A recursive SISO decoding algorithm

Liudmila Karakchieva, Peter Trifonov
ITMO University, Russia
Email: {lvkarakchieva, pvtrifonov}@itmo.ru

***Abstract*—A novel SISO decoding algorithm for linear block codes is presented. This algorithm is based on the recursive trellises and performs two passes over the recursion tree. Probability-domain implementation and its LogMax approximation are considered. Numeric results show that proposed method has lower complexity compared to the other known recursive algorithms and the classical BCJR algorithm.**

I. Introduction

Soft-input soft-output (SISO) decoders for linear block codes are building blocks of various iterative decoding and equalization methods. The best known SISO decoding method is the BCJR algorithm [1]. However, its complexity is quite high, except for very simple codes. The BCJR algorithm operates over the minimal trellis of the code and makes one forward and one backward pass. Another optimal SISO decoding algorithm based only on one forward recursion was presented in [2]. However, this method is numerically unstable and cannot exploit simplified trellises [3], [4]. Recursive trellises were introduced in [5], and shown to enable decoding of linear codes with much lower complexity compared to the Viterbi algorithm even with optimal sectionalization [6].

In this paper we introduce a SISO decoding algorithm for linear block codes based on recursive trellises. We present both an optimal probability-domain implementation, and its LogMax approximation. We show that the proposed approach has lower complexity compared to the BCJR algorithm and the algorithms given in [7], [8], which are also based on the recursive trellises. An important advantage of our approach under uniform sectionalization is much lower decoding latency compared to the BCJR algorithm.

II. Background

A. Symbolwise MAP decoding

Let us consider transmission of binary data over a binary input memoryless channel with output $R_0^{n-1} = (R_0, R_1, ..., R_{n-1})$. Let $\mathcal{C}$ be an (n,k) binary linear block code. The set of codewords $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1})$ with bit $b \in \mathbb{F}_2$ on the position i is denoted by $\mathcal{C}^{i/b} = \{\mathbf{c} \in \mathcal{C} : c_i = b\}$. Symbolwise MAP decoder computes a posteriori probabilities $P(c_i = b|R_0^{n-1}), b \in \mathbb{F}_2$. Since the channel is assumed to be memoryless and codewords are equiprobable, we have

$$P(c_i = b|R_0^{n-1}) = \sum_{\mathbf{c}\in\mathcal{C}^{i/b}} P(\mathbf{c}|R_0^{n-1}) = \frac{\sum_{\mathbf{c}\in\mathcal{C}^{i/b}} P(\mathbf{c}, R_0^{n-1})}{P(R_0^{n-1})}$$
$$= \frac{\sum_{\mathbf{c}\in\mathcal{C}^{i/b}} P(\mathbf{c}) \prod_{0\le j<n} P(R_j|c_j)}{P(R_0^{n-1})} = K \sum_{\mathbf{c}\in\mathcal{C}^{i/b}} \prod_{0\le j<n} P(R_j|c_j)$$

for each bit position $0 \le i < n$, where K is a constant. Multiplying by K can be omitted since it cancels while computing log-likelihood ratios $\Lambda_i = \log \frac{P(c_i=0|R_0^{n-1})}{P(c_i=1|R_0^{n-1})}$.

B. Recursive maximum likelihood decoding

A low complexity recursive maximum-likelihood decoding (RMLD) algorithm was introduced in [5]. It relies on recursive trellis representation of linear block codes. Here we present a brief summary of this approach using the notation introduced in [9]. Let $[x,y) = \{i | x \le i < y\}$ denote a section. Given a linear code $\mathcal{C}$, we can define its subcode $\mathcal{C}_{x,y}$, such that all its codewords have non-zero symbols only in the range $x \le i < y$. Let $p_{x,y}(\mathcal{C})$ be a linear code obtained by puncturing all symbols from codewords of $\mathcal{C}$, except those in positions $x \le i < y$. Let $s_{x,y}(\mathcal{C}) = p_{x,y}(\mathcal{C}_{x,y})$ be a code obtained from $\mathcal{C}$ by shortening it on all symbols except those in range $x \le i < y$. Trellis paths from time x to time y correspond to cosets in factorgroup $L_{x,y} = p_{x,y}(\mathcal{C})/s_{x,y}(\mathcal{C})$. It is possible to construct generator matrices of $p_{x,y}(\mathcal{C})$ and $s_{x,y}(\mathcal{C})$ in the form

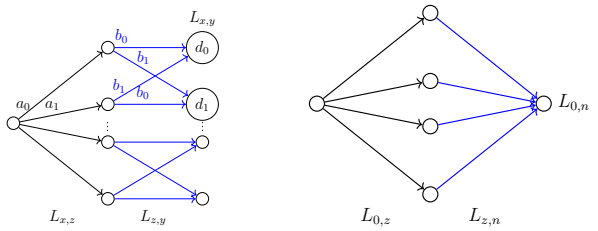
$$G_{x,y}^{(p)} = \left(\begin{array}{cc} G_{x,z}^{(s)} & 0 \\ 0 & G_{z,y}^{(s)} \\ G_{x,y}^{(00)} & G_{x,y}^{(01)} \\ \hline G_{x,y}^{(10)} & G_{x,y}^{(11)} \end{array}\right) \text{ and } G_{x,y}^{(s)} = \begin{pmatrix} G_{x,z}^{(s)} & 0 \\ 0 & G_{z,y}^{(s)} \\ G_{x,y}^{(00)} & G_{x,y}^{(01)} \end{pmatrix},$$

respectively.

