not a trellis at all. However, if we delete the unused vertices, and reorder the remaining ones appropriately, we arrive at the "true" BCJR trellis shown in Fig. 7, which is identical to the trellis of Fig. 5.

We now define the *BCJR trellis* as the BCJR ur-trellis from which the unused vertices have been deleted, and conclude this section with a proof that the BCJR trellis represents the code $\mathbb{C}$. (Neither Bahl *et al.* nor Wolf gave a proof of this fact.)

Theorem 3.3: The BCJR trellis, as defined by (3.3), represents the code $\mathbb{C}$ defined by the parity-check matrix H, i.e.,

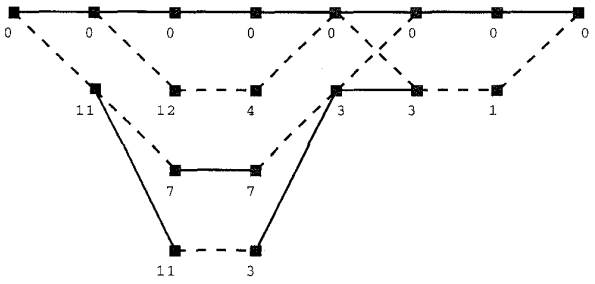

Fig. 7. The BCJR trellis obtained from the BCJR ur-trellis of Fig. 6 by deleting the unused vertices. It is identical to the trellis in Fig. 5.

the trellis paths from the source to the sink are in one-to-one correspondence with the codewords of $\mathbb{C}$.

Proof: By the construction (3.3), every codeword in $\mathbb{C}$ corresponds to a path of length n in the BCJR trellis. What we have to prove is that, conversely, every such path produces a valid codeword, and that no two paths produce the same codeword. To do this we first note that from (3.3), for every edge $e \in E_{i-1,i}$ in the BCJR trellis

$$\text{fin}(e) = \text{init}(e) + \lambda(e) \cdot \boldsymbol{h}_i. \tag{3.7}$$

Equation (3.7), together with (3.4), implies, via an easy induction argument, that for every path $e_1 \cdots e_i \in E_{0,i}$

$$\text{fin}(e_i) = \lambda(e_1)\boldsymbol{h}_1 + \cdots + \lambda(e_i)\boldsymbol{h}_i. \tag{3.8}$$

Equation (3.8) shows that the label sequence $\lambda(e_1), \cdots, \lambda(e_n)$ uniquely determines the trellis path, so that no word is produced more than once by the trellis. It therefore remains only to show that every source–sink path in the trellis produces a codeword.

With $i = n$, (3.8) says that for every trellis path $e_1 \cdots e_n$ of length n

$$\lambda(e_1) \cdot \boldsymbol{h}_1 + \cdots + \lambda(e_n) \cdot \boldsymbol{h}_n = \text{fin}(e_n).$$

But by (3.5), $\text{fin}(e_n) = 0$, so that

$$\lambda(e_1) \cdot \boldsymbol{h}_1 + \cdots + \lambda(e_n) \cdot \boldsymbol{h}_n = 0$$

which implies (see (3.2), that $(\lambda(e_1), \cdots, \lambda(e_n))$ is a codeword. Thus every path of length n in the BCJR trellis corresponds to a codeword. □

IV. The BCJR Trellis for a Linear Block Code—Analysis

In this section we will give the basic algebraic and combinatorial analysis of the BCJR trellis, culminating with Theorem 4.6, which gives a formula for the number of vertices and edges at each depth. Our starting point is the definition of the BCJR trellis as given in Section III, and for the sake of self-containedness, we shall deliberately ignore the fact that the BCJR trellis is now known to be isomorphic to the Forney–Muder "minimal" trellis. However, many of the results we shall derive for the BCJR trellis are already known for the minimal trellis, and we shall attempt to give credit where credit is due.

The key to the algebraic analysis of the BCJR trellis is the fact that for each index i, the sets V_i and $E_{i-1,i}$ can be viewed as vector spaces over GF(2), an observation first made by Forney [11]. This can be seen as follows. In the construction of the BCJR trellis, every codeword $\boldsymbol{C}$ produces a path of length n, with edges $e_1, e_2, \cdots, e_n$, according to the formula given in (3.3). Since e_i is an edge in $E_{i-1,i}$, i.e., it connects a vertex at depth $i-1$ to one at depth i, the only vertex at depth i that this sequence of edges passes through is

$$\text{init}\,(e_{i+1}) = \text{fin}\,(e_i) = C_1\boldsymbol{h}_1 + \cdots + C_i\boldsymbol{h}_i.$$

Thus V_i, the set of vertices in the BCJR trellis at depth i, is the image of the code $\mathbb{C}$ under the linear mapping $\sigma_i\colon \mathbb{C} \to V_i$ given by

$$\sigma_i(\boldsymbol{C}) = C_1\boldsymbol{h}_1 + \cdots + C_i\boldsymbol{h}_i. \tag{4.1}$$

Similarly, according to (3.3), a codeword $\boldsymbol{C}$ produces a unique edge e_i in $E_{i-1,i}$ which can be described by the triple $(\text{init}\,(e_i), \text{fin}\,(e_i), \lambda(e_i))$, which, according to (3.3), is

$$\tau_i(\boldsymbol{C}) = (\sigma_{i-1}(\boldsymbol{C}), \sigma_i(\boldsymbol{C}), C_i) \tag{4.2}$$

where σ_i is the mapping defined in (4.1). Thus $E_{i-1,i}$ is the image of $\mathbb{C}$, under the linear mapping τ_i defined in (4.2).

Definition 4.1: In what follows, we will denote the dimensions of the vertex spaces V_i and the edge spaces $E_{i-1,i}$ by s_i and b_i, respectively

$$s_i = \dim V_i, \qquad \text{for } i = 0, \cdots, n \tag{4.3}$$

$$b_i = \dim E_{i-1,i}, \quad \text{for } i = 1, \cdots, n. \tag{4.4}$$

Our first theorem about the BCJR trellis gives a useful characterization of the vertex space V_i, in terms an arbitrary pair (G, H) of generator and parity-check matrices for $\mathbb{C}$. (The "state-space theorem" of Forney and Trott [13] can be viewed as a generalization of this theorem.)

Theorem 4.2: Suppose $i \in \{0, 1, \cdots, n\}$, and denote by G_i and H_i the matrices consisting of the first i columns of G and H, respectively, and by $\bar{G}_{n-i}$ and $\bar{H}_{n-i}$ the matrices consisting of the last $n-i$ columns of G and H, respectively. Then

$$V_i = \text{row space}\, G_iH_i^T = \text{row space}\, \bar{G}_{n-i}\bar{H}_{n-i}^T \tag{4.5}$$

and hence

$$s_i = \text{rank}\, G_iH_i^T = \text{rank}\, \bar{G}_{n-i}\bar{H}_{n-i}^T. \tag{4.6}$$

Proof: As we have seen, V_i is the image of $\mathbb{C}$ under the mapping σ_i defined in (4.1). It follows that V_i is the set of r-dimensional vectors of the form $\{C_1\boldsymbol{h}_1 + \cdots + C_i\boldsymbol{h}_i\}$, where $(C_1, \cdots, C_n)$ is a codeword. But since every codeword is of the form uG, where u is a $1 \times k$ binary vector, we have

$$\begin{aligned} C_1\boldsymbol{h}_1 + \cdots + C_i\boldsymbol{h}_i &= (C_1, \cdots, C_i)H_i^T \\ &= uG_iH_i^T \end{aligned}$$

which implies the first part of (3.2) and (4.6). Similarly, by (3.2) and (4.1), we have

$$\begin{aligned} \sigma_i(\boldsymbol{C}) &= C_{i+1}\boldsymbol{h}_{i+1} + \cdots + C_n\boldsymbol{h}_n \\ &= (C_{i+1}, \cdots, C_n)\bar{H}_{n-i}^T \\ &= u\bar{G}_{n-i}\bar{H}_{n-i}^T \end{aligned}$$

which implies the second part of (4.5) and (4.6). □

Corollary 4.3 (Wolf [43], Massey [29]): The vertex dimensions s_i satisfy the following bounds:

$$s_i \le \min(i, n-i, k, r), \quad \text{for } i = 0, 1 \cdots, n.$$

Proof: First note that the matrices G_i, H_i, $\bar{G}_{n-i}$, and $\bar{H}_{n-i}$ have sizes $k \times i$, $r \times i$, $k \times n-i$, and $r \times n-i$, respectively. The result stated now follows immediately from (4.6), and the following two well-known rank inequalities: if A is an $m \times n$ matrix, then $\text{rank}\, A \le \min(m, n)$, and $\text{rank}\, AB \le \min(\text{rank}\, A, \text{rank}\, B)$ [17, sec. 0.4.5]. □

It is a remarkable fact that the parameters s_i for the dual code for $\mathbb{C}$ are the same as for those $\mathbb{C}$ itself (although the b_i's are not).

Corollary 4.4 (Forney [11]): If $\mathbb{C}^\perp$ is the dual code for $\mathbb{C}$, and if the vertex dimensions of the dual code are denoted by $s_i^\perp$, then

$$s_i^\perp = s_i, \qquad \text{for } i = 0, 1, \cdots, n.$$

Proof: For the dual code $\mathbb{C}^\perp$, the roles of the generator matrix and parity-check matrix are reversed, so that $V_i^\perp$ is the row space of the matrix $H_iG_i^T$, and so by (4.5)

$$\begin{aligned} s_i^\perp &= \text{rank}\, H_iG_i^T \\ &= \text{rank}\, G_iH_i^T \end{aligned}$$

by the "row rank = column rank" theorem of linear algebra [15, Theorem 3.22], [17, sec. 0.4.1]. Thus by (4.6), $s_i = s_i^\perp$. □

Theorem 4.2 gives us a useful computational characterization of the vertex dimension s_i, but it does not give much algebraic insight. To make a deeper analysis of the BCJR trellis, we need to define an important set of subcodes of $\mathbb{C}$, called the *past* and *future* subcodes, which were introduced by Forney [11, Appendix A]. For $i = 0, 1, \cdots, n-1$, we define the ith past subcode of $\mathbb{C}$, denoted P_i, as follows:

$$P_i = \{\boldsymbol{C} \in \mathbb{C}\colon C_{i+1} = C_{i+2} = \cdots = C_n = 0\}. \tag{4.7}$$

Similarly, for $i = 1, \cdots, n$, the ith future subcode of $\mathbb{C}$, denoted F_i, is defined as follows:

$$F_i = \{\boldsymbol{C} \in \mathbb{C}\colon C_1 = C_2 = \cdots = C_i = 0\}. \tag{4.8}$$

If we think of i as a "time" index, then P_i consists of all codewords whose nonzero components are in the "past," and F_i consists of all codewords whose nonzero components are in the "future," relative to the current time. The subcodes P_i and F_i are clearly linear, and for future reference, we denote their dimensions by p_i and f_i, respectively

$$p_i = \dim P_i, \quad i = 0, \cdots, n-1 \tag{4.9}$$

$$f_i = \dim F_i, \quad i = 1, \cdots, n. \tag{4.10}$$

By elaborating on the proof of Corollary 4.4, it is possible to show that if $p_i^\perp$ and $f_i^\perp$ are the dimensions of the past and future subcodes of the dual code, then

$$\begin{aligned} p_i^\perp &= f_i + i - k \\ f_i^\perp &= p_i - i + (n-k). \end{aligned}$$

This result can also be found in [13] or [12].

