# Time-varying block codes for synchronisation errors: maximum a posteriori decoder and practical issues

*Johann A. Briffa*[1], *Victor Buttigieg*[2], *Stephan Wesemeyer*[1]

[1]*Department of Computing, University of Surrey, Guildford GU2 7XH, UK*
[2]*Department of Communications & Computer Engineering, University of Malta, Msida MSD 2080, Malta*
*E-mail: j.briffa@surrey.ac.uk*

Published in *The Journal of Engineering*; Received on 27th February 2014; Accepted on 29th May 2014

**Abstract:** In this study, the authors consider time-varying block (TVB) codes, which generalise a number of previous synchronisation error-correcting codes. They also consider various practical issues related to maximum a posteriori (MAP) decoding of these codes. Specifically, they give an expression for the expected distribution of drift between transmitter and receiver because of synchronisation errors. They determine an appropriate choice for state space limits based on the drift probability distribution. In turn, they obtain an expression for the decoder complexity under given channel conditions in terms of the state space limits used. For a given state space, they also give a number of optimisations that reduce the algorithm complexity with no further loss of decoder performance. They also show how the MAP decoder can be used in the absence of known frame boundaries, and demonstrate that an appropriate choice of decoder parameters allows the decoder to approach the performance when frame boundaries are known, at the expense of some increase in complexity. Finally, they express some existing constructions as TVB codes, comparing performance with published results and showing that improved performance is possible by taking advantage of the flexibility of TVB codes.

## 1 Introduction

Most error-control systems are designed to detect and/or correct substitution errors, where individual symbols of the received sequence have been substituted while maintaining synchronisation with the transmitted sequence. Some channels, however, also experience 'synchronisation errors', where symbols may additionally be deleted from or inserted into the received sequence. It has long been recognised that codes can be designed specifically for synchronisation error correction [1, 2]. Except for the less-known work by Gallager [3], for a long time only some short block codes were known. This changed when Davey and MacKay [4] proposed a concatenated scheme combining an outer low-density parity-check (LDPC) code with good error-correction capability with an inner code whose aim is to correct synchronisation errors. Ratzer [5] took a different approach, using short marker sequences inserted in binary LDPC codewords; a similar approach was used by Wang *et al.* [6]. Yet another approach extends the state space of convolutional codes to allow correction of synchronisation errors [7–9]. The problem of convolutional code design for synchronisation error channels has been considered in [10]. More recently, this approach has been applied successfully to turbo codes [11]. The renewed increase in interest is mainly because of new applications requiring such codes. A recent survey can be found in [12].

We have in previous papers extended the work of Davey–MacKay, proposing a 'maximum a posteriori' (MAP) decoder [13], improved code designs [14, 15] as well as a parallel implementation of the MAP decoder resulting in speedups of up to two orders of magnitude [16]. However, these papers were restricted to the case where the frame boundaries were known by the decoder. Although Davey and MacKay showed that the frame boundaries could be accurately determined for their bit-level decoder and code construction [4], it has not been shown whether this property extends to our MAP decoder and improved constructions.

In this paper, we define time-varying block (TVB) codes in terms of the encoding used in [16], and show that TVB codes represent a new class of codes which generalises a number of previous synchronisation error-correcting codes. We use the MAP decoder of [16] for these codes, showing how it can be used in an iterative scheme with an outer code. We also consider a number of important issues related to any practical implementation of the MAP decoder. Specifically, we give an expression for the expected distribution of drift between transmitter and receiver because of synchronisation errors. We determine an appropriate choice for state space limits based on the drift probability distribution. In turn, we obtain an expression for the decoder complexity under given channel conditions in terms of the state space limits used. For a given state space, we also give a number of optimisations that reduce the algorithm complexity with no further loss of decoder performance. We also show how the MAP decoder can be used for 'stream decoding', where the boundaries of the received frames are not known a priori. In doing so, we demonstrate how an appropriate choice of decoder parameters allows stream decoding to approach the performance when frame boundaries are known, at the expense of some increase in complexity. We express some previously published codes as TVB codes, comparing performance with published results and showing that the greater flexibility of TVB codes permits the creation of improved codes.

In the following, we start with definitions in Section 2 and summaries of results from earlier work. The applicable design criteria for TVB codes are considered in Section 3, together with the representation of previously published codes as TVB codes. The appropriate choice for state space limits is given in Section 4, followed by expressions for the decoder complexity in Section 5. MAP decoder optimisations are given in Section 6 and the changes necessary for stream decoding in Section 7. Finally, practical results are given in Section 8.

## 2 Background

### 2.1 TVB codes

Consider the encoding defined in [16], used there to simplify the representation of the inner code of [4]. We observe that this encoding generalises a number of additional previous schemes, including the inner codes of [5, 6, 15] (see Section 3.2). We define a TVB code in terms of this encoding by the sequence $\mathcal{C} = (C_0, \ldots, C_{N-1})$, which consists of the constituent encodings $C_i: \mathbb{F}_q \hookrightarrow \mathbb{F}_2^n$ for $i = 0, \ldots, N-1$, where $n$, $q$, $N \in \mathbb{N}$, $2^n \geq q$ and

$\hookrightarrow$ denotes an injective mapping. Two constituent encodings $C_i$, $C_j$ are said to be 'equal' if $C_i(D) \in C_j\ \forall D$. For a given TVB code, the set of 'unique' constituent encodings is that set where no two constituent encodings are equal; the cardinality of this set, denoted by $M \leq N$, is called the order of the code. Note that unique constituent encodings may still have some common codewords. We denoted a TVB code by the tuple $(n, q, M)$. We restrict ourselves to binary TVB codes, where codewords are sequences of bits; the extension to the non-binary case is trivial.

For any sequence $\boldsymbol{z}$, denote arbitrary subsequences as $\boldsymbol{z}_a^b = (z_a, \ldots, z_{b-1})$, where $\boldsymbol{z}_a^a = ()$ is an empty sequence. Given a message $\boldsymbol{D}_0^N = (D_0, \ldots, D_{N-1})$, each $C_i$ maps the $q$-ary message symbol $D_i \in \mathbb{F}_q$ to codeword $C_i(D_i)$ of length $n$. That is, $\boldsymbol{D}_0^N$ is encoded as $\boldsymbol{X}_0^{nN} = C_0(D_0)\|\cdots\|C_{N-1}(D_{N-1})$, where $\boldsymbol{y}\|\boldsymbol{z}$ is the juxtaposition of $\boldsymbol{y}$ and $\boldsymbol{z}$. Each $q$-ary symbol is encoded independently of previous inputs and different codebooks may be used for each input symbol. This time-variation offers no advantage on a fully synchronised channel. However, in the presence of synchronisation errors, the differences between neighbouring codebooks provide useful information to the decoder to recover synchronisation.