Let $G_{x,y}^{(00)}, G_{x,y}^{(01)}$ be some $k''_{x,y} \times (z-x)$ and $k''_{x,y} \times (y-z)$ matrices and $G_{x,y}^{(10)}, G_{x,y}^{(11)}$ be some $k'_{x,y} \times (z-x)$ and $k'_{x,y} \times (y-z)$ matrices, respectively. Let $G_{x,y}^{(s)}$ be the $k'''_{x,y} \times (y-x)$ matrix. Therefore, $L_{x,y}$ contains $|L_{x,y}| = 2^{k'_{x,y}}$ cosets, and each coset $D \in L_{x,y}$ has $2^{k'''_{x,y}}$ elements.

Let the *composite branch table* $CBT_{x,y}$ be a data structure containing some scores $CBT_{x,y}(D).m = m(D)$ for the cosets $D \in L_{x,y}$. The score can be defined as probability, correlation discrepancy or correlation of the most likely element $(c_x, \ldots, c_{y-1}) \in D$. Here we assume for the sake of concreteness that $m(D) = \prod_{x\le j<y} P(R_j|c_j)$. The original RMLD algorithm requires also storing $CBT_{x,y}(D).l = c_x^{y-1}$, the most probable coset element. However, it is not used in our paper. The composite branch table can be constructed either by straightforward enumeration of elements of $p_{x,y}(\mathcal{C})$, or recursively, as discussed below. Let us consider two adjacent sections $[x,z)$ and $[z,y)$. To fill $CBT_{x,y}$ we can combine $CBT_{x,z}$ and $CBT_{z,y}$. In this way, we need to investigate two-section trellis as shown in Fig. 1a. Since there is an one-to-one correspondence between $vG'_{x,y} = v\begin{pmatrix} G_{x,y}^{(10)} & G_{x,y}^{(11)} \end{pmatrix}$, and

978-1-6654-0312-2/21/$31.00 ©2021 IEEE



(a) Combining two sections (b) Top level two-section trellis

Fig. 1: Two-section trellises

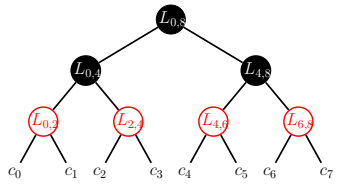


Fig. 2: Tree-like sectionalization for a code of length 8

(a) $[2,4)$ (b) $\overline{[2,4)}$

Fig. 3: Section and its complement

cosets $D \in L_{x,y}, v \in \mathbb{F}_2^{k'_{x,y}}$, we can index each coset in $L_{x,y}$ by vector v. To combine two adjacent sections we need to compute

$$CBT_{x,y}[v].m = \max_{u \in \mathbb{F}_2^{k''_{x,y}}} (CBT_{x,z}[a].m \cdot CBT_{z,y}[b].m),$$

where $v \in \mathbb{F}_2^{k'_{x,y}}$, $a \in \mathbb{F}_2^{k'_{x,z}}$ and $b \in \mathbb{F}_2^{k'_{z,y}}$ are indices of the cosets $D' \in L_{x,z}$ and $D'' \in L_{z,y}$, respectively (see Fig. 1a). To obtain these indices we need to solve the following system of equations:

$$\begin{pmatrix} a' & a \end{pmatrix} \begin{pmatrix} G^{(s)}_{x,z} \\ G'_{x,z} \end{pmatrix} = \begin{pmatrix} u & v \end{pmatrix} \begin{pmatrix} G^{(00)}_{x,y} \\ G^{(10)}_{x,y} \end{pmatrix}$$
$$\begin{pmatrix} b' & b \end{pmatrix} \begin{pmatrix} G^{(s)}_{z,y} \\ G'_{z,y} \end{pmatrix} = \begin{pmatrix} u & v \end{pmatrix} \begin{pmatrix} G^{(01)}_{x,y} \\ G^{(11)}_{x,y} \end{pmatrix}.$$

Here $a' \in \mathbb{F}_2^{k'''_{x,z}}$ and $b' \in \mathbb{F}_2^{k'''_{z,y}}$ are some vectors which are not used. The solution of this system can be represented as

$$a = \begin{pmatrix} u & v \end{pmatrix} \widehat{G}_{x,y}, b = \begin{pmatrix} u & v \end{pmatrix} \widetilde{G}_{x,y}.$$

Here $\widehat{G}_{x,y}$ and $\widetilde{G}_{x,y}$ are the matrices, which can be constructed offline. This approach is more efficient [5] compared to the Viterbi algorithm with optimal sectionalization [6], since it enables more aggressive reuse of common subexpressions.