We now define another, similar, family of codes derived from $\mathbb{C}$. Let us denote by P^i (for $i = 1, \cdots, n$) and F^i (for $i = 0, \cdots, n-1$) the ith *past projection* and *future projection* of $\mathbb{C}$, defined as follows:

$$P^i = \{(x_1, \cdots, x_i) \colon \boldsymbol{x} \in \mathbb{C}\} \tag{4.11}$$

$$F^i = \{(x_{i+1}, \cdots, x_n) \colon \boldsymbol{x} \in \mathbb{C}\} \tag{4.12}$$

and by p^i and f^i the corresponding dimensions, i.e.

$$p^i = \dim P^i, \quad i = 1, \cdots, n \tag{4.13}$$

$$f^i = \dim F^i, \quad i = 0, \cdots, n-1. \tag{4.14}$$

Occasionally, we will refer to P_n, F_0, P^0, and F^n, which have not been defined. By convention we take

$$\begin{aligned} P_n &= \mathbb{C}, \quad & p_n &= k \\ F_0 &= \mathbb{C}, \quad & f_0 &= k \\ P^0 &= (0), \quad & p_0 &= 0 \\ F^n &= (0), \quad & f_n &= 0. \end{aligned} \tag{4.15}$$

The past and future projections were also introduced by Forney and Trott [13]; see also [12].

The past and future subcodes and projections are closely related. Indeed, if π_i denotes the ith past projection mapping, i.e., if $\boldsymbol{C} = (C_1, \cdots, C_n)$ is a codeword, and if $\pi \colon \mathbb{C} \to P^i$ is defined by

$$\pi_i(\boldsymbol{C}) = (C_1, \cdots, C_i)$$

then the kernel of π_i, i.e., the set of codewords $\boldsymbol{C}$ such that $\pi_i(\boldsymbol{C})$ is zero, is the future subcode F_i (see (4.8)), and so by the well-known "rank + nullity = dimension" theorem of linear algebra ([2, Theorem 2.3], [15, Theorem 3.3]), it follows that

$$k = p^i + f_i, \qquad \text{for } i = 0, \cdots, n. \tag{4.16}$$

Similarly, if we define the ith future projection mapping $\phi_i \colon \mathbb{C} \to F^i$ by

$$\phi_i(\boldsymbol{C}) = (C_{i+1}, \cdots, C_n)$$

then the kernel of ϕ_i is P_i, so that

$$k = p_i + f^i, \quad \text{for } i = 0, \cdots, n. \tag{4.17}$$

Our next result identifies the kernels of the vertex and edge mappings σ_i and τ_i defined in (4.1) and (4.2), in terms of the past and future subcodes P_i and F_i.

Theorem 4.5 (Forney [11]): The kernel of σ_i is

$$\ker(\sigma_i) = P_i \oplus F_i, \qquad \text{for } i = 0, 1, \cdots, n \tag{4.18}$$

and the kernel of τ_i is

$$\ker(\tau_i) = P_{i-1} \oplus F_i, \qquad \text{for } i = 1, 2, \cdots, n. \tag{4.19}$$

Proof: Suppose that $\boldsymbol{C} = (C_1, \cdots, C_n) \in P_i \oplus F_i$. Then we have

$$\boldsymbol{C} = \boldsymbol{C}_1 + \boldsymbol{C}_2$$

where $\boldsymbol{C}_1 \in P_i$, and $\boldsymbol{C}_2 \in F_i$, i.e.

$$\boldsymbol{C}_1 = (C_1, \cdots, C_i, 0, \cdots, 0) \in \mathbb{C} \tag{4.20}$$

$$\boldsymbol{C}_2 = (0, \cdots, 0, C_{i+1}, \cdots, C_n) \in \mathbb{C}. \tag{4.21}$$

But it follows from (4.20) and (3.2), that $H\boldsymbol{C}_1^T = 0$, which means (see (4.1)), that $\boldsymbol{C} \in \ker(\sigma_i)$. Thus $P_i \oplus F_i \subseteq \ker(\sigma_i)$.

To prove the opposite inequality, i.e., $\ker(\sigma_i) \subseteq P_i \oplus F_i$, we suppose $\boldsymbol{C} \in \ker(\sigma_i)$, i.e., $\sigma_i(\boldsymbol{C}) = 0$. Then (4.20) holds. Since, however, $\boldsymbol{C} \in \mathbb{C}$, then (3.2) also holds. But if we add these two equations, we find that $C_{i+1}\boldsymbol{h}_{i+1} + \cdots + C_n\boldsymbol{h}_n = 0$, i.e., (4.21) holds as well. Thus $\boldsymbol{C} \in P_i \oplus F_i$. This shows that $\ker(\sigma_i) \subseteq P_i \oplus F_i$, and completes the proof of (4.18).

To prove (4.19), we note that from (4.2), we have

$$\ker(\tau_i) = (\ker \sigma_{i-1}) \cap (\ker \sigma_i) \cap \{\boldsymbol{C} \colon C_i = 0\}.$$

But from (4.18), (4.19) now easily follows. □

We conclude this section with a theorem which counts, in detail, the number of vertices and edges in the BCJR trellis.

Theorem (Forney [11], Muder [35]): The number of vertices at depth i in the BCJR trellis is

$$|V_i| = 2^{k - p_i - f_i} \tag{4.22}$$

for $i = 0, 1, \cdots, n$. Similarly, the number of edges connecting vertices at depth $i-1$ to those at depth i is

$$|E_{i-1,i}| = 2^{k - p_{i-1} - f_i} \tag{4.23}$$

for $i = 1, \cdots, n$. Finally, all $v \in V_i$ have common out- and in-degrees, denoted by ρ_i^+ and ρ_i^-, where

$$\rho_i^+ = 2^{f_i - f_{i+1}}, \qquad \text{for } i = 0, 1, \cdots, n-1. \tag{4.24}$$

$$\rho_i^- = 2^{p_i - p_{i-1}}, \qquad \text{for } i = 1, 2, \cdots, n. \tag{4.25}$$

Proof: According to (4.1), the vertex space V_i is the image, under the mapping σ_i, of the code $\mathbb{C}$. Thus again according to the "dimension = rank + nullity" theorem, we have $\dim \mathbb{C} = \dim V_i + \dim \ker \sigma_i$. But $\dim \mathbb{C} = k$, and by (4.18), $\dim \ker \sigma_i = p_i + f_i$. This proves (4.22). Similarly, by (4.2), the edge space $E_{i-1,i}$ is the image, under the mapping τ_i, of $\mathbb{C}$. Thus $\dim \mathbb{C} = \dim E_{i-1,i} + \dim \ker \tau_i$. But by (4.19), $\dim \ker \tau_i = p_{i-1} + f_i$, which proves (4.23).

It remains to prove (4.24) and (4.25). If $v \in V_i$, let us denote the set of edges $e \in E_{i,i+1}$ for which $\text{init}(e) = v$ by $E_{i,i+1}^v$. Then $\rho^+(v) = |E_{i,i+1}^v|$. If we regard the set $E_{i,i+1}^0$ as a subspace of $E_{i,i+1}$, it follows that each set $E_{i,i+1}^v$ is a coset of this subspace, and so each of the sets $E_{i,i+1}^v$ has the same size. But since there are $|E_{i,i+1}|$ edges originating from the $|V_i|$ vertices at depth i, it thus follows that the common out-degree of each $v \in V_i$ is $|E_{i,i+1}|/|V_i|$, which, by (4.22), and (4.23), is $2^{f_i - f_{i+1}}$. This proves (4.24). The proof of (4.25) is similar and is omitted. □

Example 4.7: In Section VI, we will find efficient ways to compute the p_i's and f_i's directly from a "minimal span" generator matrix for $\mathbb{C}$. In this example, we will indicate how the past and future subcodes can be found by "inspecting" the BCJR trellis. Thus consider the BCJR trellis for the $(7,3,3)$ code of Examples 3.1 and 3.2, as shown in Figs. 6 and 7. By definition ((4.7)), the subcode P_i consists of all codewords which become, and remain, zero from coordinate $i+1$ onwards. What this means geometrically is that the corresponding trellis path, which must begin in state 0, must have returned to state 0 at depth i, and then continue in state zero thereafter. Since we can see by inspecting the trellis that no nonzero code path returns to state 0 until $i = 4$, it follows that

$$P_0 = P_1 = P_2 = P_3 = \{0000000\}$$
$$p_0 = p_1 = p_2 = p_3 = 0.$$

For $i = 4$, we see that, besides the all-zero path, there is one other path which has returned to state 0 at depth 4, viz., the path $0 \to 0 \to 12 \to 4 \to 0 \to 0 \to 0 \to 0$, which corresponds to the codeword 0111000. Thus

$$P_4 = \{0000000, 0111000\}$$
$$p_4 = 1.$$

Continuing in this way, we find that

$$P_5 = P_6 = \{0000000, 0111000, 1101100, 1010100\}$$
$$p_5 = p_6 = 2$$
$$P_7 = \mathbb{C}$$
$$p_7 = 3.$$

The future subcode F_i is the set of codewords whose trellis paths diverge from state 0 at depth i or later. Thus by default (or else by (4.15)), we have

$$F_0 = \mathbb{C}$$
$$f_0 = 3.$$

By inspecting the trellis we see that there are four codepaths which diverge from state 0 at depth 1 or later, viz.,

$$0 \to 0 \to\ 0 \to 0 \to 0 \to 0 \to 0 \to 0$$
$$0 \to 0 \to\ 0 \to 0 \to 0 \to 3 \to 1 \to 0$$
$$0 \to 0 \to 12 \to 4 \to 0 \to 0 \to 0 \to 0$$
$$0 \to 0 \to 12 \to 4 \to 0 \to 3 \to 1 \to 0.$$

Thus F_1 is the set of codewords corresponding to these code paths, viz.

$$F_1 = \{0000000, 0000111, 0111000, 0111111\}$$
$$f_1 = 2.$$

Similarly, we obtain

$$F_2 = F_3 = F_4 = \{0000000, 0000111\}$$
$$f_2 = f_3 = f_4 = 1$$
$$F_6 = F_7 = \{0000000\}$$
$$f_6 = f_7 = 0.$$

Having now found the P_i's and the F_i's, we apply Theorems 4.6, 4.1, as well as (4.16) and (4.17), and obtain the following table:

i	p_i	f_i	p^i	f^i	s_i	b_i
0	0	3	0	3	0	–
1	0	2	1	3	1	1
2	0	1	2	3	2	2
3	0	1	2	3	2	2
4	1	1	2	2	1	2
5	2	0	3	1	1	2
6	2	0	3	1	1	1
7	3	0	3	0	0	1

We will do this same calculation another way in Example 6.20, below.

We conclude this section with an information-theoretic interpretation of the vertex dimension s_i defined in (4.3). We assume that the reader is familiar with the notions of the *entropy* $H(\boldsymbol{X})$ of a random variable or vector, and the *mutual information* $I(\boldsymbol{X};\boldsymbol{Y})$ between a pair of random variables or vectors (see, e.g., [30, ch. 1]).