In practice, a TVB code is suitable as an inner code to correct synchronisation errors in a serially concatenated construction. A conventional outer code corrects residual substitution errors. In such a scheme, the inner code's MAP decoder 'a posteriori' probabilities (APPs) are used to initialise the outer decoder. The concatenated code can be iteratively decoded, in which case the prior symbol probabilities of the inner decoder are set using extrinsic information from the previous pass of the outer decoder.

### 2.2 Channel model

We consider the binary substitution, insertion and deletion (BSID) channel, an abstract random channel with unbounded synchronisation and substitution errors, originally presented in [17] and more recently used in [4, 5, 13–15] and others. At 'time' $t$, one bit enters the channel, and one of three events may happen: insertion with probability $P_\mathrm{i}$ where a random bit is output; deletion with probability $P_\mathrm{d}$ where the input is discarded; or transmission with probability $P_\mathrm{t} = 1 - P_\mathrm{i} - P_\mathrm{d}$. A substitution occurs in a transmitted bit with probability $P_\mathrm{s}$. After an insertion, the channel remains at time $t$ and is subject to the same events again; otherwise it proceeds to time $t+1$, ready for another input bit.

We define the 'drift' $S_t$ at time $t$ as the difference between the number of received bits and the number of transmitted bits before the events of time $t$ are considered. As in [4], the channel can be seen as a Markov process with the state being the drift $S_t$. It is helpful to see the sequence of states as a trellis diagram, observing that there may be more than one way to achieve each state transition. In addition, note that the state space is unlimited for positive drifts, but limited for negative drifts. Specifically, $S_t$ may take any positive value for $t > 0$, although with decreasing probability as the value increases. On the other hand, $S_t \geq -t$ where the lower limit corresponds to receiving the null sequence.

### 2.3 MAP decoder

We summarise here the MAP decoder of [16]; this is the same as the MAP decoder of [13] with a trivial modification to work with the notation of TVB codes. The decoder uses the standard forward–backward algorithm for hidden Markov models. We assume a message sequence $\boldsymbol{D}_0^N$, encoded using a $(n, q, M)$ TVB code to the sequence $\boldsymbol{X}_0^\tau$, where $\tau = nN$. The sequence $\boldsymbol{X}_0^\tau$ is transmitted over the BSID channel, resulting in the received sequence $\boldsymbol{Y}_0^\rho$, where, in general, $\rho$ is not equal to $\tau$. To avoid ambiguity, we refer to the message sequence as a 'block' of size $N$ and the encoded sequence as a 'frame' of size $\tau$. We calculate the APP $L_i(D)$ of having encoded symbol $D \in \mathbb{F}_q$ in position $i$ for $0 \leq i < N$, given the entire received sequence, using

$$L_i(D) = \frac{1}{\lambda_N(\rho - \tau)} \sum_{m',m} \sigma_i(m', m, D) \tag{1}$$

$$\text{where } \lambda_i(m) = \alpha_i(m)\beta_i(m) \tag{2}$$

$$\sigma_i(m', m, D) = \alpha_i(m')\gamma_i(m', m, D)\beta_{i+1}(m) \tag{3}$$

and $\alpha_i(m)$, $\beta_i(m)$ and $\gamma_i(m', m, D)$ are the forward, backward and state transition metrics, respectively. Note that strictly, the above metrics depend on $\boldsymbol{Y}_0^\rho$, but for brevity we do not indicate this dependence in the notation. The summation in (1) is taken over the combination of $m'$, $m$, being, respectively, the drift before and after the symbol at index $i$. The forward and backward metrics are obtained recursively using

$$\alpha_i(m) = \sum_{m',D} \alpha_{i-1}(m')\gamma_{i-1}(m', m, D) \tag{4}$$

$$\text{and } \beta_i(m) = \sum_{m',D} \beta_{i+1}(m')\gamma_i(m, m', D) \tag{5}$$

Initial conditions for known frame boundaries are given by

$$\alpha_0(m) = \begin{cases} 1, & \text{if } m = 0 \\ 0, & \text{otherwise,} \end{cases}$$

and

$$\beta_N(m) = \begin{cases} 1, & \text{if } m = \rho - \tau \\ 0, & \text{otherwise} \end{cases}$$

Finally, the state transition metric is defined as

$$\gamma_i(m',\ m,\ D) = \Pr\{D_i = D\} R\left(\boldsymbol{Y}_{ni+m'}^{n(i+1)+m} | C_i(D)\right) \tag{6}$$

where $C_i(D)$ is the $n$-bit sequence encoding $D$ and $R(\dot{\boldsymbol{y}}|\boldsymbol{x})$ is the probability of receiving a sequence $\dot{\boldsymbol{y}}$ given that $\boldsymbol{x}$ was sent through the channel (we refer to this as the receiver metric). The a priori probability $\Pr\{D_i = D\}$ is determined by the source statistics, which we generally assume to be equiprobable so that $\Pr\{D_i = D\} = 1/q$. In iterative decoding, the prior probabilities are set using extrinsic information from the previous pass of an outer decoder, as explained in Section 2.1. The receiver metric is obtained by calculating the forward recursion

$$\dot{\alpha}_t(m) = \sum_{m'} \dot{\alpha}_{t-1}(m')\, Q\left(\dot{\boldsymbol{y}}_{t-1+m'}^{t+m} | x_{t-1}\right) \tag{7}$$

where for brevity we do not show the dependence on $\dot{\boldsymbol{y}}$ and $\boldsymbol{x}$ and $Q(\boldsymbol{y}|x)$ can be directly computed from $\boldsymbol{y}$, $x$ and the channel parameters

$$Q(\boldsymbol{y}|x) = \begin{cases} P_\mathrm{d}, & \text{if } \mu = 0 \\ \left(\frac{P_\mathrm{i}}{2}\right)^{\mu-1}\left(P_\mathrm{t}P_\mathrm{s} + \frac{1}{2}P_\mathrm{i}P_\mathrm{d}\right), & \text{if } \mu > 0,\ y_{\mu-1} \neq x \\ \left(\frac{P_\mathrm{i}}{2}\right)^{\mu-1}\left(P_\mathrm{t}\bar{P}_\mathrm{s} + \frac{1}{2}P_\mathrm{i}P_\mathrm{d}\right), & \text{if } \mu > 0,\ y_{\mu-1} = x \end{cases}$$

where $\mu$ is the length of $y$ and $\bar{P}_\mathrm{s} = 1 - P_\mathrm{s}$. The required value of the receiver metric is given by $R(\dot{\boldsymbol{y}}|\boldsymbol{x}) = \dot{\alpha}_n(\dot{\mu} - n)$, where $\dot{\mu}$ is the length of $\dot{\boldsymbol{y}}$ and $n$ is the length of $\boldsymbol{x}$.

As in [16], the $\alpha$, $\beta$ and $\dot{\alpha}$ metrics are normalised as they are computed to avoid exceeding the limits of floating-point representation. We also assume that (7) is computed at single precision (i.e. 32-bit floating point), whereas the remaining equations use double precision (i.e. 64-bit floating point).