III. Proposed algorithm

A. Probability-domain decoding

Let R_j be channel output symbols, and consider computing

$$P(c_i = b|R_0^{n-1}) \sim \sum_{\mathbf{c} \in \mathcal{C}^{i/b}} \prod_{j=0}^{n-1} P(R_j|c_j) \triangleq \pi(c_i = b|R_0^{n-1}). \quad (1)$$

Each node in the recursive trellis decomposition tree of code $\mathcal{C}$ (see Fig. 2) corresponds to some section $[x,y)$. Leaves, shown in red, represent sections, which are not further partitioned.

Consider the node labeled by $L_{x,y}$ and assume that it is not a leaf. For each coset D in $L_{x,y}$ we define

$$A^{(D)}_{x,y} = \sum_{c_x^{y-1} \in D} P(R_x^{y-1}|c_x^{y-1}) = \sum_{c_x^{y-1} \in D} \prod_{x \le j < y} P(R_j|c_j), \quad (2)$$

which is proportional to the probability of receiving vector R_x^{y-1}, provided that the elements of $D \in L_{x,y}$ were transmitted. This probability can be computed either straightforwardly, or recursively. Consider some $z : x < z < y-1$. It can be seen that

$$A^{(D)}_{x,y} = \sum_{\substack{D',D'' \\ D' \bullet D'' = D}} \sum_{c_x^{z-1} \in D'} \sum_{c_z^{y-1} \in D''} P(R_x^{z-1}|c_x^{z-1}) P(R_z^{y-1}|c_z^{y-1})$$
$$= \sum_{\substack{D',D'' \\ D' \bullet D'' = D}} A^{(D')}_{x,z} A^{(D'')}_{z,y}, \quad (3)$$

where $D' \in L_{x,z}, D'' \in L_{z,y}, D \in L_{x,y}$, and $D' \bullet D''$ denotes the set containing vectors from D' concatenated with vectors from D''.

Given section $[x,y)$ we define its complement as $\overline{[x,y)} = [0,x) \cup [y,n)$, as shown in Fig 3. Consider a coset D in $L_{x,y}$. Let us further define

$$B^{(D)}_{x,y} = \sum_{\substack{c_0^{x-1}, c_y^{n-1} \\ c_x^{y-1} \in D, \mathbf{c} \in \mathcal{C}}} P(R_0^{x-1}, R_y^{n-1}|c_0^{x-1}, c_y^{n-1})$$
$$= \sum_{\substack{c_0^{x-1}, c_y^{n-1} \\ c_x^{y-1} \in D, \mathbf{c} \in \mathcal{C}}} P(R_0^{x-1}|c_0^{x-1}) P(R_y^{n-1}|c_y^{n-1}),$$

which is proportional to the probability of obtaining R_0^{x-1}, R_y^{n-1}, provided that the fragments c_0^{x-1}, c_y^{n-1} of codewords in $\mathcal{C}$ were transmitted, such that $c_x^{y-1} \in D$, where $D \in L_{x,y}$. In what follows, we assume that $c_0^{n-1} \in \mathcal{C}, c_0^{n-1} = \mathbf{c}$. It can be seen that at the top level of the recursion tree

$$B^{(D')}_{0,z} = \sum_{c_z^{n-1} \in D''} P(R_z^{n-1}|c_z^{n-1}) = A^{(D'')}_{z,n}, \quad (4)$$

$$B^{(D'')}_{z,n} = \sum_{c_0^{z-1} \in D'} P(R_0^{z-1}|c_0^{z-1}) = A^{(D')}_{0,z}. \quad (5)$$

The two-section trellis for this case is depicted in Fig. 1b.

For an intermediate level (see Fig. 1a), one obtains

$$B^{(D')}_{x,z} = \sum_{\substack{c_0^{x-1}, c_z^{y-1} \in D'', c_y^{n-1} \\ c_x^{z-1} \in D', D = D' \bullet D''}} P(R_0^{x-1}, R_z^{y-1}, R_y^{n-1}|c_0^{x-1}, c_z^{y-1}, c_y^{n-1})$$
$$= \sum_D \sum_{\substack{c_0^{x-1}, c_y^{n-1} \\ c_x^{y-1} \in D}} \sum_{c_z^{y-1} \in D''} P(R_0^{x-1}, R_y^{n-1}|c_0^{x-1}, c_y^{n-1}) P(R_z^{y-1}|c_z^{y-1})$$

$$= \sum_{\substack{D \\ D' \bullet D'' = D}} B_{x,y}^{(D)} A_{z,y}^{(D'')} \quad (6)$$

and

$$B_{z,y}^{(D'')} = \sum_{\substack{D \\ D' \bullet D'' = D}} B_{x,y}^{(D)} A_{x,z}^{(D')}. \quad (7)$$