Theorem 4.8: For a given code $\mathbb{C}$, make $\mathbb{C}$ into a uniform probability space, by assigning each codeword a probability of 2^{-k}. Let $(X_1, \cdots, X_n)$ be a random codeword from this space. Then, for each $i = 0, 1, \cdots, n$, we have

$$I(X_1, \cdots, X_i; X_{i+1}, \cdots, X_n) = s_i.$$

Proof: For convenience, we denote $(X_1, \cdots, X_i)$ by $\boldsymbol{X}^L$, and $(X_{i+1}, \cdots, X_n)$ by $\boldsymbol{X}^R$. Then, by the $I(X;Y) = H(X) + H(Y) - H(X,Y)$ formula [30, eq. (1.10)], we have

$$I(\boldsymbol{X}^L; \boldsymbol{X}^R) = H(\boldsymbol{X}^L) + H(\boldsymbol{X}^R) - H(\boldsymbol{X}^L, \boldsymbol{X}^R).$$

But $H(\boldsymbol{X}^L) = p^i$, by the definitions (4.11) and (4.13). Similarly, by (4.12) and (4.14), we have $H(\boldsymbol{X}^R) = f^i$. Thus since

$$H(\boldsymbol{X}^L, \boldsymbol{X}^R) = H(X_1, \cdots, X_n) = k$$

we have

$$\begin{aligned} I(\boldsymbol{X}^L; \boldsymbol{X}^R) &= p^i + f^i - k \\ &= (k - f_i) + (k - p_i) - k \quad \text{by (4.16) and (4.17)} \\ &= k - p_i - f_i = s_i \qquad \text{by (4.3) and (4.22).} \end{aligned}$$

□

V. The Optimality of the BCJR Trellis

In Section III, we described the BCJR trellis in detail, and in Section IV, we counted the number of vertices and edges at each depth in the BCJR trellis. In this section, we will show that among all trellises that represent a given linear block code, the BCJR trellis has both the fewest vertices, and the fewest edges, and that up to isomorphism, it is unique in these attributes. The following theorem gives the precise statement.

Theorem 5.1: Let $T = (V, E, \lambda)$ be any trellis that represents the linear block code $\mathbb{C}$. Then

$$|V_i| \geq 2^{k-p_i-f_i}, \qquad \text{for } i = 0, \cdots, n \tag{5.1}$$

$$|E_{i-1,i}| \geq 2^{k-p_{i-1}-f_i}, \qquad \text{for } i = 1, \cdots, n \tag{5.2}$$

where p_i and f_i are the dimensions of $\mathbb{C}$'s past and future subcodes, as defined in (4.9) and (4.10). Furthermore, if either (5.1) or (5.2) holds with equality for all indices i, then T is isomorphic to the BCJR trellis.

Remark: In his important 1988 paper, Muder [35], building on the work of Forney [11] in the same year, proved inequality (5.1), and furthermore showed that any trellis for which (5.1) holds for all i must be the "minimal" trellis for the code. Thus since the BCJR trellis is the minimal trellis, half of Theorem 5.1 is already known. However, again for the sake of self-containedness, and because the proof of the full theorem is almost as short as the half dealing with the edges, we include a proof of both halves here.

Proof: We begin by proving the inequalities (5.1) and (5.2). Then we will show that the BCJR trellis is the only trellis that meets all of these bounds simultaneously.

Suppose that $T = (V, E, \lambda)$ is a labeled trellis representing $\mathbb{C}$. For each vertex $v \in V$, define $\mathbb{C}(T, v)$ to be the set of codewords in $\mathbb{C}$ such that the corresponding T-trellis path passes through v. Since every trellis path must pass through exactly one vertex at depth i, we have

$$\mathbb{C} = \bigcup_{v \in V_i} \mathbb{C}(T, v), \qquad \text{for } i = 0, 1, \cdots, n. \tag{5.3}$$

The following lemma gives useful information about the sets $\mathbb{C}(T, v)$.

Lemma 5.2: If $v \in V_i$, then $\mathbb{C}(T, v)$ is a subset of one of the cosets of $P_i \oplus F_i$ in $\mathbb{C}$. Thus since each such coset contains $2^{p_i+f_i}$ elements, we have the upper bound

$$|\mathbb{C}(T, v)| \leq 2^{p_i+f_i}. \tag{5.4}$$

Proof: With the index i fixed, for $\boldsymbol{C} = (C_1, \cdots, C_n)$, define $\boldsymbol{C}^L$ (the left part of $\boldsymbol{C}$), and $\boldsymbol{C}^R$ (the right part of $\boldsymbol{C}$) as follows:

$$\boldsymbol{C}^L = (C_1, \cdots, C_i)$$
$$\boldsymbol{C}^R = (C_{i+1}, \cdots, C_n).$$

Similarly, define $\mathbb{C}^L(T, v)$ and $\mathbb{C}^R(T, v)$ as the left and right parts of the codewords in $\mathbb{C}(T, v)$

$$\mathbb{C}^L(T, v) = \{\boldsymbol{C}^L : \boldsymbol{C} \in \mathbb{C}(T, v)\}$$
$$\mathbb{C}^R(T, v) = \{\boldsymbol{C}^R : \boldsymbol{C} \in \mathbb{C}(T, v)\}.$$

Then if "$*$" denotes vector concatenation, we have, since every path in the trellis represents a codeword,

$$\mathbb{C}(T, v) = \mathbb{C}^L(T, v) * \mathbb{C}^R(T, v).$$

Now suppose a_1 and b_1 are fixed elements of $\mathbb{C}^L(T, v)$ and $\mathbb{C}^R(T, v)$, respectively. Then if $a \in \mathbb{C}^L(T, v)$ and $b \in \mathbb{C}^R(T, v)$ are arbitrary, we have

$$(a * b) = (a_1 * b_1) + (a - a_1 * 0) + (0 * b - b_1). \tag{5.5}$$

But $(a - a_1 * 0) = (a * b_1) - (a_1 * b_1)$ is a difference of codewords which is zero in positions $i+1, \cdots, n$, and so it is an element of P_i. Similarly, $(0 * b - b_1) = (a_1 * b) - (a_1 * b_1)$ is a difference of codewords which is zero in positions $1, \cdots, i$, and so it is an element of F_i. Thus from (5.5), we see that every codeword in $\mathbb{C}(T, v)$ is an element of the coset of $P_i \oplus F_i$ with "coset leader" $(a_1 * b_1)$. □

The bound (5.4), together with (5.3), immediately implies (5.1), since $|\mathbb{C}| = 2^k$.

To prove (5.2), we proceed similarly. For each edge $e \in E$, define $\mathbb{C}(T, e)$ to be the set of codewords in $\mathbb{C}$ such that the corresponding T-trellis path contains e. Since every trellis path must contain exactly one edge in $E_{i-1,i}$, we have

$$\mathbb{C} = \bigcup_{e \in E_{i-1,i}} \mathbb{C}(T, e), \qquad \text{for } i = 1, \cdots, n. \tag{5.6}$$

The following lemma gives useful information about the sets $\mathbb{C}(T, e)$, analogous to that in Lemma 5.2 about the sets $\mathbb{C}(T, v)$.

Lemma 5.3: If $e \in E_{i-1,i}$, then $\mathbb{C}(T, e)$ is a subset of one of the cosets of $P_{i-1} \oplus F_i$ in $\mathbb{C}$. Thus since each such coset contains $2^{p_{i-1}+f_i}$ elements, we have the upper bound

$$|\mathbb{C}(T, e)| \leq 2^{p_{i-1}+f_i}. \tag{5.7}$$

Proof: With $e \in E_{i-1,i}$, denote by $\mathbb{C}^L(T, e)$ the set of "left parts" $(C_1, \cdots, C_{i-1})$ of the codewords in $\mathbb{C}(T, e)$, and $\mathbb{C}^R(T, e)$, the set of "right parts" $(C_{i+1}, \cdots, C_n)$ of the codewords in $\mathbb{C}(T, e)$. Then if x denotes the label of the edge e, it follows, again because every trellis path must correspond to a codeword, that

$$\mathbb{C}(T, e) = \mathbb{C}^L(T, e) * x * \mathbb{C}^R(T, e).$$

Now suppose a_1 and b_1 are fixed elements of $\mathbb{C}^L(T, e)$ and $\mathbb{C}^R(T, e)$, respectively. Then if $a \in \mathbb{C}^L(T, e)$ and $b \in \mathbb{C}^R(T, e)$ are arbitrary, we have

$$(a * x * b) = (a_1 * x * b_1) + (a - a_1 * 0 * 0) + (0 * 0 * b - b_1). \tag{5.8}$$

But $(a - a_1 * 0 * 0) = (a * x * b_1) - (a_1 * x * b_1)$ is a difference of codewords which is zero in positions $i, \cdots, n$, and so it is an element of P_{i-1}. Similarly, $(0 * 0 * b - b_1) = (a_1 * x * b) - (a - 1 * x * b_1)$ is a difference of codewords which is zero in positions $1, \cdots, i$, and so it is an element of F_i. Thus from (5.8), we see that every codeword in $\mathbb{C}(T, e)$ is an element of the coset of $P_{i-1} \oplus F_i$ with "coset leader" $(a_1 * x * b_1)$.

The bound (5.7), together with (5.6), immediately implies (5.1).

Combining the lower bounds in (5.1) and (5.2), with the results of Theorem 4.6, we see that the BCJR trellis simultaneously minimizes both the number of vertices, and the number of edges, at each depth, among all trellises representing $\mathbb{C}$. In the remainder of this section we will show that the BCJR trellis is unique in this regard.

Before proceeding, we need to introduce some more notation. We will henceforth denote the ubiquitous subcode $P_i \oplus F_i$ by W_i. We note that since $P_{i-1} \subseteq P_i$ and $F_i \subseteq F_{i-1}$, it

follows that $W_{i-1} \cap W_i = P_{i-1} \oplus F_i$. We will denote the coset of W_i to which a given codeword $\boldsymbol{C}$ belongs by $\boldsymbol{C} \bmod W_i$.

We first suppose that (5.1) holds for all indices $i = 0, 1, \cdots, n$. Then by Lemma 5.2, each set $C(T, v)$ is a coset of W_i in $\mathbb{C}$. It follows that every $v \in V_i$ corresponds in a natural way to a unique coset of W_i; namely, the set of codewords for which the corresponding trellis path contains v. We will henceforth assume that the elements of V_i have been relabeled, in this natural way, with the cosets of W_i. Since every edge of the trellis corresponds to a coordinate of at least one codeword, it follows that the trellis, with the vertices relabeled with the cosets of W_i, can be described as follows. (Compare this definition to that in (3.3).) If $\boldsymbol{C} = (C_1, \cdots, C_n)$ is a codeword, there is a path of length n, consisting of the n labeled edges $e_1, \cdots, e_n$, defined as follows:

$$\begin{aligned} \text{init}\,(e_i) &= \boldsymbol{C} \bmod W_{i-1} \\ \text{fin}\,(e_i) &= \boldsymbol{C} \bmod W_i \\ \lambda(e_i) &= C_i. \end{aligned} \tag{5.9}$$

This definition of the trellis is independent of the original vertex labels, and thus all trellises for which (5.1) holds for all indices are isomorphic to each other. But as we have seen, the BCJR trellis has this property, and so all vertex-minimal trellises must be isomorphic to the BCJR trellis. (Indeed, the definition (5.9) is equivalent to the definition Forney offered in 1988 for the "trellis diagram" of a code [11].)

Finally, we suppose that (5.2) holds for all indices $i = 1, \cdots, n$. We will show that this implies that the trellis must be isomorphic to the BCJR trellis.

According to Lemma 5.3, if (5.2) holds for the index i, then every edge $e \in E_{i-1,i}$ corresponds, in a natural way, to a coset of $W_{i-1} \cap W_i$ in $\mathbb{C}$; namely, the set of codewords whose trellis paths include e. By Lemma 5.2, every vertex in V_{i-1} must correspond to a subset of a coset of W_{i-1}. Now every coset of W_{i-1} is a union of exactly $|W_{i-1}|/|W_{i-1} \cap W_i|$ cosets of $W_{i-1} \cap W_i$, so that the out-degree $\rho^+(v)$, for each vertex $v \in V_{i-1}$, must satisfy

$$\rho(v) \leq |W_{i-1}|/|W_{i-1} \cap W_i| \tag{5.10}$$

with equality if and only if $\mathbb{C}(T, v)$ is a complete coset of W_{i-1}.