## 3 TVB code design

### 3.1 Construction criteria

In any error-correcting scheme, the decoder's objective is to minimise the probability of decoding error (at the bit or codeword level depending on the application). If the channel does not introduce synchronisation errors, this optimisation may be performed independently of previous or subsequent codewords. Hence the performance of the code depends exclusively on its distance properties. In particular, the performance of the code at low channel error rate is dominated by the code's minimum Hamming distance. At any channel error rate, the performance is determined by the code's distance spectrum [18, 19]. Thus when designing codes for substitution error channels, either the minimum Hamming distance or the more complete distance spectrum needs to be optimised for the given code parameters.

In the case of the BSID channel and other channels that allow synchronisation errors, a similar behaviour is observed if the codeword boundaries are known, only this time the Levenshtein distance [2] replaces the Hamming one. Recall that the Levenshtein distance gives the minimum number of edits (insertions, deletions or substitutions) that will change one codeword into another.

For the BSID channel, an upper bound for the probability of decoding a codeword in error was given in [15], assuming codeword boundaries are known. For $P_i$, $P_d$, $P_s \ll 1$, the bound of [15, (9)] is dominated by the number of correctable errors, $t$. Now, for a code with minimum Levenshtein distance $d_{l_{\min}}$, it can be shown that $t = \left\lfloor (d_{l_{\min}} - 1)/2 \right\rfloor$ [2]. Hence designing TVB codes with constituent encodings having large $d_{l_{\min}}$ will result in the greatest improvement to the code's performance at low channel error rates.

However, in general, the codeword boundaries are not known and need to be estimated by the decoder. Therefore decoding a given codeword on synchronisation error channels depends not only on the current received word, but also on previous and subsequent ones. This means that the performance of a TVB code depends not only on the distance properties of constituent encodings considered separately, but also on the relationship between constituent encodings. This effect becomes more significant under poorer channel conditions, where the drift can easily exceed the length of a codeword. Unfortunately, the required relationship between constituent encodings for optimal performance over the BSID channel is still an open problem. What is known is that the diversity created by a sequence of different encodings helps the decoder estimate the drift within a codeword length, improving performance at higher channel error rates [15].

### 3.2 Representation of previous schemes as TVB codes

TVB codes generalise a number of existing synchronisation error-correcting codes. The flexibility of the generalisation allows the creation of improved codes at the same size and rate, as we shall show.

Consider first the sparse inner codes with a distributed marker sequence (originally called a watermark sequence) of the Davey–MacKay construction [4]. It is clear that the sparse code is a fixed encoding $C' : \mathbb{F}_q \hookrightarrow \mathbb{F}_2^n$; these codewords are then added to a distributed marker sequence $\boldsymbol{w}_i$ of length $n$, specific for each codeword index $i$. Thus, we can write $C_i(D_i) = C'(D_i) + \boldsymbol{w}_i$ to represent the inner codes of [4] as TVB codes. The equivalence of this mapping to the inner code of [4] has also been shown in [16]. The distributed marker serves the same function as the use of different encodings in TVB codes. The decoder of [4] tracks the marker sequence directly, treating the additive encoded message sequence as substitution errors. Therefore, to corrupt the marker sequence as little as possible, the inner code used is sparse. The sparseness results in a low $d_{l_{\min}}$, making it harder for the decoder to distinguish between the various codewords, and leads to relatively poor performance at low channel error rates.

**Table 1** A (7,8,4) TVB code $\mathcal{C} = (C_0, \ldots, C_3)$ with $d_{l_{\min}} = 3$

| $C_0$ | $C_1$ | $C_2$ | $C_3$ |
|---|---|---|---|
| 0000000 | 0000000 | 0000011 | 0000000 |
| 0000111 | 0000111 | 0001100 | 0001111 |
| 0011001 | 0011110 | 0011111 | 0101001 |
| 0110110 | 0110101 | 0101010 | 0110110 |
| 1001010 | 1001001 | 1011001 | 1000011 |
| 1100001 | 1100110 | 1100000 | 1001100 |
| 1111000 | 1111000 | 1100111 | 1110000 |
| 1111111 | 1111111 | 1111110 | 1111111 |

The codes of [15] can similarly be represented as TVB codes, with $C'$ corresponding to the synchronisation and error-correcting (SEC) code and $\boldsymbol{w}_i$ corresponding to the allowed modification vectors (AMVs). SEC codes are designed with a large $d_{l_{\min}}$ for good performance at low channel error rates. For such channels, this code can perform much better than the sparse code of [4]. AMVs are chosen such that when added to the SEC code the resulting code's $d_{l_{\min}}$ does not change. Clearly, the AMVs serve the same function as the use of different encodings in TVB codes. In contrast to a random distributed marker sequence, the use of AMVs does not compromise the performance of the underlying SEC code at low channel error rates. In general, however, the Levenshtein distance spectrum is altered. The separate constituent encodings in TVB codes give greater design freedom than SEC codes with AMVs and also allows the design of constituent encodings that maintain the required optimised Levenshtein distance spectrum.

The marker codes given by Ratzer [5] can also be cast as TVB codes by letting each possible sequence of data bits (between markers) be represented by a $q$-ary symbol. For example, consider a marker code with three marker bits inserted after every nine data bits, where the 3-bit marker is randomly chosen between the sequences 001 and 110. This can be represented as a (12, 512, 2) TVB code, where encoding $C_0$ consists of all possible 9-bit sequences appended with 001 and $C_1$ consists of all possible 9-bit sequences appended with 110. Like the sparse codes of [4], these marker codes suffer from a low $d_{l_{\min}}$, leading to relatively poor performance at low channel error rates. On the other hand, the fixed marker bits improve the determination of codeword boundaries and the random use of different marker bits creates the necessary diversity to improve performance in poorer channel conditions.

To illustrate the difference in performance between the various designs, consider encodings of size $(n, q) = (7, 8)$ with $N = 666$ (same size as codes C and H in [4]). A (7,8,4) TVB code where each constituent code has the best possible Levenshtein distance spectrum with $d_{l_{\min}} = 3$, found through an exhaustive search, is given in Table 1. In Fig. 1, we compare this TVB code with earlier constructions from the literature at the same size. Consider first the SEC code of the same size and $d_{l_{\min}}$ from [15], used with eight AMVs in a random sequence. As expected, the TVB code performs better because of its improved Levenshtein distance spectrum, even though both TVB and SEC codes have the same $d_{l_{\min}}$. The performance of the sparse code with random distributed marker from [4] is considerably worse, particularly at low channel error rates. Similarly a code with three data bits and four marker bits (randomly chosen between 0011/1100), similar to [5], also performs poorly at low channel error rates.