Given some $D \in L_{x,y}$, we define

$$D^{i/b} = \{c_x^{y-1} = (c_x, ..., c_{i-1}, b, c_{i+1}, ..., c_{y-1}) | c_0^{y-1} \in D\},$$

where $x \le i < y$. Let $L_{x,y}^{i/b} = \{D^{i/b} | D \in L_{x,y}\}$. One can rewrite (1) as

$$\pi(c_i = b|R_0^{n-1}) = \sum_{\mathbf{c} \in \mathcal{C}^{i/b}} P(R_0^{n-1}|\mathbf{c})$$
$$= \sum_{\mathbf{c} \in \mathcal{C}^{i/b}} P(R_0^{x-1}, R_y^{n-1}|c_0^{x-1}, c_y^{n-1}) P(R_x^{y-1}|c_x^{y-1}). \quad (8)$$

We partition the cosets D into the subsets, such that their elements contain the same value $b \in \mathbb{F}_2$ in position i. Therefore, one can rewrite (8) as

$$\pi(c_i = b|R_0^{n-1}) = \sum_{D^{i/b} \in L_{x,y}^{i/b}} \sum_{\substack{c_0^{x-1}, c_y^{n-1} \\ c_x^{y-1} \in D^{i/b}}} P(R_0^{x-1}, R_y^{n-1}|c_0^{x-1}, c_y^{n-1})$$
$$\cdot \sum_{c_x^{y-1} \in D^{i/b}} P(R_x^{y-1}|c_x^{y-1}) = \sum_{D \in L_{x,y}} B_{x,y}^{(D)} A_{x,y}^{(D,i/b)}, \quad (9)$$

where

$$A_{x,y}^{(D,i/b)} = \sum_{c_x^{y-1} \in D^{i/b}} P(R_x^{y-1}|c_x^{y-1}). \quad (10)$$

Hence SISO decoding, i.e. computation of (9), can be implemented by the following two-pass recursive process.

Upward recursion:

(a) Consider the shortest sections $[x, y)$ of the recursion tree, and compute probabilities for the cosets $D \in L_{x,y}$ by (2) and for their subsets $D^{i/b}$ by (10).

(b) Recursively combine adjacent sections by (3), until we reach the level preceding the top of the tree.

Exchange step:

(c) Initialize $B_{0,z}^{(D')} = A_{z,n}^{(D'')}, B_{z,n}^{(D'')} = A_{0,z}^{(D')}$, where $D' \bullet D'' = \mathcal{C}, D' \in L_{0,z}, D'' \in L_{z,n}$.

Downward recursion:

(d) Recursively compute probabilities $B_{x,z}^{(D')}, B_{z,y}^{(D'')}$ for subsections using (6) and (7).

(e) Compute the symbol a-posteriori probabilities by (9).

This approach is similar to the BCJR algorithm, which involves forward and backward passes.

B. Log-Max approximation

Unfortunately, the above described probability-domain implementation requires costly multiplications and is prone to numeric errors. To avoid these problems, we propose an LLR-domain approximation. Consider computing

$$\Lambda_i = \log \frac{P(c_i = 0|R_0^{n-1})}{P(c_i = 1|R_0^{n-1})} = \log \frac{\sum_{\mathbf{c} \in \mathcal{C}^{i/0}} P(\mathbf{c}|R_0^{n-1})}{\sum_{\mathbf{c} \in \mathcal{C}^{i/1}} P(\mathbf{c}|R_0^{n-1})}. \quad (11)$$

Let us replace summations in (11) with maximization. We obtain

$$\Lambda_i \approx \log \frac{\max_{\mathbf{c} \in \mathcal{C}^{i/0}} P(\mathbf{c}|R_0^{n-1})}{\max_{\mathbf{c} \in \mathcal{C}^{i/1}} P(\mathbf{c}|R_0^{n-1})}$$
$$= \max_{\mathbf{c} \in \mathcal{C}^{i/0}} \sum_{j=0}^{n-1} \log P(R_j|c_j) - \max_{\mathbf{c} \in \mathcal{C}^{i/1}} \sum_{j=0}^{n-1} \log P(R_j|c_j). \quad (12)$$

Let us subtract $\log P(R_j|\hat{c}_j)$ from each term in (12), where $\hat{c}$ is a vector of hard decisions. Therefore, we need to compute

$$S^{(i)}(b) = \max_{\mathbf{c} \in \mathcal{C}^{i/b}} \underbrace{\sum_{j=0}^{n-1} \Big(\log P(R_j|c_j) - \log P(R_j|\hat{c}_j) \Big)}_{E(R_0^{n-1}, \mathbf{c})}$$

to obtain $\Lambda_i \approx S^{(i)}(0) - S^{(i)}(1)$. The value $E(R_x^{y-1}, c_x^{y-1})$ denotes correlation discrepancy, which is given by

$$E(R_x^{y-1}, c_x^{y-1}) = \sum_{x \le i < y} \tau(S_i, c_i), S_i = \log \frac{P(R_i|c_i = 0)}{P(R_i|c_i = 1)},$$

where $\tau(S, c) = \begin{cases} -|S|, & (-1)^c \cdot S < 0 \\ 0, & \text{otherwise.} \end{cases}$