Before continuing with the proof, we will need a simple lemma about counting paths in trellises. We suppose that $T = (V, E)$ is a trellis of depth n as defined in Section I, and denote by ρ_i^+ the maximum out-degree at depth i, i.e.

$$\rho_i^+ = \max_{v \in V_i} \rho^+(v). \tag{5.11}$$

Also denote by t_i the total number of trellis paths from the source A to some vertex at depth i.

Lemma 5.4: We have, for $i = 1, \cdots, n$

$$t_i \leq \prod_{j=0}^{i-1} \rho_i^+ \tag{5.12}$$

with equality if and only if $\rho^+(v) = \rho_j^+$ for all $v \in V_j$, for $j = 0, 1, \cdots, i-1$.

Proof: We use induction on i. The value of t_1 is $\rho^+(A)$, which is by (5.11) also the value of ρ_0^+, and so (5.12) is true for $i = 1$. (The condition for equality holds automatically in this case, since V_0 contains only one vertex.)

Assuming now that (5.12) is true for the index i, we move on and consider the value of t_{i+1}. Every path from A to some vertex at depth $i+1$ passes through a unique vertex at depth i; so if we denote by $t(v)$ the total number of paths from A to v, we have

$$t_{i+1} = \sum_{v \in V_i} t(v) \rho^+(v).$$

Thus we have

$$\begin{aligned} t_{i+1} &= \sum_{v \in V_i} t(v) \rho^+(v) \\ &\leq \left(\sum_{v \in V_i} t(v) \right) \cdot \rho_i^+ \qquad (5.13) \\ &\leq \left(\prod_{j=0}^{i-1} \rho_j^+ \right) \cdot \rho_i^+ \qquad (5.14) \\ &= \prod_{j=0}^{i} \rho_j^+. \end{aligned}$$

The inequality in (5.13) holds because of the definition (5.11), and since (by our definition of a trellis) no $t(v)$ in the sum is zero, equality holds in (5.13) if and only if $\rho^+(v) = \rho_i^+$ for all $v \in V_i$. The inequality in (5.14) holds because of the induction assumption, and equality holds in (5.14), because of the induction assumption, if and only if $\rho^+(v) = \rho_j^+$, for all $v \in V_j$, for $j = 0, \cdots, i-1$. This completes the proof of the Lemma. □

Now we can complete the proof of the last part of Theorem 5.1. If we combine (5.10) with Lemma 5.4, we obtain an upper bound on the total number of paths of length n through the trellis. But since the trellis represents $\mathbb{C}$, which is an (n, k)-linear code, it follows that this number is $\geq 2^k$ (some codewords might be represented by more than one trellis path). Thus

$$\begin{aligned} 2^k &\leq \prod_{i=0}^{n-1} \frac{|W_i|}{|W_i \cap W_{i+1}|} = \prod_{i=0}^{n-1} \frac{|P_i \oplus F_i|}{|P_i \oplus F_{i+1}|} \\ &= \prod_{i=0}^{n-1} \frac{|F_i|}{|F_{i+1}|} = \frac{|F_0|}{|F_n|} = 2^k. \end{aligned} \tag{5.15}$$

(The last equality by (4.15)). Thus by Lemma 5.4, equality holds in (5.10), for all $v \in V_i$. This means that each vertex in V_i corresponds to an entire coset of W_i, and this in turn implies that equality holds in (5.1) for all $i = 1, \cdots, n$. But we have already seen that this implies that the trellis is isomorphic to the BCJR trellis. This completes the proof. □

Theorem 5.1 says that the BCJR trellis is locally, as well as globally, minimal, since it minimizes not only $|E|$ and $|V|$, but also $|V_i|$ and $|E_{i-1,i}|$, for each index i. It is important to note, however, that some non-BCJR trellises have the same values of $|V_i|$ and/or $|E_{i-1,i}|$ as the BCJR trellis for *some* values of the index i. For example, the trellis of Fig. 3 has the same

values of $|V_i|$ as the BCJR trellis (Fig. 7) for $i = 0, 2, 3, 4, 7$, and the same values of $|E_{i-1,i}|$ for $i = 2, 3, 4, 5$. But only the BCJR trellis simultaneously minimizes all these quantities.

Along these same lines, we note that the quantity $s = \max_i |V_i|$, often called the *state complexity*, has been taken by several authors [18], [19], [34], [35], [40], [42] as a measure of trellis complexity. The BCJR trellis certainly minimizes the state complexity, but it is not unique in this respect, as the trellis in Fig. 3 illustrates. Still, we should point out that in [34], it is argued cogently that s is the "right" measure of the complexity for the design of a VLSI circuit for decoding using a trellis. In any case, we can say, in view of Theorem 5.1, that the BCJR trellis uniquely minimizes the closely related quantity

$$S = \sum_{i=1}^{n} |V_i|.$$

VI. Minimal-Span Generator Matrices

In [11, Appendix A], Forney, in an elliptical remark, asserted the existence of a useful class of generator matrices for linear codes which he called "trellis oriented." Although this remark has since been nicely elaborated on in [22], the results in this section are an attempt to do the same thing, in a somewhat different way. (Also, in [13], Forney and Trott consider trellis structures for the general class of group codes, and introduce the notion of "granules." When specialized to binary block codes, a granule turns out to be a codeword which can appear as a row of a minimal-span generator matrix.)

We begin with some definitions. If $\boldsymbol{x} = (x_1, x_2, \cdots, x_n)$ is a nonzero binary n-vector, its *left index*, denoted $L(\boldsymbol{x})$, is the smallest index i such that $x_i \neq 0$. Similarly, the *right index* of $\boldsymbol{x}$, denoted $R(\boldsymbol{x})$ is the largest index i such that $x_i \neq 0$. The *span* of $\boldsymbol{x}$, denoted $\text{Span}\,(\boldsymbol{x})$, is the discrete "interval" $[s, t] = (L(\boldsymbol{x}), L(\boldsymbol{x}) + 1, \cdots, R(\boldsymbol{x}))$. The *spanlength* of $\boldsymbol{x}$, denoted $\text{spanlength}\,(\boldsymbol{x})$, is the number of elements in $\text{Span}\,(\boldsymbol{x})$, i.e., $\text{spanlength}\,(\boldsymbol{x}) = |\text{Span}\,(\boldsymbol{x})|$.

A nonzero vector $\boldsymbol{x} = (x_1, \cdots, x_n)$ is said to be *active at coordinate* i if $i \in \text{Span}\,(\boldsymbol{x})$, i.e., $L(\boldsymbol{x}) \leq i$ and $R(\boldsymbol{x}) \geq i$. Similarly, $\boldsymbol{x}$ is said to be *active at depth* i $(i = 0, 1, \cdots, n)$, if both i and $i + 1$ are in $\text{Span}\,(\boldsymbol{x})$, i.e., if $L(\boldsymbol{x}) \leq i$ and $R(\boldsymbol{x}) \geq i + 1$. (We will need these definitions in Corollary 6.16, below.)

If G is a $k \times n$ binary matrix, with rows $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_k$ (which from now on we will indicate with the notation $G = (\boldsymbol{x}_1, \cdots, \boldsymbol{x}_k)$), its *span set* is the set of row spans, i.e.

$$\{[L(\boldsymbol{x}_1), R(\boldsymbol{x}_1)], \cdots, [L(\boldsymbol{x}_k), R(\boldsymbol{x}_k)]\}$$

and its *spanlength*, denoted $\text{spanlength}\,(G)$, is the sum of the spanlengths of the rows.

Example 6.1: Consider the following generator matrix for a $(7, 3, 3)$ linear code, which is the same as the code from Example 3.1

$$G_1 = \begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{1} \\ \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} & 0 & 0 \\ 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 \end{pmatrix}. \tag{6.1}$$

If we denote the rows of G by $\boldsymbol{x}_1$, $\boldsymbol{x}_2$, and $\boldsymbol{x}_3$, then $L(\boldsymbol{x}_1) = 1$, $R(\boldsymbol{x}_1) = 7$, and $\text{spanlength}\,(\boldsymbol{x}_1) = 7$; $L(\boldsymbol{x}_2) = 1$, $R(\boldsymbol{x}_2) = 5$, and $\text{spanlength}\,(\boldsymbol{x}_2) = 5$; and $L(\boldsymbol{x}_3) = 2$, $R(\boldsymbol{x}_3) = 4$, and $\text{spanlength}\,(\boldsymbol{x}_3) = 3$. The active elements in each row are shown in boldface. The vector $\boldsymbol{x}_3$ is active at coordinates 2, 3, and 4, and is active at depths 2 and 3. The span set of G_1 is therefore $\{[1, 7], [1, 5], [2, 4]\}$, and $\text{spanlength}\,(G_1) = 15$. □

Definition 6.2: Let $\mathbb{C}$ be an (n, k) binary linear code. Among all generator matrices for $\mathbb{C}$, those for which the spanlength is as small as possible are called *minimal span generator matrices*, abbreviated MSGM's.

In this section we will see that MSGM's have many useful and interesting properties, among them the property of being trellis-oriented. The key to these properties are two other properties, the left–right property and the predictable span property, which we now introduce.

Definition 6.3: We say that a set of binary vectors $\{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_k\}$ has the *left–right property* (LR Property), if $L(\boldsymbol{x}_i) \neq L(\boldsymbol{x}_j)$, and $R(\boldsymbol{x}_i) \neq R(\boldsymbol{x}_j)$, whenever $i \neq j$.

Example 6.4: The rows of the matrix $G_1 = (\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3)$ in (3.1) do not have the LR property, since $L(\boldsymbol{x}_1) = L(\boldsymbol{x}_2) = 1$. However, the rows of the row-equivalent matrix $G_2 = (\boldsymbol{x}_2 + \boldsymbol{x}_3, \boldsymbol{x}_3, \boldsymbol{x}_1 + \boldsymbol{x}_2)$, i.e.

$$G_2 = \begin{pmatrix} \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} & 0 & 0 \\ 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} \end{pmatrix} \tag{6.2}$$

do have the LR property, since its span set is $\{[1, 5], [2, 4], [5, 7]\}$. Here $\text{spanlength}(G_2) = 11$.

Note that for any two n-vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, $\text{Span}\,(\boldsymbol{x} + \boldsymbol{y}) \subseteq \text{Span}\,(\boldsymbol{x}) \bigcup \text{Span}\,(\boldsymbol{y})$, with equality if and only if $L(\boldsymbol{x}) \neq L(\boldsymbol{y})$ and $R(\boldsymbol{x}) \neq R(\boldsymbol{y})$. The next definition and Lemma generalize this observation.

Definition 6.5: A set of binary n-vectors $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_k\}$ is said to have the *predictable span property*, if

$$\text{Span}\left(\sum_{j \in J} \boldsymbol{x}_j\right) = \bigcup_{j \in J} \text{Span}\,(\boldsymbol{x}_j) \tag{6.3}$$

for all subsets $J \subseteq \{1, 2, \cdots, k\}$.

Lemma 6.6: A set of binary n-vectors $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_k\}$ has the predictable span property if and only if it has the LR property.

Proof: It is clear that

$$\text{Span}\left(\sum_{j \in J} \boldsymbol{x}_j\right) \subseteq \bigcup_{j \in J} \text{Span}\,(\boldsymbol{x}_j).$$

The inclusion will be strict only if there is "cancellation" at either the left or right ends of the $\text{Span}\,(\boldsymbol{x}_j)$'s. But such cancellation is possible if and only if some of the left or right endpoints of the $\text{Span}\,(\boldsymbol{x}_j)$'s are equal, i.e., if the LR property fails to hold. □

Our first result of significance is the following.

Lemma 6.7: If G is an MSGM for the code $\mathbb{C}$, then the rows of G have the LR property, and so also, by Lemma 6.6, the predictable span property.