## 4 Appropriate limits on state space

The equations in Section 2.3 assume that summations can be taken over the set of all possible states. For a channel such as the one considered, the state space is unbounded for positive drifts. A practical

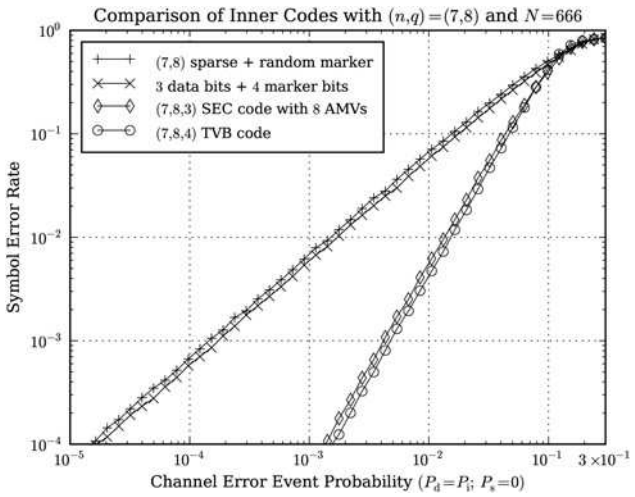


**Fig. 1** *Comparison of inner code designs of size (n, q) = (7, 8) with N = 666: a sparse code with random distributed marker from [4], a marker code with 0011/1100 marker bits similar to [5], an SEC code with randomly sequenced AMVs from [15] and the TVB code of Table 1*

implementation will have to take sums over a finite subset of states. In [4], the state space was limited to a drift $|S_i| \leq x_{\max}$, where $x_{\max}$ was chosen to be 'several times larger' than the standard deviation of the synchronisation drift over one block length, assuming this takes a Gaussian distribution. No recommendation was given for the value that should be used.

Limiting the state space is by definition sub-optimal. However, we can arbitrarily lower the number of cases where the sub-optimal solution is worse than the optimal one, by ensuring that only the least likely states are omitted. The choice of summation limits also involves a trade-off with complexity, which has a polynomial relationship with the size of the state space (see Section 5). Therefore an appropriate choice of summation limits will result in the smallest state space such that the probability of the drift being outside that range is as low as required. The first step to identify good summation limits is to derive an accurate probability distribution of the state space, avoiding the Gaussian approximation of [4].

### 4.1 Drift probability distribution

The drift $S_T$ after transmission of T bits was stated in [4] (and shown in [20]) to be normally distributed with zero mean and a variance equal to $Tp/(1-p)$ for the special case where $p := P_i = P_d$. This distribution is asymptotically valid as $T \to \infty$. For cases where $P_i \neq P_d$ or where $T$ is not large enough, this distribution cannot be used. This is particularly relevant for determining the summation limits of (7) where the sequence length $n$ is not large. An exact expression for the probability distribution of $S_T$ is given by

$$\begin{aligned}\Phi_T(m) &= \Pr\{S_T = m\} \\ &= P_t^T P_i^m \sum_{j=j_0}^{T} \binom{T}{j} \binom{T+m+j-1}{m+j} \left[\frac{P_i P_d}{P_t}\right]^j \end{aligned} \tag{8}$$

where $j_0 = \max(-m, 0)$. Observe that for a drift of $m$ bits, we need $m$ insertion events more than we have deletion events. Over a sequence of $T$ bits, for $j$ deletion events, this means $m+j$ insertion events and $T-j$ transmission events. The probability of this is $P_i^{m+j} P_d^j P_t^{T-j} = P_t^T P_i^m \left[P_i P_d / P_t\right]^j$. We obtain (8) by adding all different combinations of these events, and summing over $j$, noting that we cannot have fewer than 0 events of any type. Specifically, the number of combinations for $j$ deletions in $T$ transmitted bits is given by $\binom{T}{j}$. The number of combinations for $m+j$ insertions is given by $\binom{T+m+j-1}{m+j}$, as the $m+j$ insertion events create an additional $m+j$ opportunities for insertion.

### 4.2 Avoiding numerical issues

In a practical implementation, computing the drift probability (8) requires a few special considerations. Practical codes from the literature have codeword size $n$ in the range 5–12 bits and number of codewords $N$ up to 1000, for a frame length $nN$ of about 4000–6000 bits. These codes are designed to operate under channel conditions $P_i$, $P_d$ from $10^{-3}$ to above $10^{-1}$. Evaluating (8) under these conditions, one encounters very large values for the two binomial coefficients and very small values for the power term. For example, consider evaluating (8) at $m = 0$ for $T = 6000$ and $P_i = P_d = 10^{-3}$. The two binomial coefficients have a range of up to $1.56 \times 10^{1804}$ (at $j = 3000$) and $8.34 \times 10^{3609}$ (at $j = 6000$). The power term has a range of down to $1.65 \times 10^{-35\,995}$ (at $j = 6000$). This range is far beyond that representable even in double-precision floating point. A direct implementation of (8) will therefore result in numerical overflow and underflow (in computing the binomial coefficients and power term, respectively) for typical frame sizes and channel conditions, even though the summation term itself is representable.

The above numerical range problem can be avoided by combining the computation of all terms in the summation as follows. Observe that (8) can be rewritten as

$$\Phi_T(m) = \sum_{j=j_0}^{T} \delta_j \tag{9}$$

$$\text{where } \delta_j = P_t^T P_i^m \binom{T}{j} \binom{T+m+j-1}{m+j} \left[\frac{P_i P_d}{P_t}\right]^j \tag{10}$$

and $j_0 = \max(-m, 0)$ as before. In this expression note that the summation is empty if $j_0 > T$, resulting in zero probability. In addition, since $j \geq 0$, the first binomial coefficient is always non-zero, whereas the second binomial coefficient is non-zero if $T > 0$. Expanding the binomial coefficients using the factorial formula, we can express the summation term recursively as

$$\delta_j = \delta_{j-1} \frac{P_i P_d}{P_t} \frac{T+m+j-1}{m+j} \frac{T-j+1}{j} \tag{11}$$

allowing successive factors to be determined easily from previous ones. The initial factor required is the one at $j_0$ and can be determined from (10) by expanding the binomial coefficients using the multiplicative formula

$$\delta_{j_0} = P_t^T P_i^m \prod_{i=1}^{j_0} \frac{T-j_0-i}{i} \prod_{i=1}^{m+j_0} \frac{T-1-i}{i} \left[\frac{P_i P_d}{P_t}\right]^{j_0} \tag{12}$$

Consider the earlier example, now evaluating (9) at $m = 0$ for $T = 6000$ and $P_i = P_d = 10^{-3}$. In this case, the initial value $\delta_{j_0} = 6.07 \times 10^{-6}$ and the multiplier $\delta_j / \delta_{j-1}$ in the recursive expression (11) has its smallest value of $3.34 \times 10^{-10}$ at $j = 6000$. Both values are easily representable as floating point numbers.

Using (9), numerical range issues remain when computing $\delta_{j_0}$ for larger values of $P_i$, $P_d$ and consequently also for storing successive values of $\delta_j$. For example, consider evaluating (9) at $m = 0$ for $T = 6000$ and $P_i = P_d = 10^{-1}$. In this case $\delta_{j_0} = 3.47 \times 10^{-582}$ and one needs to accumulate a number of $\delta_j$ values in this range to obtain the required result $\Phi_{6000}(0) = 0.0109$. Again, the intermediate values are beyond the range of double-precision floating point numbers, although the final result is representable. These numerical range issues can be avoided by computing (11) and (12) using logarithms.