Thus, instead of calculation of probabilities $P(c_i = b|R_0^{n-1})$, we need to compute the correlation discrepancy $S^{(i)}(b)$ of the most probable codeword in $\mathcal{C}^{i/b}$. This enables one to replace multiplication and summation operations in the above expressions with summation and comparison operations, respectively. Thus, the proposed algorithm can be implemented similarly to the algorithm described in section III-A with

$$\widetilde{A}_{x,y}^{(D)} = \max_{c_x^{y-1} \in D} E(R_x^{y-1}, c_x^{y-1})$$

and

$$\widetilde{B}_{x,y}^{(D)} = \max_{\substack{c_0^{x-1}, c_y^{n-1} \\ c_x^{y-1} \in D, \mathbf{c} \in \mathcal{C}}} \Big(E(R_0^{x-1}, c_0^{x-1}) + E(R_y^{n-1}, c_y^{n-1}) \Big).$$

C. Processing of short sections

In this section we present the algorithm to be used at the recursion boundary. Most of these techniques are based on [6].

Let us consider the section $[x, y)$ and define

$$M_b[i] = \begin{cases} \sum_{\mathbf{c} \in \mathcal{C}^{i/b}} \prod_{0 \le j < n} P(R_j|c_j), & \text{probability-domain decoding} \\ \max_{\mathbf{c} \in \mathcal{C}^{i/b}} E(R_0^{n-1}, \mathbf{c}), & \text{LLR-domain decoding.} \end{cases}$$

The LLRs are obtained as

$$\Lambda_i = \begin{cases} \log \frac{M_0[i]}{M_1[i]}, & \text{probability-domain decoding} \\ M_0[i] - M_1[i], & \text{LLR-domain decoding.} \end{cases}$$

We can reduce the complexity of making composite branch tables in the following cases:

1) $(y - x) = 1$

It can be seen that there is no need to perform any operations for forward pass. For backward pass we need to perform only 2 operations to compute $M_b[x], b \in \mathbb{F}_2$:

$$M_b[x] = \begin{cases} B_{x,y}^{(b)} \cdot A_{x,y}^{(b,x/b)}, & \text{probability-domain decoding} \\ \widetilde{B}_{x,y}^{(b)} + \widetilde{A}_{x,y}^{(b,x/b)}, & \text{LLR-domain decoding.} \end{cases}$$

2) $(y-x) = 2 \wedge (k'_{x,y} + k''_{x,y}) = 2$
Such section corresponds to either $L'_{x,y} = \{\{00\},\{10\},\{01\},\{11\}\}$ (i.e. $k''_{x,y} = 2$) or $L''_{x,y} = \{\{00,11\},\{10,01\}\}$ (i.e. $k''_{x,y} = 1$). Consider LLR-domain decoding. It requires to compute

$$M^{(b_0,b_1)} = \tau(S_x, b_0) + \tau(S_{x+1}, b_1), \tag{13}$$

where $b_0, b_1 \in \mathbb{F}_2, S_i = \log\frac{P(R_i|c_i=0)}{P(R_i|c_i=1)}$. We can reduce the number of operations by subtracting $\tau(S_x,1)+\tau(S_{x+1},1)$ from each $M^{(b_0,b_1)}$ in (13). This results in

$$\begin{aligned} M^{(00)} &= S_x + S_{x+1}, \quad & M^{(10)} &= S_{x+1}, \\ M^{(01)} &= S_x, & M^{(11)} &= 0. \end{aligned} \tag{14}$$

It takes only 1 addition for the case of $L'_{x,y}$. In the case of $L''_{x,y}$, we need to perform one more comparison to find $\max\{M^{(10)}, M^{(01)}\}$. Moreover, when $k''_{x,y} \neq 0$, we should save $\widetilde{A}^{(D,i/b)}_{x,y} = \max_{c_x^{y-1} \in D^{i/b}} E(R_x^{y-1}, c_x^{y-1})$. These values are needed to compute $S^{(i)}(b)$.
In the case of probability-domain decoding one obtains

$$\begin{aligned} M^{(00)} &= P(R_x|c_x=0)\cdot P(R_{x+1}|c_{x+1}=0) \\ M^{(10)} &= P(R_{x+1}|c_{x+1}=0) - M^{(00)} \\ M^{(01)} &= P(R_{x+1}|c_x=0) - M^{(00)} \\ M^{(11)} &= P(R_{x+1}|c_{x+1}=1) - M^{(01)}. \end{aligned} \tag{15}$$

It can be seen that instead of 4 multiplications, we can apply 1 multiplication and 3 summations. Similarly to the LLR-domain approach, we need to save $A^{(D,i/b)}_{x,y} = \sum_{c_x^{y-1}\in D^{i/b}} \prod_{x\le j<y} P(R_j|c_j)$ and perform two more summations to obtain $M^{(00)}+M^{(11)}$ and $M^{(10)}+M^{(01)}$ in the case of $L''_{x,y}$.