Proof: Suppose that $G = (\boldsymbol{x}_1, \cdots, \boldsymbol{x}_k)$, and that the LR property fails to hold. Then for some pair (i, j) we have either $L(\boldsymbol{x}_i) = L(\boldsymbol{x}_j)$ or $R(\boldsymbol{x}_i) = R(\boldsymbol{x}_j)$. Thus without essential loss of generality we can assume that $L(\boldsymbol{x}_1) = L(\boldsymbol{x}_2)$ and $R(\boldsymbol{x}_1) \geq R(\boldsymbol{x}_2)$. But then spanlength $(\boldsymbol{x}_1 + \boldsymbol{x}_2) <$ spanlength $(\boldsymbol{x}_1)$, so that if we define $\boldsymbol{x}_1' = \boldsymbol{x}_1 + \boldsymbol{x}_2$, it follows that the generator matrix $G' = (\boldsymbol{x}_1', \boldsymbol{x}_2, \cdots, \boldsymbol{x}_k)$ has spanlength strictly less than the spanlength of G. In other words, if the LR property fails to hold for the rows of G, then G cannot be an MSGM. □

Our first main result is that minimal span generator matrices are minimal in a very strong sense. In what follows, a generator matrix for $\mathbb{C}$ whose rows have the LR property will be called an LR-generator matrix.

Theorem 6.8: If $G = (\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_k)$ is an LR-generator matrix for the code $\mathbb{C}$, and if $G' = (\boldsymbol{x}_1', \boldsymbol{x}_2', \cdots, \boldsymbol{x}_k')$ is any other generator matrix for $\mathbb{C}$, then it is possible to rearrange the rows of G so that

$$\text{Span}\,(\boldsymbol{x}_j) \subseteq \text{Span}\,(\boldsymbol{x}_j'), \qquad \text{for } j = 1, 2, \cdots, k.$$

Proof: Each $\boldsymbol{x}_j'$ is a linear combination of a subset of the $\boldsymbol{x}_j$'s, say

$$\boldsymbol{x}_j' = \sum_{i \in I_j} \boldsymbol{x}_i. \tag{6.4}$$

But by Lemma 6.6, $(\boldsymbol{x}_1, \cdots, \boldsymbol{x}_k)$ has the predictable span property, so that by (6.4), we have

$$\text{Span}\,(\boldsymbol{x}_i) \subseteq \text{Span}\,(\boldsymbol{x}_j'), \qquad \text{if } i \in I_j. \tag{6.5}$$

Furthermore, if $1 \leq s \leq k$, any collection of s distinct I_j's must contain at least s distinct $\boldsymbol{x}_i$'s, since otherwise we would have a collection of fewer than s $\boldsymbol{x}_i$'s spanning an s-dimensional space, which is impossible. Thus by Philip Hall's "marriage theorem" (see [14, Theorem 5.1.1], [28, Theorem 8.7], or [31, sec. 4.3)] the I_j's contain a system of distinct representatives, say (after renumbering) $\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_k$. Then by (6.5), we have

$$\text{Span}\,(\boldsymbol{x}_j) \subseteq \text{Span}\,(\boldsymbol{x}_j'), \qquad \text{for } j = 1, 2, \cdots, k$$

as asserted. □

Example 6.9: We saw above that the generator matrix G_2 in (6.2) has the LR property, whereas the row-equivalent matrix G_1 in (6.1) does not. To verify that Theorem 6.8 holds in this case, we compare the span sets of G_1 and G_2, and find that, indeed, $[5, 7] \subseteq [1, 7]$, $[1, 5] \subseteq [1, 5]$, and $[2, 4] \subseteq [2, 4]$.

Corollary 6.10 (Kschischang and Sorokine [22]): Any two LR-generator matrices have the same span set.

Proof: Let $G = (\boldsymbol{x}_1, \cdots, \boldsymbol{x}_k)$ and $G' = (\boldsymbol{x}_1', \cdots, \boldsymbol{x}_k')$ be two LR-generator matrices for $\mathbb{C}$. Then by Theorem 6.8, by reordering the rows of G, if necessary, we have

$$\text{Span}\,(\boldsymbol{x}_j) \subseteq \text{Span}\,(\boldsymbol{x}_j'), \qquad \text{for } j = 1, 2, \cdots, k. \tag{6.6}$$

Similarly, for a suitable permutation π of $\{1, 2, \cdots, k\}$, we have

$$\text{Span}\,(\boldsymbol{x}_j') \subseteq \text{Span}\,(\boldsymbol{x}_{\pi(j)}), \qquad \text{for } j = 1, 2, \cdots, k. \tag{6.7}$$

It follows from (6.6) and (6.7), that $\text{Span}\,(\boldsymbol{x}_j) = \text{Span}\,(\boldsymbol{x}_j')$, for $j = 1, 2, \cdots, k$. □

The main theoretical result about MSGM's follows. It is an almost immediate corollary to Theorem 6.8.

Theorem 6.11 (Kschischang and Sorokine [22]): A matrix G is an MSGM if and only if it has the LR property. Any two MSGM's have the same span sets.

Proof: By Lemma 6.7, any MSGM has the LR property. Now suppose that G is an LR-generator matrix, and G_0 is an MSGM. Then by Corollary 6.10, G and G_0 have the same span set, and so also the same spanlength. Thus since G_0 has minimal spanlength, so does G. □

Example 6.12: Let $\mathbb{C}$ be the $(7, 3, 3)$ code defined by the generator matrix $G_1 = (\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3)$ given in (6.1), and define $G_2 = (\boldsymbol{x}_2 + \boldsymbol{x}_3, \boldsymbol{x}_3, \boldsymbol{x}_1 + \boldsymbol{x}_2)$ (see (6.2)) and $G_3 = (\boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_1 + \boldsymbol{x}_2)$, i.e.

$$G_2 = \begin{pmatrix} \mathbf{1} & \mathbf{1} & \mathbf{0} & 1 & 1 & 0 & 0 \\ 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} \end{pmatrix}$$

$$G_3 = \begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} & 1 & 0 & 0 \\ 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} \end{pmatrix}.$$

Then both G_2 and G_3 are generator matrices for $\mathbb{C}$ with the LR property, so by Theorem 6.11, both are MSGM's for $\mathbb{C}$. Their common span set is $\{[1, 5], [2, 4], [5, 7]\}$, and their common spanlength, which is the minimum possible spanlength among all generator matrices for $\mathbb{C}$, is 11. This shows that a given code can have several essentially different MSGM's. However, in this case it is easy to see that there are no other MSGM's for $\mathbb{C}$, apart from those that can be obtained by permuting the rows of G_2 or G_3.

The question now arises as to how to produce an MSGM for a given code. One approach, which we might call a "greedy" approach, is to select the rows of G sequentially, with each new row being of smallest possible spanlength subject to the constraint of being linearly independent of the rows already chosen. The algorithm is described formally by the following pseudocode fragment:

```
/* Greedy Algorithm I for finding a
        minimal span generator matrix */
{
x_0 = 0;
for (i == 1 to k)
  x_i = a codeword independent of {x_0, ..., x_{i-1}}
    of smallest possible spanlength;
}
```

Surprisingly, "Greedy Algorithm I" does always produce an MSGM.

Theorem 6.13: A generator matrix produced by "Greedy Algorithm I" will be an MSGM.

Proof: We first note that the operation of the algorithm guarantees that

$$\text{spanlength}\,(\boldsymbol{x}_1) \leq \text{spanlength}\,(\boldsymbol{x}_2) \leq \cdots \leq \text{spanlength}\,(\boldsymbol{x}_k).$$

To prove the theorem, we will show, by induction on j, that the set $\{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_j\}$ has the LR property, for $j = 1, 2, \cdots, k$. It will then follow from Theorem 6.11 that G is an MSGM.

For $j = 1$, there is nothing to prove. Assume that $\{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_j\}$ has the LR property, but that $\{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_j, \boldsymbol{x}_{j+1}\}$ does not. Then it must be the case that either $L(\boldsymbol{x}_{j+1}) = L(\boldsymbol{x}_i)$, or $R(\boldsymbol{x}_{j+1}) = R(\boldsymbol{x}_i)$, for some $i \in \{1, \cdots, j\}$. In either case, since as noted above

$$\text{spanlength}\,(\boldsymbol{x}_{j+1}) \geq \text{spanlength}\,(\boldsymbol{x}_i)$$

it follows that

$$\text{spanlength}\,(\boldsymbol{x}_{j+1} + \boldsymbol{x}_i) < \text{spanlength}\,(\boldsymbol{x}_{j+1}).$$

Now by the definition of the algorithm, $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_j, \boldsymbol{x}_{j+1}$ are linearly independent, and so also are $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_j, \boldsymbol{x}_{j+1} + \boldsymbol{x}_i$. But we have seen that

$$\text{spanlength}\,(\boldsymbol{x}_{j+1} + \boldsymbol{x}_i) < \text{spanlength}\,(\boldsymbol{x}_{j+1})$$

which contradicts the selection of $\boldsymbol{x}_{j+1}$ as having the smallest possible spanlength among vectors independent from $\{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_j\}$. □

Greedy Algorithm I, while straighforward and correct, is not of much practical value, since there is no obvious way to find the minimal spanlength codewords required, apart from exhaustive search. However, there is another "greedy" algorithm that is practical, and which also produces an MSGM, using a sequence of elementary row operations. It is based on Theorem 6.11, and is described by the following pseudocode fragment:

```
/* Greedy Algorithm II for finding a
   minimal span generator matrix */
/* Due to Kschischang and Sorokine [22] */
while (LR property fails to hold)
{
   find a pair (i, j) such that (L(x_i) = L(x_j)
            and R(x_i) ≤ R(x_j))
    or (R(x_i) = R(x_j) and L(x_i) ≥ L(x_j));
x_j = x_j + x_i;
}
```

This algorithm is based on the fact that if the LR property fails to hold, say if $L(\boldsymbol{x}_1) = L(\boldsymbol{x}_2)$ and $R(\boldsymbol{x}_1) \geq R(\boldsymbol{x}_2)$, then by replacing $\boldsymbol{x}_1$ with $\boldsymbol{x}_1 + \boldsymbol{x}_2$, the spanlength of G will be reduced, whereas if the LR property does hold, the matrix is automatically in minimal-span form, by Theorem 6.11.

Example 6.14: Consider the matrix G_1 in (6.1). Since it does not have the LR property, it is not an MSGM. Let us therefore apply "Greedy Algorithm II." Since $L(\boldsymbol{x}_1) = L(\boldsymbol{x}_2)$, and $R(\boldsymbol{x}_1) > R(\boldsymbol{x}_2)$, we replace $\boldsymbol{x}_1$ with $\boldsymbol{x}_1 + \boldsymbol{x}_2$; the result is a matrix row-equivalent to G_1 with smaller spanlength

$$G_1' = \begin{pmatrix} 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \end{pmatrix}.$$

The span set of G_1' is seen to be $\{[5,7],[1,5],[2,4]\}$, so that G_1' has the LR property, and so by Theorem 6.11, it is also MSGM for the code. Indeed, it is, apart from a row permutation, the same as the MSGM G_3 in Example 6.12. □

We are now prepared to show that MSGM's are "trellis-oriented" in the sense that the dimensions p_i and f_i of the past and future subcodes (see (4.7)–(4.8)) can be read directly from the matrix. To simplify the notation, we suppose that $G = (\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_k)$ is a fixed MSGM for the linear code $\mathbb{C}$, and write $L_i = L(\boldsymbol{x}_i)$ and $R_i = R(\boldsymbol{x}_i)$, for $i = 1, 2, \cdots, k$.