For the earlier example with $m=0$, $T=6000$ and $P_\mathrm{i}=P_\mathrm{d}=10^{-1}$, we now obtain $\log \delta_{j_0} = -1.34 \times 10^3$ and the smallest value of $\log \delta_j$ is $-1.93 \times 10^4$ at $j=6000$.

Finally, the required drift probability is obtained by accumulating the exponential of the $\log \delta_j$ values using (9). However, the individual values of $\delta_j$ are still beyond the range of double-precision floating point. In practice, we have found that the use of extended-precision (80 bit) floating point provides sufficient range. Alternatively, the accumulation in (9) may be computed in logarithmic domain using the property $\log(A+B) = \log A + \log(1 + e^{\log B - \log A})$.

Note that expression (8) is valid for any $P_\mathrm{i} \geq 0$, $P_\mathrm{d} \geq 0$ and $P_\mathrm{i} + P_\mathrm{d} < 1$. However, the computation using logarithms cannot be applied directly when either or both of $P_\mathrm{i}$ and $P_\mathrm{d}$ are zero. These degenerate cases have to be handled as special cases, by first reducing (8) and then implementing the simplified equations using logarithms.

### 4.3 Probability of drift outside range

We want to choose lower and upper limits $m_T^-$, $m_T^+$ such that the drift after transmitting a sequence of $T$ bits is outside the range $\{m_T^- \cdots m_T^+\}$ with an arbitrarily low probability $P_\mathrm{r}$

$$\Pr\{S_T < m_T^-\} + \Pr\{S_T > m_T^+\} < P_\mathrm{r} \tag{13}$$

$$\text{or equivalently:} \quad 1 - \sum_{m=m_T^-}^{m_T^+} \Phi_T(m) < P_\mathrm{r} \tag{14}$$

An appropriate choice of limits can be obtained iteratively as follows. Observe that for the BSID channel $\Phi_T(m)$ is monotonically decreasing with increasing $|m|$. A first estimate for the limits is given by

$$m_T^{-(1)} = \max m \left| \Phi_T(m-1) < \frac{P_\mathrm{r}}{2} \right. \tag{15}$$

$$\text{and } m_T^{+(1)} = \min m \left| \Phi_T(m+1) < \frac{P_\mathrm{r}}{2} \right. \tag{16}$$

where the number in superscript parentheses indicates the iteration count. If these estimates satisfy (14), we use them as our lower and upper limits. Otherwise these estimates are updated iteratively as follows

$$m_T^{-(i+1)} = \begin{cases} m_T^{-(i)} - 1, & \text{if } \Phi_T(m_T^{+(i)} + 1) \leq \Phi_T(m_T^{-(i)} - 1) \\ m_T^{-(i)}, & \text{otherwise} \end{cases} \tag{17}$$

$$m_T^{+(i+1)} = \begin{cases} m_T^{+(i)} + 1, & \text{if } \Phi_T(m_T^{+(i)} + 1) > \Phi_T(m_T^{-(i)} - 1) \\ m_T^{+(i)}, & \text{otherwise} \end{cases} \tag{18}$$

That is, we extend the range by one in the direction of greatest gain. The iterative process is repeated until (14) is satisfied. The size of the state space is given by $M_T = m_T^+ - m_T^- + 1$.

### 4.4 Choice of summation limits

When considering the whole frame, $T=\tau$, so that the overall size of the state space is given by $M_\tau$. Now the final output of the MAP decoder is calculated using (1), which sums over all $M_\tau$ prior states $m_\tau^- \leq m' \leq m_\tau^+$. For each prior state, however, only the drifts introduced by the transmission of $n$ bits need to be considered, corresponding to a subset $M_n$ of states $m_n^- \leq m \leq m_n^+$. Similarly, the computation of (4) and (5) is required for all $M_\tau$ states $m_\tau^- \leq m \leq m_\tau^+$, each involving a summation over $M_n$ prior or posterior states $m_n^- \leq m' \leq m_n^+$. Finally, the state transition metric is obtained using the forward pass of (7); this is computed over a sequence of $n$ bits for each of $M_n$ states $m_n^- \leq m \leq m_n^+$. Each recursion consists of a summation over prior states $m'$; in this case, only the drifts introduced by the transmission of one bit need to be considered, corresponding to a subset $M_1$ of prior states $m_1^- \leq m' \leq m_1^+$.

Now consider that we want to limit the probability of any of these summations not covering an actual channel event over a whole frame to, say, no more than $P_\mathrm{e}$. When computing the limits over the whole frame, $m_\tau^\pm$, we simply need to set $P_\mathrm{r}=P_\mathrm{e}$. However, when computing limits over an $n$-bit sequence, $m_n^\pm$, since this summation is repeated for each of $N$ such sequences, we set $P_\mathrm{r} = 1 - \sqrt[N]{1-P_\mathrm{e}} \simeq P_\mathrm{e}/N$ for small $P_\mathrm{e}$. Similarly, for limits over a 1-bit sequence, $m_1^\pm$, we use $P_\mathrm{r} = 1 - \sqrt[\tau]{1-P_\mathrm{e}} \simeq P_\mathrm{e}/\tau$ for small $P_\mathrm{e}$.

Except in the case of stream decoding (see Section 7), the state space limits only need to be determined once and remain valid as long as the channel conditions do not change. In any case, the required values of $\Phi_T(m)$ depend only on the code parameters and channel conditions, so that a table may be pre-computed. This makes the average complexity of determining the state space limits negligible.

### 4.5 Example

Overestimating the required state space increases computational complexity, whereas underestimating the state space often results in poor decoding performance. Accurate limits are particularly important for restricting the drifts considered across each codeword. It is therefore useful to illustrate the discrepancy between the approximate distribution of [4] and the exact expression for the distribution of the drift. Consider a system with typical block and codeword sizes $N=500$ and $n=10$. We plot in Fig. 2*a* the number of states within summation limits using the approximate and exact expressions, in each case for $P_\mathrm{e}=10^{-10}$. For $T=1$, [4, Section VII.A] assumes a maximum of two successive insertions; this is equivalent to setting $m_1^+ = 2$, so that $M_1 = 4$. It is immediately apparent that while the approximation is very close for large $T$ and high $P_\mathrm{i}=P_\mathrm{d}$, it quickly starts to underestimate the required range at lower channel error rates. As expected, the discrepancy is particularly large when considering shorter sequences. For $T=1$, it is not surprising that there is a large discrepancy for channels with high error rate.