3) $(y-x) = k'_{x,y}$
Consider LLR-domain decoding. We need to compute $E(R_x^{y-1}, c_x^{y-1})$ for each $c_x^{y-1} \in \mathbb{F}_2^{y-x}$. These values can be computed in a tree-like way. This process takes only $2^{k'_{x,y}} - k'_{x,y} - 1$ summations.
For the probability-domain decoding one can divide all $A^{(D)}_{x,y} = \prod_{x\le j<y} P(R_j|c_j)$ values, where c_x^{y-1} is the only element in coset D, by $\prod_{x\le j<y} P(R_j|c_j=0)$. Then, one can compute the modified $A^{(D)}_{x,y}$ values in the same tree-like way. It takes only $2^{k'_{x,y}} - k'_{x,y} - 1$ multiplications.

4) $\begin{pmatrix} G^{(00)}_{x,y} & G^{(01)}_{x,y} \end{pmatrix} = \mathbf{1}$.
Fig. 1a illustrates two-section trellis for this case. For upward pass of probability-domain decoding expression (3) reduces to several instances of 2-point cyclic convolution $d_0 = a_0b_0 + a_1b_1, d_1 = a_1b_0 + a_0b_1$.
This can be computed as

$$\begin{aligned} d_0 &= \frac{(a_0+a_1)(b_0+b_1) + (a_0-a_1)(b_0-b_1)}{2} \\ d_1 &= \frac{(a_0+a_1)(b_0+b_1) - (a_0-a_1)(b_0-b_1)}{2}. \end{aligned}$$

It takes only 2 multiplications to obtain d_0 and d_1. Division by 2 can be omitted since it cancels while computing the LLR.

TABLE I: Complexity functions for the proposed method

Upward direction		
Function	Probability-domain	LLR-domain
$\phi_U^{(c)}(x,y)$	$\begin{cases} 0, & (y-x)=n \\ 2\cdot 2^{k'_{x,y}-1}, & \begin{pmatrix} G^{(00)}_{x,y} & G^{(01)}_{x,y} \end{pmatrix} = \mathbf{1} \\ 2^{k'_{x,y}+k''_{x,y}}, & \text{otherwise} \end{cases}$	$\begin{cases} 0, & (y-x)=n \\ 2\cdot 2^{k'_{x,y}+k''_{x,y}} - 2^{k'_{x,y}}, & \text{otherwise} \end{cases}$
$\phi_U^{(l)}(x,y)$	$\begin{cases} 0, & (y-x)=1 \\ 1, & \begin{matrix}(y-x)=2\wedge(y-x)*k'_{x,y} \\ \wedge(k'_{x,y}+k''_{x,y})=2\end{matrix} \\ 2^{k'_{x,y}} - k'_{x,y} - 1, & (y-x)=k'_{x,y} \\ (2^{k'_{x,y}+k''_{x,y}})(y-x-1), & \text{otherwise} \end{cases}$	$\begin{cases} 0, & (y-x)=1 \\ 2, & \begin{matrix}(y-x)=2\wedge(y-x)*k'_{x,y} \\ \wedge(k'_{x,y}+k''_{x,y})=2\end{matrix} \\ 2^{k'_{x,y}} - k'_{x,y} - 1, & (y-x)=k'_{x,y} \\ (2^{k'_{x,y}+k''_{x,y}})(y-x), & \text{otherwise} \end{cases}$
Downward direction		
Function	Probability-domain	LLR-domain
$\phi_D^{(c)}(x,y)$	$\begin{cases} 0, & (y-x)=n \\ 4\cdot 2^{k'_{x,y}-1}, & \begin{pmatrix} G^{(00)}_{x,y} & G^{(01)}_{x,y} \end{pmatrix} = \mathbf{1} \\ 2\cdot 2^{k'_{x,y}+k''_{x,y}}, & \text{otherwise} \end{cases}$	$\begin{cases} 0, & (y-x)=n \\ 4\cdot 2^{k'_{x,y}+k''_{x,y}} - 2^{k'_{x,z}} - 2^{k'_{z,y}}, & \text{otherwise} \end{cases}$
$\phi_D^{(l)}(x,y)$	$\begin{cases} 2, & (y-x)=1 \\ 4, & \begin{matrix}(y-x)=2\wedge(y-x)*k_{x,y} \\ \wedge(k'_{x,y}+k''_{x,y})=2\end{matrix} \\ 2^{k'_{x,y}}, & (y-x)=k'_{x,y} \\ 2\cdot 2^{k'_{x,y}}(y-x), & \text{otherwise} \end{cases}$	$\begin{cases} 2, & (y-x)=1 \\ 2^{k'_{x,y}} + k'_{x,y}(2^{k'_{x,y}}-2), & (y-x)=k'_{x,y} \\ 8, & \begin{matrix}(y-x)=2\wedge(y-x)*k'_{x,y} \\ \wedge(k'_{x,y}+k''_{x,y})=2\end{matrix} \\ 4\cdot 2^{k'_{x,y}}(y-x), & \text{otherwise} \end{cases}$

For downward pass one can compute

$$\begin{aligned} a'_0 &= b_0d_0 + b_1d_1, \quad & a'_1 &= b_1d_0 + b_0d_1 \\ b'_0 &= a_0d_0 + a_1d_1, \quad & b'_1 &= a_1d_0 + a_0d_1 \end{aligned}$$

in a similar way. It takes only 4 multiplications.