Theorem 6.15: For $i = 0, 1, \cdots, n$, we have

$$p_i = |\{j\colon R_j \leq i\}| \tag{6.8}$$

$$f_i = |\{j\colon L_j \geq i+1\}|. \tag{6.9}$$

In words, p_i is the number of rows of G for which the rightmost nonzero entry lies in column i or earlier, and f_i is the number of rows of G for which the leftmost nonzero entry lies in column $i+1$ or later.

Proof: We will prove (6.8). The proof of (6.9) is essentially the same. Clearly, every $\boldsymbol{x}_j$ such that $R_j \leq i$ belongs to the subcode P_i. Thus $p_i \geq |\{j\colon R_j \leq i\}|$. On the other hand, any codeword $\boldsymbol{x} \in P_i$, i.e., with $R(\boldsymbol{x}) \leq i$, must, by the "predictable span" property (see Lemma 6.7), be a linear combination of the codewords in the set $\{j\colon R_j \leq i\}$. Hence $p_i \leq |\{j\colon R_j \leq i\}|$ as well. □

The following Corollary shows that the important vertex and edge dimensions s_i and b_i (see (4.3) and (4.4)) for the BCJR trellis can also be read directly from an MSGM. In the statement, we recall that a vector x is said to be active at coordinate i if $i \in \text{Span}\,(\boldsymbol{x})$, and we say $\boldsymbol{x}$ is active at depth i if i and $i+1$ are both in $\text{Span}\,(\boldsymbol{x})$.

Corollary 6.16: If G is an MSGM for the code $\mathbb{C}$, then the number of rows of G which are active at depth i is s_i, and the number of rows which are active at coordinate i is b_i.

Proof: We use Theorem 6.15. If a row $\boldsymbol{x}$ is *not* active at depth i, then either $L(\boldsymbol{x}) \geq i+1$ or $R(\boldsymbol{x}) \leq i$, but not both, since $L(\boldsymbol{x}) \leq R(\boldsymbol{x})$. The number of rows with $L(\boldsymbol{x}) \geq i+1$ is f_i, by (6.9). Similarly, the number of rows with $R(\boldsymbol{x}) \leq i$ is p_i, by (6.8). Since G has k rows altogether, it follows that the number of rows active at depth i is $k - p_i - f_i = s_i$.

Similarly, if a row is *not* active at coordinate i, then either $L(\boldsymbol{x}) \geq i+1$ or $R(\boldsymbol{x}) \leq i-1$, but not both. By (6.9), the number of rows with $L(\boldsymbol{x}) \geq i+1$ is f_i; by (6.8), the number of rows with $R(\boldsymbol{x}) \leq i-1$ is p_{i-1}. Thus the number of rows active at coordinate i is $k - p_{i-1} - f_i = b_i$. □

There is a result dual to Theorem 6.15, which we now present, in which the past and future subcodes P_i and F_i are replaced with the past and future projections introduced in (4.11) and (4.12).

Theorem 6.17: For $i = 0, 1, \cdots, n$, we have

$$p^i = |\{j\colon L_j \leq i\}| \tag{6.10}$$

$$f^i = |\{j\colon R_j \geq i+1\}|. \tag{6.11}$$

In words, p^i is the number of rows of G for which the leftmost nonzero entry lies in column i or earlier, and f^i is the number of rows of G for which the rightmost nonzero entry lies in column $i+1$ or later.

Proof: The proof is similar to the proof of Theorem 6.15 and is omitted. □

There is a corollary to Theorem 6.17 which shows that there is a strong similarity between MSGM's, and *row-reduced echelon* (RRE) generator matrices. It is well known that every linear code has a unique RRE generator matrix (see, for example, [30, Theorem 7.1]). For our purposes, we shall call

such a generator matrix a "left" RRE generator matrix, because its definition requires that every column containing the *leftmost* 1 in some row have all its other entries 0. If we replace the word *leftmost* with *rightmost*, we obtain what might be called the "right" RRE generator matrix.

Corollary 6.18: If $G^L_{\rm RRE}$ is the "left" RRE generator matrix, $G^R_{\rm RRE}$ is the "right" RRE generator matrix, and $G_{\rm MS}$ is an MSGM, for the code $\mathbb{C}$, then the left indices of $G^L_{\rm RRE}$ are the same as the left indices of $G_{\rm MS}$, and the right indices of $G^R_{\rm RRE}$ are the same as the right indices of $G_{\rm MS}$.

Proof: Assume $G^L_{\rm RRE} = (\boldsymbol{y}_1, \cdots, \boldsymbol{y}_k)$, and let us use $G^L_{\rm RRE}$ to compute the dimensions p^i of the ith past projection of $\mathbb{C}$. Plainly, p^i is the rank of the matrix formed by taking the first i columns of $G^L_{\rm RRE}$. But this matrix will have exactly $|\{j\colon L(\boldsymbol{y}_j) \le i\}|$ nonzero rows, and these rows must, by the defining property of a left RRE matrix, be linearly independent. Thus $p^i = |\{j\colon L(\boldsymbol{y}_j) \le i\}|$. Comparing this result to (6.10), we see that the sets $\{L(\boldsymbol{x}_1), \cdots, L(\boldsymbol{x}_k)\}$ and $\{L(\boldsymbol{y}_1), \cdots, L(\boldsymbol{y}_k)\}$ must be identical. The assertion about the right indices is proved similarly. □

Example 6.19: For the $(7, 3, 3)$ code specified by the generator matrix in (6.1), a short calculation shows that

$$G^L_{\rm RRE} = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}$$

$$G^R_{\rm RRE} = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \end{pmatrix}.$$

By inspection, we see that the left indices of $G^L_{\rm RRE}$ are $(1, 2, 5)$ and the right indices of $G^R_{\rm RRE}$ are $(7, 5, 4)$ which indeed are the left and right indices of $G_{\rm MS}$, as we saw in Example 6.12.

Example 6.20: Let us return to the MSGM G_2 from (6.2). Using Theorems 6.15 and 6.17, we find the following table:

i	p_i	f_i	p^i	f^i	s_i	b_i
0	0	3	0	3	0	–
1	0	2	1	3	1	1
2	0	1	2	3	2	2
3	0	1	2	3	2	2
4	1	1	2	2	1	2
5	2	0	3	1	1	2
6	2	0	3	1	1	1
7	3	0	3	0	0	1

We computed this same table in Example 4.7. However, there we needed the trellis, whereas here, we only needed the minimal-span generator matrix for $\mathbb{C}$, which is much easier to obtain. In the next section, we will see how an MSGM can be used to efficiently construct the BCJR trellis.

We conclude this section with a brief discussion of the "Massey trellis." Shortly after the appearance of Wolf's paper [43], Massey [29] introduced an alternative trellis construction for block codes. It turns out that Massey's trellis is also isomorphic to the BCJR trellis. Here let us sketch a proof. If the n coordinate positions of an (n, k) linear block code $\mathbb{C}$ are partitioned into k information positions and $n - k$ parity-check positions, where the information positions are the left indices of an RRE generator matrix for $\mathbb{C}$, Massey defines the "state" of a codeword $\boldsymbol{C}$ at depth i as the vector of parity bits in positions $i + 1, \cdots, n$, as determined by the information bits in $(C_1, \cdots, C_i)$. It is then relatively easy to verify, using Corollary 6.18, that the "vertex kernels" of the Massey trellis are in fact the "minimal kernels" $P_i \oplus F_i$, and so by Theorem 5.1, the Massey trellis is isomorphic to the BCJR trellis, a result announced by Kot and Leung in 1993 [21].

VII. Constructing the BCJR Trellis from a Minimal-Span Generator Matrix

In this section we will describe a general method for using an MSGM for $\mathbb{C}$ to construct a family of trellises for $\mathbb{C}$, which we call the *simple linear trellises*. We will see that the BCJR trellis is a simple linear trellis, and so the results of this section give an efficient algorithm for constructing the BCJR trellis. (Also, the important *segmented* trellises of [27] are simple linear trellises.)

Thus let $G = (\boldsymbol{x}_1, \cdots, \boldsymbol{x}_k)$ be a minimal-span generator matrix for $\mathbb{C}$, and let $S = (S_1, \cdots, S_k)$ be a set of *row covers* for G. By this we mean that each S_j is a discrete interval $[L_j, R_j]$ containing the span of $\boldsymbol{x}_j$. For a given set of row covers S, we define, for $i = 1, \cdots, n$, the sets A_i as follows:

$$A_i = \{j\colon i \in S_j\}. \tag{7.1}$$

In words, A_i represents the (indices of the) rows of G which are "S-active" at coordinate i. Similarly, we define, for $i = 0, \cdots, n$, the sets B_i as follows:

$$\begin{aligned} B_0 &= B_n = \emptyset \\ B_i &= A_i \cap A_{i+1}, \quad \text{for } i = 1, \cdots, n-1. \end{aligned} \tag{7.2}$$

The set B_i represents the (indices of the) rows of G that are "S-active" at depth i. We denote the cardinalities of A_i and B_i by α_i and β_i, respectively.

Example 7.1: We use the code and MSGM from Example 6.4, with row covers $S = \{[1, 5], [1, 5], [5, 7]\}$. Then we can display the generator matrix G, and the row covers S as follows:

$$G = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \begin{pmatrix} \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} & 0 & 0 \\ \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{0} & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} \end{pmatrix} \end{array}$$

with the entries in each row corresponding to the elements of the row covers indicated in boldface. The following table gives the A_i's, B_i's, α_i's, and β_i's:

i	A_i	B_i	α_i	β_i
0	$\emptyset$	$\emptyset$	0	0
1	$\{1, 2\}$	$\{1, 2\}$	2	2
2	$\{1, 2\}$	$\{1, 2\}$	2	2
3	$\{1, 2\}$	$\{1, 2\}$	2	2
4	$\{1, 2\}$	$\{1, 2\}$	2	2
5	$\{1, 2, 3\}$	$\{3\}$	3	1
6	$\{3\}$	$\{3\}$	1	1
7	$\{3\}$	$\emptyset$	1	0

The (G, S)-*trellis* for $\mathbb{C}$ is now defined as follows. For each $i = 0, 1, \cdots, n$, the vertex set V_i is the set 2^{B_i} of subsets of B_i, each represented by a binary β_i-tuple. Each codeword $\boldsymbol{C} = (C_1, \cdots, C_n)$, which is necessarily of the form $\boldsymbol{C} = uG$ for a unique k-tuple u, corresponds to a path of length n in the trellis, consisting of the edges $e_1(u), \cdots, e_n(u)$, where $e_i = e_i(u) \in E_{i-1,i}$ is defined by

$$\begin{aligned} \text{init}\,(e_i) &= u \cap B_{i-1} \\ \text{fin}\,(e_i) &= u \cap B_i \\ \lambda(e_i) &= C_i. \end{aligned} \tag{7.3}$$

In (7.3), if $u = (u_1, \cdots, u_k)$ is a binary k-vector, and if $B = \{j_1, \cdots, j_s\}$ is a subset of $\{1, 2, \cdots, k\}$, the notation "$u \cap B$" represents the binary s-vector obtained by extracting the components of u corresponding to the elements of B, i.e., $u \cap B = (u_{j_1}, \cdots, u_{j_s})$. As usual, edges with the same values of init (e), fin (e), and $\lambda(e)$ are considered to be identical.

The following Theorem gives the basic combinatorial information about the (G, S)-trellis (compare this to Theorem 4.6).