Next, we determine the probability of encountering a channel event outside the chosen limits over a single frame, shown in Fig. 2*b* for the same limits used in Fig. 2*a*. For the exact distribution, this probability is always below the chosen threshold $P_\mathrm{e}=10^{-10}$, as expected. For the approximation, however, the probability of exceeding the chosen limits is higher than the threshold throughout the range considered. At lower channel error rates, the discrepancy is significant (several orders of magnitude) even for large $T$. For small $T$, the probability of exceeding the chosen limits is high enough to make the approximation useless. For $T=1$, the artificial limit of two successive insertions of [4] means that channels with high error rate will exceed this limit with high probability.

## 5 Algorithm complexity

### 5.1 Complexity of the MAP decoder

As a first step towards determining the overall complexity of the decoder, consider first the calculation of the state transition metric in (6). This is recursively computed using the forward pass of (7) over a sequence of $n$ bits, for each of $M_n$ states $m$. Each recursion consists of a summation over $M_1$ prior states $m'$ as argued in Section 4.4. The bit-level probability $Q$ can be obtained by a look-up table. Thus, the complexity for calculating a single state transition metric is $\Theta(nM_nM_1)$.

The final output of the algorithm consists of $q$ probabilities for each of $N$ symbols, calculated using (1). This equation sums over all $M_\tau$ prior states $m'$ and $M_n$ states $m$, defining the domain for

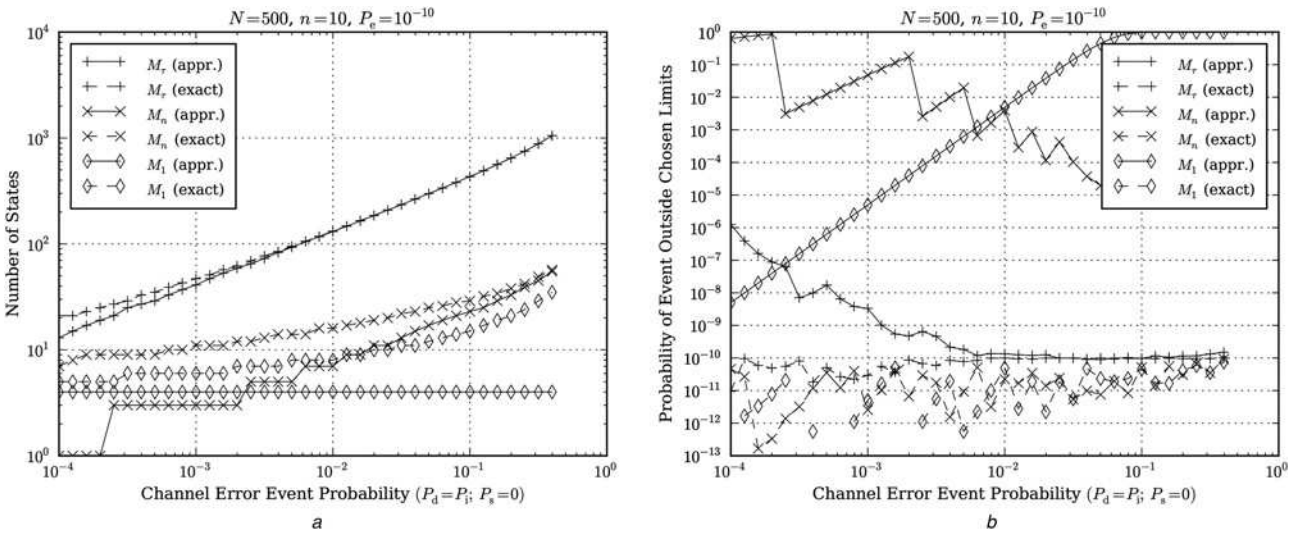


**Fig. 2** *Comparison of summation limits using the approximation of [4] and our exact computation*
*a* Number of states within summation limits
*b* Probability of encountering a channel event outside the chosen summation limits over a single frame

$\sigma_i(m', m, D)$. It follows from (3) that the domain for $\gamma_i(m', m, D)$ is the same. Now the computation of (3) is dominated by the evaluation of $\gamma_i(m', m, D)$ in (6), whose complexity is $\Theta(nM_nM_1)$ as shown. Considering the number of times the $\gamma$ metric is computed, the MAP decoder has an asymptotic complexity of $\Theta(NnqM_\tau M_n^2 M_1)$.

### 5.2 Complexity of the Davey–MacKay decoder

It would initially appear that the MAP decoder complexity is significantly higher than that of the Davey–MacKay decoder, given as $O(NnM_\tau M_1)$ in [4] for a direct implementation (using our notation). However, the expression of the Davey–MacKay decoder seems to consider only the complexity of the initial forward and backward passes, ignoring the additional small forward passes needed to compute the final decoder output.

The final output of the Davey–MacKay algorithm also consists of $q$ probabilities for each of $N$ symbols. Each of these is computed using [4, (4)], which sums over all possible prior and posterior states. In a direct implementation, all possible prior states need to be considered; using our notation the number of states is $M_\tau$. Although not stated in [4], the number of posterior states that need to be considered is $M_n$, as argued for the MAP decoder. The computation within the summation of [4, (4)] is dominated by the conditional probability, which is computed using a separate forward pass. This forward pass is effectively identical to (7) whose complexity is $\Theta(nM_nM_1)$. Considering the number of times the forward pass is computed, it follows that the Davey–MacKay decoder has an overall asymptotic complexity of $\Theta(NnqM_\tau M_n^2 M_1)$.

### 5.3 Comments on algorithm complexity

Comparing the complexity expressions for the MAP decoder for TVB codes and the Davey–MacKay decoder for sparse codes with a distributed marker sequence, it follows that the asymptotic complexity for both decoders is the same. This is consistent with experimental running times for both decoders in [13].

In the complexity expression, note that $N$, $n$ and $q$ depend only on the code parameters while $M_\tau$, $M_n$ and $M_1$ also depend on the channel conditions. For $M_1$, it was argued in [4, Section VII.A] that it is sufficient to consider a maximum of two successive insertions, at a minimal cost to decoding performance. This is equivalent to setting $m_1^+ = 2$, so that $M_1 = 4$; these limits were also used in [13, 14]. However, this artificially low limit is insufficient for more advanced code constructions, as shown in [15]. It was also argued in [4] that useful speedups can be obtained by only following paths through the trellis that pass through nodes with probabilities above a certain threshold. However, the choice of this threshold was not analysed. It is also likely that this choice would depend on the properties of the inner code being used.

## 6 Speeding things up

### 6.1 Batch computation of receiver metric

In a naïve implementation, each $\gamma$ computation (6) requires the computation of the receiver metric as a separate forward pass using (7). However it can be observed that for a given starting state $m'$ and symbol $D$, the $\gamma$ metric will be computed for each end state $m$ within the limits considered (see (1), (3)–(5), where the $\gamma$ computations are used). In turn, this means that for a given $x$, the receiver metric will need to be determined for all subsequences $\dot{\boldsymbol{y}}$ within the drift limit considered. It is therefore sufficient to compute the forward pass (7) once, with the longest subsequence $\dot{\boldsymbol{y}}$ required. In doing so, the values of the receiver metric for shorter subsequences are obtained for free. We call this approach 'batch' computation. This effectively reduces the complexity of computing the collection of $\gamma$ metrics by a factor of $M_n$. The asymptotic complexity of the MAP decoder is therefore reduced to $\Theta(NnqM_\tau M_n M_1)$.