IV. Complexity and Optimum Sectionalization

In this section we present expressions for the complexity of the proposed algorithm. For probability-domain decoding, we report the number of multiplications and do not take into account any summations, since multiplication operation is more complicated.

Let $\phi_U^{(c)}(x,y)$ and $\phi_D^{(c)}(x,y)$ denote the number of operations required for non-leaf nodes of upward and downward recursion trees, respectively. Let $\phi_U^{(l)}(x,y)$ and $\phi_D^{(l)}(x,y)$ denote the number of operations required for initialization and for computation of target $P(c_i = b|R_0^{n-1})$, $x \le i < y$, respectively (leaf nodes). Tab. I presents the complexity of these operations.

The complexity of the proposed decoding algorithm depends on sectionalization, i.e. a rule for partitioning section $[x,y)$ into subsections $[x,z)$ and $[z,y)$. We need to find the sectionalization with the smallest overall decoding complexity, i.e. the optimum sectionalization for the code.

Let $\phi^{min}(x,y)$ be the minimum number of operations required to make table $CBT_{x,y}$. It is given by

$$\phi^{min}(x,y) = \begin{cases} \phi^{(l)}(x,y), & \text{if techniques of section III-C are applicable} \\ \min\Big\{\phi^{(l)}(x,y), \min_{x<z<y-1}\{\phi^{(c)}(x,y) \\ \quad + \phi^{min}(x,z) + \phi^{min}(z,y)\}\Big\}, & \text{otherwise,} \end{cases}$$

where $\phi^{(c)}(x,y) = \phi_U^{(c)}(x,y) + \phi_D^{(c)}(x,y)$ and $\phi^{(l)}(x,y) = \phi_U^{(l)}(x,y) + \phi_D^{(l)}(x,y)$. Therefore, the total number of operations is given by $\phi^{min}(0,n)$.

TABLE II: Time and space complexity

	BCJR		rMAP [7]	Hybrid [8]	Proposed probability-domain			Proposed LLR-domain		
Code	mult	space	mult	mult	mult	sum	space	comp	sum	space
$RM(64,22)$	1125300	324861	174464	116096	114096	134784	14848	114304	126384	14848
$RM(64,42)$	1648308	324861	4492672	498307	495024	491136	14848	486016	498096	14848
$eBCH(64,24)$	3398580	938237	669056	411008	409008	423552	27136	403072	421296	27136
$eBCH(64,36)$	32637108	7241725	105056705	20579792	17401968	13278328	4393544	13112260	17501190	4393544
$eBCH(64,39)$	64880820	12681213	444744424	58328480	42567758	32283700	10553896	32117620	42666980	10553896
$eBCH(64,45)$	2250420	418045	15525680	1625440	1624680	1248476	399296	1244624	1626984	399296

TABLE III: Decoding latency

Code	BCJR	Proposed (uniform)
$RM(64,7)$	135	17
$RM(64,22)$	150	29
$RM(64,42)$	170	39
$RM(64,57)$	185	35

Let $\psi(x,y)$ denote space complexity, i.e. the number of entries in the CBT for section $[x,y)$. It is given by

$$\psi(x,y) = \begin{cases} 0, & (y-x) = n \\ 2^{k'_{x,y}}(y-x) + 2^{k'_{x,y}+1}, & (y-x) = 2 \neq k'_{x,y} \\ 2^{k'_{x,y}+1}, & \text{otherwise.} \end{cases}$$

Let $f(x,y)$ denote the decoding latency, i.e. the maximum number of sequential operations for section $[x,y)$. This function can be given by

$$f(x,y) = \begin{cases} 0, \quad (y-x) = n \\ \log_2(y-x) + k'_{x,y} + k''_{x,y} + 1, \quad \text{non-recursive steps} \\ \max(f(x,z), f(z,y)) + k'_{x,y} + 2k''_{x,y} \\ \qquad -\min(k'_{x,z}, k'_{z,y}) + 2, \qquad \text{recursive steps.} \end{cases}$$

Here we assume that computation of (2) and (10) requires $\log_2(y-x)+k''_{x,y}$ sequential operations, and computation of (9) requires $k'_{x,y}+1$ sequential operations. On the recursive steps one need to compute (3) with latency $1+k''_{x,y}$ operations and (6)–(7) with latency $k'_{x,y}+k''_{x,y}-\min(k'_{x,z}, k'_{z,y})+1$ operations.