Theorem 7.2: The number of vertices at depth i in the (G, S)-trellis is

$$|V_i| = 2^{\beta_i} \tag{7.4}$$

for $i = 0, 1, \cdots, n$. Similarly, the number of edges connecting vertices at depth $i - 1$ to those at depth i is

$$|E_{i-1,i}| = 2^{\alpha_i} \tag{7.5}$$

for $i = 1, \cdots, n$. Finally, all vertices $v \in V_i$ have common out- and in-degrees, denoted by ρ_i^+ and ρ_i^-, where

$$\rho_i^+ = 2^{\alpha_{i+1} - \beta_i}, \qquad \text{for } i = 0, 1, \cdots, n-1 \tag{7.6}$$

$$\rho_i^- = 2^{\alpha_i - \beta_i}, \qquad \text{for } i = 1, 2, \cdots, n. \tag{7.7}$$

Proof: From the definition (7.3), it follows that if $u \cap A_i = 0$, then the edge $e = e_i(u)$ has init $(e) =$ fin $(e) = \lambda(e) = 0$. Thus if u_1 and u_2 agree on A_i, then $e_i(u_1) = e_i(u_2)$, so that the edge set $E_{i-1,i}$ can be defined as the set 2^{A_i} of subsets of A_i, each represented by a binary α_i-tuple u, with init (u), fin (u), and $\lambda(u)$ defined as follows:

$$\begin{aligned} \text{init}\,(u) &= u \cap B_{i-1} \\ \text{fin}\,(u) &= u \cap B_i \\ \lambda(u) &= u \cdot \tilde{g}_i \end{aligned} \tag{7.8}$$

where $\tilde{g}_i = g_i \cap A_i$, g_i being the ith column of G.

It follows from (7.8) that the (G, S)-trellis has $|V_i| = 2^{\beta_i}$ for $i = 0, \cdots, n$, and $|E_{i-1,i}| = 2^{\alpha_i}$, for $i = 1, \cdots, n$. This proves (7.4) and (7.5).

To prove (7.6), let $v \in V_i$, and let $e \in E_{i,i+1}$, with init $(e) = v$. Then according to (7.8), e corresponds to a subset $u \subseteq A_{i+1}$ such that $u \cap B_i = v$. Thus the edges in $E_{i,i+1}$ with init $(e) = v$ are in one-to-one correspondence with the subsets of $A_{i+1} - B_i$, and there are $2^{\alpha_{i+1} - \beta_i}$ such subsets. Thus $\rho^+(v) = 2^{\alpha_{i+1} - \beta_i}$ for all $v \in E_{i,i+1}$. This proves (7.6). The proof of (7.5) is similar. □

Example 7.3: Let us construct the (G, S)-trellis, where G and S are as given in Example 7.1. We begin with the edge set $E_{0,1}$, using the recipe given in (7.8). We have

$$\begin{aligned} V_0 &= 2^{B_0} = 2^{\emptyset} = \{\emptyset\} \\ V_1 &= 2^{B_1} = 2^{\{1,2\}} = \{00, 01, 10, 11\} \\ A_1 &= 2^{A_1} = 2^{\{1,2\}} = \{00, 01, 10, 11\} \end{aligned}$$

and

$$\tilde{g}_1 \cap A_1 = [10]$$

so that the edges in $E_{0,1}$ are given by the following table:

$E_{0,1}$

u	init (u)	fin (u)	$\lambda(u)$
12	–	12	[10]
00	$\emptyset$	00	0
01	$\emptyset$	01	0
10	$\emptyset$	10	1
11	$\emptyset$	11	1

In the table, the entry "12" in the "u" column indicates that $A_1 = \{1, 2\}$. The entry "–" in the "init (u)" column indicates that $B_0 = \emptyset$, i.e., no components are to be extracted from u to obtain init (u); the entry "12" in the "fin (u)" column indicates that $B_1 = \{1, 2\}$, and so that components 1 and 2 are to be extracted from u to obtain fin (u). Finally, the entry "[10]" in the "$\lambda(u)$" column is the value of $\tilde{g}_1$, i.e., the S-active components of the first column of G. Thus the entries in the "$\lambda(u)$" column are the values of the inner product $u \cdot [10]$.

Similarly, to construct the edge set $E_{1,2}$, we have

$$\begin{aligned} V_1 &= 2^{B_1} = 2^{\{1,2\}} = \{00, 01, 10, 11\} \\ V_2 &= 2^{B_2} = 2^{\{1,2\}} = \{00, 01, 10, 11\} \\ A_2 &= 2^{A_2} = 2^{\{1,2\}} = \{00, 01, 10, 11\} \end{aligned}$$

and

$$\tilde{g}_2 \cap A_2 = [11]$$

so that we have the following table describing $E_{1,2}$:

$E_{1,2}$

u	init (u)	fin (u)	$\lambda(u)$
12	12	12	[11]
00	00	00	0
01	01	01	1
10	10	10	1
11	11	11	0

For $E_{2,3}$ we have the following table:

$E_{2,3}$

u	init (u)	fin (u)	$\lambda(u)$
12	12	12	[01]
00	00	00	0
01	01	01	1
10	10	10	0
11	11	11	1

(From the second row of the table we see that $A_3 = \{1,2\}$, $B_2 = \{1,2\}$, $B_3 = \{1,2\}$, and $\tilde{g}_3 = [01]$.)

Continuing this way, we obtain the following tables for $E_{3,4}$, $E_{4,5}$, $E_{5,6}$, and $E_{6,7}$:

$E_{3,4}$

u	init (u)	fin (u)	$\lambda(u)$
12	12	12	[11]
00	00	00	0
01	01	01	1
10	10	10	1
11	11	11	0

$E_{4,5}$

u	init (u)	fin (u)	$\lambda(u)$
123	12	3	[101]
000	00	0	0
001	00	1	1
010	01	0	0
011	01	1	1
100	10	0	1
101	10	1	0
110	11	0	1
111	11	1	0

$E_{5,6}$

u	init (u)	fin (u)	$\lambda(u)$
3	3	3	[1]
0	0	0	0
1	1	1	1

$E_{6,7}$

u	init (u)	fin (u)	$\lambda(u)$
3	3	–	[1]
0	0	$\emptyset$	0
1	1	$\emptyset$	1

If we piece these tables together into a graphical representation of the trellis, we arrive at Fig. 8, which is seen to be identical to that in Fig. 4. Indeed, each of the trellises that we pulled out of the hat in Section III is in fact an (G,S)-trellis for the MSGM given in Example 7.1 an appropriately chosen row cover S. For Fig. 2, $S = \{[1,7],[1,7],[1,7]\}$; for Fig. 3, $S = \{[1,5],[2,4],[1,7]\}$; and for Fig. 5, $S = \{[1,5],[2,4],[5,7]\}$. Note that the row covers for Fig. 5 are in fact the row *spans* for G. The following theorem shows that, in general, the (G,S)-trellis always represents $\mathbb{C}$, and that the (G,S)-trellis is isomorphic to the BCJR trellis if and only if S is the set of row spans.

Theorem 7.4: If S is a row cover for the minimal-span generator matrix G, then the source–sink trellis paths in the (G,S) trellis are in one-to-one correspondence with the codewords of $\mathbb{C}$. The (G,S) trellis is isomorphic to the BCJR trellis if and only if S is the set of row spans for G.

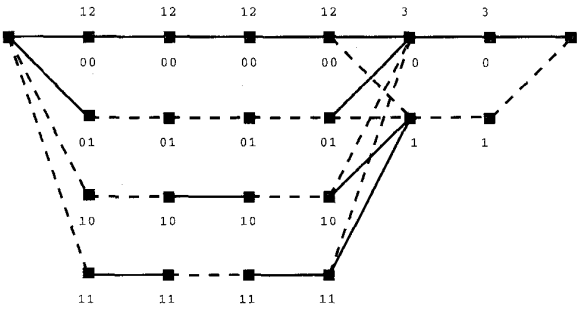

Fig. 8. The (G,S) trellis for the $(7,3,3)$ code described in Example 7.1. This trellis is identical to the one in Fig. 4.

Proof: It follows from the definition (7.3), that every codeword is produced by at least one source–sink path. On the other hand, by combining Lemma 5.4 and Theorem 7.2, we see that the total number of source–sink paths in the (G,S) trellis is

$$\prod_{i=0}^{n-1} \rho_i^+ = \prod_{i=0}^{n-1} 2^{\alpha_{i+1}-\beta_i} = 2^{\sum_{i=0}^{n-1}(\alpha_{i+1}-\beta_i)}. \tag{7.9}$$

But

$$\sum_{i=0}^{n-1}(\alpha_{i+1}-\beta_i) = \sum_{i=0}^{n-1} |A_{i+1} - A_{i+1} \cap A_i|.$$

Each index $j = 1, \cdots, k$ is an element of exactly one of the sets $A_{i+1} - A_{i+1} \cap A_i$, viz., the one corresponding to the smallest index i for which the the jth row is active, so that the sum in the exponent of (7.9) is k, and so the (G,S) trellis contains exactly 2^k source–sink paths. But we have already observed that there is at least one path that produces each of the 2^k codewords, so that the 2^k source–sink paths are in one-to-one correspondence with the codewords of $\mathbb{C}$.

To complete the proof, we note from Theorem 7.2 that the number of vertices at depth i is 2^{β_i}. But β_i is the number of rows which are active at depth i with respect to the row cover S. According to Corollary 6.16, the vertex dimension s_i of the BCJR trellis is the number of rows which are active with respect to the row spans of G. Thus since each row span is a subset of the corresponding row cover, we have $s_i \leq \beta_i$, with equality if and only if the set S is the set of row spans. It therefore follows from Theorem 5.1, that the (G,S) trellis is isomorphic to the BCJR trellis if and only if S is the set of row spans for G. □

Example 7.5: Let us conclude this section by constructing the BCJR trellis for the code that is dual to the $(7,3,3)$ code discussed elsewhere in this paper (Examples 3.1, 3.2, 4.7, 6.1, etc.). Any parity-check matrix for the original code will serve a generator matrix for the dual code, and so from Example 3.2, we take as a generator matrix

$$G = \begin{pmatrix} \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & \mathbf{0} & \mathbf{1} & 0 & 0 & 0 \\ \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{1} & 0 \\ \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} \end{pmatrix}. \tag{7.10}$$

This matrix is not in minimal-span form, but if we apply "Greedy Algorithm II" from Section III, we obtain the following MSGM for the code:

$$G' = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \begin{pmatrix} \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & \mathbf{0} & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} \end{pmatrix} \end{array}.$$

If we take as row covers the row spans, viz., $\{[1,3],[2,4],[3,6],[6,7]\}$, the the A_i's, B_i's, α_i's, and β_i's are as given in the following table:

i	A_i	B_i	α_i	β_i
0	–	$\emptyset$	–	0
1	$\{1\}$	$\{1\}$	1	1
2	$\{1,2\}$	$\{1,2\}$	2	2
3	$\{1,2,3\}$	$\{2,3\}$	3	2
4	$\{2,3\}$	$\{3\}$	2	1
5	$\{3\}$	$\{3\}$	1	1
6	$\{3,4\}$	$\{4\}$	2	1
7	$\{4\}$	$\emptyset$	1	0

Using the technique developed in Example 7.5, we then obtain the following sequence of tables, which describe the edge sets $E_{i-1,i}$, for $i = 1, \cdots, 7$:

$E_{0,1}$

u	init (u)	fin (u)	$\lambda(u)$
1	–	1	[1]
0	$\emptyset$	0	0
1	$\emptyset$	1	1

$E_{1,2}$

u	init (u)	fin (u)	$\lambda(u)$
12	1	12	[11]
00	0	00	0
01	0	01	1
10	1	10	1
11	1	11	0

$E_{2,3}$

u	init (u)	fin (u)	$\lambda(u)$
123	12	23	[101]
000	00	00	0
001	00	01	1
010	01	10	0
011	01	11	1
100	10	00	1
101	10	01	0
110	11	10	1
111	11	11	0

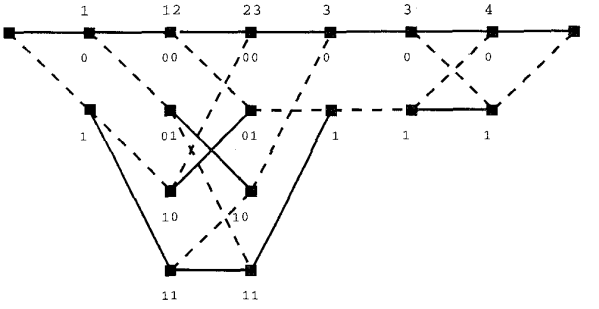

Fig. 9. The BCJR trellis for the $(7,4,2)$ code discussed in Example 7.5. (This trellis has $|E| = 26$, $|V| = 18$, and $|E| - |V| + 1 = 9$.)