### 6.2 Lattice implementation of receiver metric

To compute the receiver metric, an alternative to the trellis of (7) is to define a recursion over a lattice as in [17]. For the computation of $R(\dot{\boldsymbol{y}}|\boldsymbol{x})$, the required lattice has $n+1$ rows and $\dot{\mu}+1$ columns. Each horizontal path represents an insertion with probability $P_i/2$, each vertical path is a deletion with probability $P_d$, whereas each diagonal path is a transmission with probability $P_tP_s$ if the corresponding elements from $x$ and $\dot{y}$ are different or $P_t\bar{P}_s$ if they are the same. Let $F_{i,j}$ represent the lattice node in row $i$, column $j$. Then the lattice computation in the general case is defined by the recursion

$$F_{i,j} = \frac{1}{2}P_iF_{i,j-1} + P_dF_{i-1,j} + \dot{Q}(\dot{y}_j|x_i)F_{i-1,j-1} \qquad (19)$$

which is valid for $i < n$ and where $\dot{Q}(y|x)$ can be directly computed

from $y$, $x$ and the channel parameters

$$\dot{Q}(y|x) = \begin{cases} P_t P_s, & \text{if } y \neq x \\ P_t \bar{P}_s, & \text{if } y = x \end{cases} \tag{20}$$

Initial conditions are given by

$$F_{i,j} = \begin{cases} 1, & \text{if } i = 0,\ j = 0 \\ 0, & \text{if } i < 0 \text{ or } j < 0 \end{cases} \tag{21}$$

The last row is computed differently as the channel model does not allow the last event to be an insertion. In this case, when $i = n$, the lattice computation is defined by

$$F_{n,j} = P_d F_{n-1,j} + \dot{Q}(\dot{y}_j|x_n) F_{n-1,j-1} \tag{22}$$

Finally, the required receiver metric is obtained from this computation as $R(\dot{\boldsymbol{y}}|\boldsymbol{x}) = F_{n,\dot{\mu}}$. The calculation of a single run through the lattice requires a number of computations proportional to the number of nodes in the lattice. Now for the transmitted sequence of $n$ bits considered, the number of rows will always be $n$ while the number of columns is at most $n + m_n^+$. The complexity of a direct implementation of this algorithm is therefore $\Theta\big(n[n + m_n^+]\big)$.

It has been argued in Section 6.1 that for a given $x$, the receiver metric $R(\dot{\boldsymbol{y}}|\boldsymbol{x})$ needs to be determined for all subsequences $\dot{y}$ within the drift limit considered. Observe that the same argument applies equally when the receiver metric is computed using the lattice implementation (19). Therefore, when the lattice implementation is used in batch mode, the MAP decoder has an asymptotic complexity of $\Theta(NnqM_\tau[n + m_n^+])$.

### 6.3 Optimising the lattice implementation

In the lattice implementation of the receiver metric, it can be readily seen that the horizontal distance of a lattice node from the main diagonal is equivalent to the channel drift for the corresponding transmitted bit. It should therefore be clear that the likelihood of a path passing through a lattice node decreases as the distance to the main diagonal increases.

We can take advantage of the above observation by limiting the lattice computation to paths within a fixed corridor around the main diagonal. Specifically, the arguments of Section 4 can be applied directly, resulting in a corridor of width $M_n$, in general, for the transmitted sequence of $n$ bits considered. Exceptions to this width occur in the first few rows with index $i < -m_n^-$ and the last few rows with index $i > \dot{\mu} - m_n^+$, where part of the corridor falls outside the lattice rectangle. The number of nodes within this corridor is given by

$$\kappa = nM_n - \kappa_{\text{UL}} - \kappa_{\text{LR}} \tag{23}$$

$$\text{where } \kappa_{\text{UL}} = \Delta(-m_n^-) - \Delta(-m_n^- - n) \tag{24}$$

$$\kappa_{\text{LR}} = \Delta(n + m_n^+ - \dot{\mu}) - \Delta(m_n^+ - \dot{\mu}) \tag{25}$$

$$\text{and } \Delta(k) = \begin{cases} \dfrac{k^2 + k}{2}, & \text{if } k > 0 \\ 0, & \text{otherwise} \end{cases} \tag{26}$$

The complexity of the corridor-limited lattice algorithm is therefore $\Theta(nM_n - \kappa_{\text{UL}} - \kappa_{\text{LR}})$.

Some simplification of this expression is possible when the corridor-limited lattice algorithm is used in the MAP decoder with batch computation for the channel considered. When batch computation is used, $\dot{\mu} = n + m_n^+$ by definition, so that $\kappa_{\text{LR}} = 0$. Furthermore, for the BSID channel, $-n \leq m_n^- \leq 0$, so that $\kappa_{\text{UL}} = \big[(m_n^-)^2 - m_n^-\big]/2$. Under these conditions, the MAP decoder has an asymptotic complexity of $\Theta\big(NqM_\tau\big[nM_n - \big[(m_n^-)^2 - m_n^-\big]/2\big]\big)$.

**Table 2** Complexity expressions for the MAP decoder for various computation modes of the receiver metric

| | Algorithm | Complexity |
|---|---|---|
| A | original | $\Theta(NnqM_\tau M_n^2 M_1)$ |
| B | batch computation | $\Theta(NnqM_\tau M_n M_1)$ |
| C | lattice receiver | $\Theta(NnqM_\tau[n + m_n^+])$ |
| D | corridor constraint | $\Theta\big(NqM_\tau\big[nM_n - \big[(m_n^-)^2 - m_n^-\big]/2\big]\big)$ |

### 6.4 Comparing complexity

A summary of the complexity expressions for the MAP decoder for various computation modes of the receiver metric is given in Table 2. Comparing the expressions in rows A and B of Table 2, we can immediately see that the batch computation of the receiver metric reduces complexity by a factor equal to $M_n$. Unfortunately, the remaining complexity expressions contain terms that depend on the code parameters and channel conditions in a rather opaque way, making it harder to understand the benefits of these improvements. In the first instance, we can simplify the expressions further to facilitate comparison. Consider the expression in row C of Table 2, when the lattice implementation is used. It can be shown that $n + m_n^+ \to M_n - 1$ as channel conditions get worse; we can therefore simplify the complexity expression to $O(NnqM_\tau M_n)$. Comparing this to the expression in row B of Table 2, we can see that the use of the lattice implementation reduces complexity by a factor of at least $M_1$. Finally, consider the expression in row D of Table 2, when the corridor constraint is applied to the lattice algorithm. Since $m_n^- \leq 0$, the $\big[(m_n^-)^2 - m_n^-\big]/2$ term is strictly positive. The reduction in complexity offered by the corridor constraint is therefore equal to $2nM_n/\big(2nM_n - (m_n^-)^2 + m_n^-\big)$ and becomes significant as channel conditions improve. As channel conditions get worse, $m_n^- \to -n$, so that the expression is dominated by the $nM_n$ term. Under these conditions, the complexity of the corridor-constrained lattice implementation becomes approximately equal to that of the unconstrained lattice implementation.