V. Numeric results

We consider decoding of Reed-Muller (RM) and extended BCH codes of length 64. Tab. II presents the number of multiplications needed by the probability-domain implementation of the proposed algorithm, the recursive SISO algorithms given in [7], [8], and the classical BCJR algorithm. It can be seen that the proposed algorithm has the smallest complexity. Moreover, Tab. II demonstrates noticeable space complexity benefit of the proposed approach compared to the original BCJR algorithm.

Tab. II also shows the complexity of the probability- and LLR-domain implementations of the proposed method. It can be seen that the total number of operations in the latter case is lower, despite of the fast 2-point cyclic convolutional algorithm being used only in the former implementation. Tab. III presents the latency analysis for some RM codes. It can be seen that significant reduction of latency is achieved under uniform sectionalization. Fig. 4 illustrates the performance of $(64,45,8)$ eBCH code. Although the LLR-domain implementation is an approximate one, it results in slightly better FER-performance compared to the probability-domain implementation. This is due to the fact that LLR-domain decoder always returns the vector of LLRs, such that their sign bits correspond to the most likely codeword (maximum-likelihood decoder).

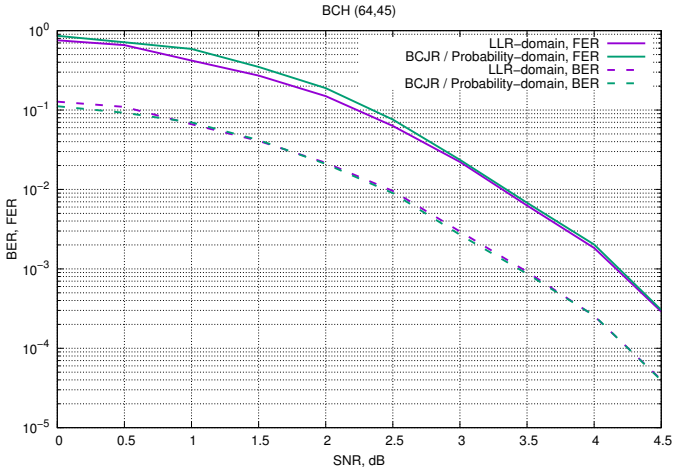


Fig. 4: FER/BER comparison of $(64,45,8)$ eBCH code for probability-domain and LLR-domain implementations

VI. Conclusion

In this paper a novel SISO decoding algorithm for linear block codes was presented. It employs different data structures compared to [7], [8] and performs two passes over the recursion tree. The proposed approach was shown to have lower complexity compared to the classical BCJR algorithm and recursive algorithms given in [7], [8]. Observe that the purely recursive algorithm [7] has much higher complexity compared to the proposed method. Hybrid algorithm in [8] employs a combination of the BCJR algorithm and recursive algorithm in [7], but has higher complexity compared to the proposed purely recursive approach.

An LLR-domain implementation of the proposed method was presented, which employs only summation and comparison operations. It is less prone to numeric errors compared to the probability-domain implementation.

Acknowledgment

The authors thank Prof. B.D. Kudryashov and Dr. I.E. Bocharova for many discussions and suggestions. This work is partially supported by the Ministry of Science and Higher Education of Russian Federation, passport of goszadanie no. 2019-0898.

References

[1] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Transactions on Information Theory*, pp. 284–287, 1974.

[2] T. Johansson and K. Zigangirov, "A simple one-sweep algorithm for optimal APP symbol decoding of linear block codes," *IEEE Transactions on Information Theory*, vol. 44, no. 7, pp. 3124–3129, 1998.
[3] A. Vardy and Y. Beery, "Maximum-likelihood soft decision decoding of BCH codes," *IEEE Transactions On Information Theory*, vol. 40, no. 2, March 1994.
[4] F. R. Kschischang and G. B. Horn, "A heuristic for ordering a linear block code to minimize trellis state complexity," in *Proceedings of 32nd Annual Allerton Conference on Communication, Control, and Computing*, Allerton Park, Illinois, 1994, pp. 75–84.
[5] T. Fujiwara, H. Yamamoto, T. Kasami, and S. Lin, "A trellis-based recursive maximum-likelihood decoding algorithm for binary linear block codes," *IEEE Transactions On Information Theory*, vol. 44, no. 2, March 1998.
[6] A. Lafourcade and A. Vardy, "Optimal sectionalization of a trellis," *IEEE Transactions On Information Theory*, vol. 42, no. 3, pp. 689–702, May 1996.
[7] Y. Kaji, R. Shibuya, T. Fujiwara, T. Kasami, and S. Lin, "MAP and LogMAP decoding algorithms for linear block codes using a code structure," *IEICE Transactions on Fundamentals*, vol. E83-A, no. 10, October 2000.
[8] R. Shibuya and Y. Kaji, "An efficient MAP decoding algorithm which uses the BCJR and the recursive techniques," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 84, no. 10, pp. 2389–2396, 2001.
[9] P. Trifonov, "Recursive trellis decoding techniques of polar codes," in *Proceedings of IEEE International Symposium on Information Theory*, 2020.