$E_{3,4}$

u	init (u)	fin (u)	$\lambda(u)$
23	23	3	[11]
00	00	0	0
01	01	1	1
10	10	0	1
11	11	1	0

$E_{4,5}$

u	init (u)	fin (u)	$\lambda(u)$
3	3	3	[1]
0	0	0	0
1	1	1	1

$E_{5,6}$

u	init (u)	fin (u)	$\lambda(u)$
34	3	4	[11]
00	0	0	0
01	0	1	1
10	1	0	1
11	1	1	0

$E_{6,7}$

u	init (u)	fin (u)	$\lambda(u)$
4	4	–	[1]
0	0	$\emptyset$	0
1	1	$\emptyset$	1

If we piece these seven tables together into a graphical representation of the trellis, we obtain Fig. 9. Notice that the vertex structure of the trellis in Fig. 9 is identical to that in Figs. 5 and 7, as guaranteed by Corollary 4.4. The edge structures, however, are quite different. (There is, however, as close connection between the edge structures of the BCJR trellises for codes which are dual to each other, as explained in [20].)

VIII. Conclusion: The "Viterbi Decoding Complexity" of Linear Codes

Based on our thesis that the edge count is the right measure of the quality of a trellis representing a given (n, k) block code $\mathbb{C}$, we propose that the "Viterbi Decoding Complexity" (VDC) for $\mathbb{C}$ be defined as $|E|/k$, where $|E|$ is the number of edges in the BCJR trellis for $\mathbb{C}$. The (dimensionless) units of the VDC are *computations per decoded bit*. The VDC for block and convolutional codes can then be directly compared. For example, by a computation in [11], the VDC of the $(24, 12)$ Golay code is $3580/12 = 298.33$, whereas the VDC for the $(2, 1, 6)$ NASA standard convolutional code is 256. Of course, it may be possible to decrease the decoding complexity by making small or large modifications to the Viterbi algorithm, and a lot of research has been devoted to doing just this [5], [11], [27], [40]. But we feel that it is important to differentiate between the problem of minimizing the *combinatorial complexity* of the trellis representation of the code, and the problem of minimizing the *decoding complexity*. Minimizing the combinatorial complexity, i.e., finding the edge-minimal trellis representation of the code, is a well-defined problem, which we feel should be thought of as the necessary first step in minimizing the decoding complexity, which is much less well-defined.

Many authors have studied the problem of minimizing the "trellis complexity" of a given linear block code, allowing *column permutations*, beginning with Forney (as cited in [3]), Massey [29], and more recently in [8], [18], [19], [23], [40], and [42]. In this paper we have not allowed column permutations, and so we have nothing direct to contribute. We observe, however, that there is no guarantee that there is a column permutation that simultaneously minimizes the vertex, edge, state, and bifurcation complexities, so that in searching for the "optimum" column permutation, it is important to specify the figure of merit one is trying to optimize. Unfortunately, there appears to no general agreement as to what this figure of merit should be. Some authors take state complexity [35], [42], some take vertex complexity [18], [19], and some use the number of "addition-equivalent operations" [11], [40]. We believe that the results in this paper show that the most appropriate figure of merit is the *edge count* of the trellis, and we encourage future researchers in this area to take the edge count as the measure of trellis complexity. We envision a large table, in the spirit of Brouwer and Verhoeff [6], listing a number of good codes and the best known edge count for a trellis representing the code. Such a table would be an invaluable resource for researchers interested in finding good decoding algorithms for block codes. A start in this direction appears in [8] and [26].

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*. Reading, MA: Addison-Wesley, 1974.

[2] T. M. Apostol, *Calculus*, vol. II. New York: Wiley, 1969.

[3] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inform. Theory*, vol. IT-20, pp. 284–287, 1974.

[4] Y. Berger and Y. Be'ery, "Bounds on the trellis size of linear block codes," *IEEE Trans. Inform. Theory*, vol. 39, pp. 764–773, 1993.

[5] ——, "Soft trellis-based decoder for linear block codes," *IEEE Trans. Inform. Theory*, vol. 40, pp. 203–209, 1994.

[6] A. E. Brouwer and T. Verhoeff, "An updated table of minimum-distance bounds for binary linear codes," *IEEE Trans. Inform. Theory*, vol. 39, pp. 662–677, Mar. 1993.

[7] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. Cambridge, MA: MIT Press/McGraw-Hill, 1990.

[8] S. Dolinar, L. Ekroot, A. Kiely, R. McEliece, and W. Lin, "The permutation trellis complexity of linear block codes," in *Proc. 32nd Allerton Conf. on Communication, Control, and Computing*, Oct. 1994, pp. 60–74.

[9] Dornhoff and Hohn, *Applied Madern Algebra*. New York: Macmillan, 1978.

[10] G. D. Forney, Jr., "The Viterbi algorithm," *Proc. IEEE*, vol. 61, pp. 268–276, Mar. 1973.

[11] ——, "Coset codes—Part II: Binary lattices and related codes," *IEEE Trans. Inform. Theory*, vol. 34, pp. 1152–1187, Sept. 1988.

[12] ——, "On dimension/length profiles and trellis complexity of linear block codes," *IEEE Trans. Inform. Theory*, vol. 40, pp. 1741–1752, Nov. 1994.

[13] G. D. Forney, Jr., and M. D. Trott, "The dynamics of group codes: State spaces, trellis diagrams, and canonical encoders," *IEEE Trans. Inform. Theory*, vol. 39, pp. 1491–1513, Sept. 1993.

[14] M. Hall, Jr., *Combinatorial Theory*. Waltham, MA: Blaisdell, 1967.

[15] K. Hoffman and R. Kunze, *Linear Algebra*. Englewood Cliffs, N.J.: Prentice-Hall, 1961.

[16] B. Honary, G. Markarian, and P. Farrell, "Generalized array codes and their trellis structure," *Electron. Lett.*, vol. 29, pp. 541–542, 1993.

[17] R. A. Horn and C. R. Johnson, *Matrix Analysis*. Cambridge, UK: Cambridge Univ. Press, 1985.

[18] T. Kasami, T. Takata, T. Fuijwara, and S. Lin, "On the optimum bit orders with respect to the state complexity of trellis diagrams for binary linear codes," *IEEE Trans. Inform. Theory*, vol. 39, pp. 242–245, 1993.

[19] ——, "On the complexity of trellis structure of linear block codes," *IEEE Trans. Inform. Theory*, vol. 39, pp. 1057–1064, 1993.

[20] A. Kiely, S. Dolinar, R. McEliece, L. Ekroot, and W. Lin, "Trellis decoding complexity of linear block codes," to be published in *IEEE Trans. Inform. Theory*, vol. 42, 1996.

[21] A. D. Kot and C. Leung, "On the construction and dimensionality of linear block code trellises," in *Proc. 1993 ISIT*, p. 291.

[22] F. R. Kschischang and V. Sorokine, "On the trellis structure of block codes," *IEEE Trans. Inform. Theory*, vol. 41, pp. 1924–1937, Nov. 1995.

[23] F. R. Kschischang and G. B. Horn, "A heuristic for ordering a linear block code to minimize trellis state complexity," in *Proc. 32nd Allerton Conf. on Communication, Control, and Computing*, Oct. 1994, pp. 75–84.

[24] F. Kschischang and A. Vardy, "Proof of a conjecture of McEliece regarding the optimality of the minimal trellis," *IEEE Trans. Inform. Theory*, submitted for publication.

[25] A. Lafourcade and A. Vardy, "Asymptotically good codes have infinite trellis complexity," to be published in *IEEE Trans. Inform. Theory*, vol. IT-42, 1996.

[26] ——, "Lower bounds on trellis complexity of block codes," to be published in *IEEE Trans. Inform. Theory*, vol. 42, 1996.

[27] ——, "Optimal sectionalization of a trellis," *IEEE Trans. Inform. Theory*, vol. 42, pp. 689–703, May 1996.

[28] J. H. van Lint and R. M. Wilson, *A Course in Combinatorics*. Cambridge, UK: Cambridge Univ. Press, 1992.

[29] J. L. Massey, "Foundations and methods of channel coding" *NTG-Fachberichte* (Proc. Int. Conf. on Information Theory and Systems), vol. 65, pp. 148–157, 1978.

[30] R. J. McEliece, *The Theory of Information and Coding*. Reading, MA: Addison-Wesley, 1977.

[31] R. J. McEliece, R. B. Ash, and C. Ash, *Introduction to Discrete Mathematics*. New York: Random House, 1989.

[32] R. J. McEliece, "The Viterbi decoding complexity of linear block codes," in *Proc. Int. Symp. on Information Theory* (Trondheim, Norway, June 1994), p. 341.

[33] R. J. McEliece and W. Lin, "The trellis complexity of convolutional codes," to be published in *Proc. 3rd Int. Symp. on Communication Theory and Applications* (Ambleside, UK, July 1995).

[34] H. T. Moorthy and S. Lin, "Good nonminimal trellises for linear block codes," submitted to *IEEE Trans. Commun.*

[35] D. J. Muder, "Minimal trellises for block codes," *IEEE Trans. Inform. Theory*, vol. 34, pp. 1049–1053, Sept. 1988.

[36] A. V. Oppenheim, A. S. Willsky, and I. T. Young, *Signals and Systems*. Englewood Cliffs, N.J.: Prentice-Hall, 1983.

[37] G. Solomon, L. H. Oswald, and M. Indurain, "An inefficient trellis representing the words in a block code," *Math. Rejecta*, in press.

[38] V. V. Zyablov and V. R. Siderenko, "Decoding of convolutional codes using a syndrome trellis," *IEEE Trans. Inform. Theory*, vol. 40, pp. 1663–1666, Sept. 1994.

[39] R. P. Stanley, *Enumerative Combinatorics*, vol. I. Monterey, CA: Wadsworth and Brooks/Cole, 1986.

[40] A. Vardy and Y. Be'ery, "Maximum-likelihood soft decision decoding of BCH codes," *IEEE Trans. Inform. Theory*, vol. 40, pp. 546–554, Mar. 1994.

[41] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. Inform. Theory*, vol. IT-13, pp. 260–269, Apr. 1967.

[42] Y.-Y. Wang and C.-C. Lu, "The trellis complexity of equivalent binary [17, 9] quadratic residue code is five," in *Proc. 1993 Int. Symp. on Information Theory*, p. 200.

[43] J. K. Wolf, "Efficient maximum likelihood decoding of linear block codes," *IEEE Trans. Inform. Theory*, vol. IT-24, pp. 76–80, Jan. 1978.

[44] V. V. Zyablov and V. R. Siderenko, "Bounds on complexity of trellis decoding," *Probl. Pered. Inform.*, vol. 29, no. 3, pp. 1–6, July–Sept. 1993.