We can also illustrate the effect of the proposed speedups by considering a rate-1/2 TVB code with typical block and codeword sizes $N = 500$, $n = 10$ and $q = 32$. We compute the MAP decoder

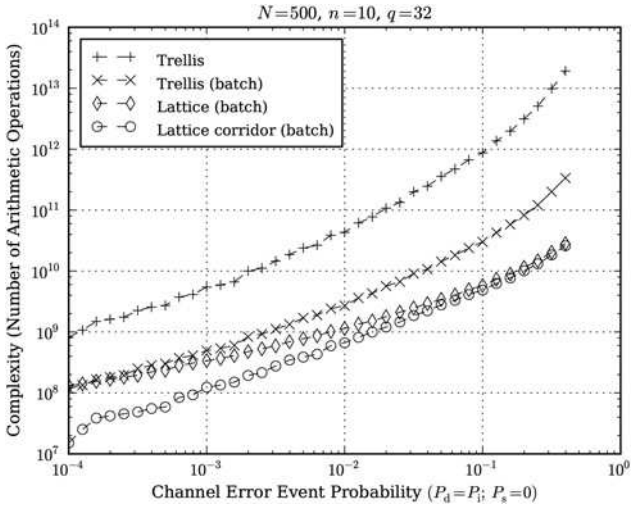


**Fig. 3** *MAP decoder complexity (in number of arithmetic operations) under a range of channel conditions, for various computation modes of the receiver metric*

complexity for this code under a range of channel conditions, using the original algorithm of Section 2.3 and the improvements described above. We plot these in Fig. 3 using the same summation limits as in Section 4.5. Note that for a fairer comparison between the lattice and trellis modes, we include a constant factor of three in the lattice computation. This follows the observation that each lattice node computation (19) requires three multiplications while each trellis computation (7) requires only one. A few general observations can be made on this graph: (a) The batch computation of the receiver metric results in a considerable reduction of complexity throughout, but is even more significant under poor channel conditions. (b) The lattice implementation is considerably less complex than the trellis implementation at high channel error rates. (c) The lattice corridor constraint extends this improvement to the low channel error rate range. In conclusion, the proposed speedups result in a considerable reduction in complexity of almost two orders of magnitude for typical code sizes and channel conditions. We have observed a similar trend under a range of typical code sizes, so this result can be taken as representative.

## 7 Stream decoding

We have so far considered the case where frame boundaries are known exactly. Although there are practical cases involving single-frame transmission where this is true, exact frame boundaries are often unknown. The MAP decoder can handle such cases by changing the initial conditions for (4) and (5) and choosing appropriate state space limits. This obviates the need for explicit frame-synchronisation markers as used in conventional communication systems, and can therefore reduce this overhead. The approach presented here is in principle similar to that used in [4] for 'sliding window' decoding. However, there are some critical differences which we explore further in Section 7.3.

### 7.1 Choosing end-of-frame priors

Consider first the common case where a sequence of frames is transmitted in a stream. The usual practice in communication systems is for the receiver to decode one frame at a time, starting the decoding process as soon as all the data related to the current frame are obtained from the channel. In this case, the current received frame is considered to be $Y_{m_\tau^-}^{\tau+m_\tau^+}$, which may include some bits from the end of the previous frame and start of the next frame. The end-state boundary condition for (5) can be obtained by convolving the expected end-of-frame drift probability distribution with the start-state distribution

$$\beta_N(m) = \sum_{m'} \alpha_0(m')\Phi_\tau(m - m') \tag{27}$$

Note that, in general, this distribution $\beta_N(m)$ has a wider spread than $\Phi_\tau(m)$.

As discussed in Sections 4.3 and 4.4, the choice of state space limits depends on the expected distribution of drift. For limits involving the whole frame, the distribution used is $\Phi_\tau(m)$, which assumes that the initial drift is zero. The assumption does not hold under stream decoding conditions, where the initial drift is not known a priori, although its distribution can be estimated. The uncertainty in locating the start-of-frame position increases the uncertainty in locating the end-of-frame position, resulting in a wider prior distribution for the end-state boundary condition $\beta_N(m)$. Therefore any limits on state space determined using $\Phi_\tau(m)$ will be underestimated. The severity of this error depends on the difference between $\beta_N(m)$ and $\Phi_\tau(m)$, which increases as channel conditions get worse. For stream decoding, therefore, it is sensible to recompute the state space limit $M_\tau$ at the onset of decoding a given frame, using $\beta_N(m)$ in lieu of $\Phi_\tau(m)$. Doing so avoids underestimating the required state space, and implies that for stream decoding, the state space size will change depending on how well-determined the frame boundaries are.

After decoding the current frame, we obtain the posterior probability distribution for the drift at end-of-frame, given by

$$\Pr\left\{S_\tau = m | Y_{m_\tau^-}^{\tau+m_\tau^+}\right\} = \lambda_N(m) / \Pr\left\{Y_{m_\tau^-}^{\tau+m_\tau^+}\right\} = \frac{\lambda_N(m)}{\sum_{m'} \lambda_N(m')} \tag{28}$$

The most likely drift at end-of-frame can be found by

$$\hat{S}_\tau = \arg\max_m \frac{\lambda_N(m)}{\sum_{m'} \lambda_N(m')} = \arg\max_m \lambda_N(m) \tag{29}$$

As in [4], we determine the nominal start position of the next frame by shifting the received stream by $\tau + \hat{S}_\tau$ positions. The initial condition for the forward metric for the next frame, $\hat{\alpha}_0(m)$, is set to

$$\hat{\alpha}_0(m) = \frac{\lambda_N(m + \hat{S}_\tau)}{\sum_{m'} \lambda_N(m')} \tag{30}$$

replacing the initial condition for (4) reflecting a known frame boundary.

### 7.2 Stream look-ahead

Taking advantage of the different constituent encodings in TVB codes, the MAP decoder can make use of information from the following frame to improve the determination of the end-of-frame position. We augment the current block of $N$ symbols with the first $\nu$ symbols from the following block (or blocks, when $\nu > N$), for an augmented block size $N' = N + \nu$. The MAP decoder is applied to the corresponding augmented frame. After decoding, only the posteriors for the initial $N$ symbols are kept; the start of the next frame is determined from the drift posteriors at the end of the first $N$ symbols, and the process is repeated.

Consider the latency of the MAP decoder to be the time from when the first bit of a frame enters the channel to when the decoded frame is available. The cost of look-ahead is an increase in decoding complexity and latency corresponding to the change in block size from $N$ to $N'$. The effect on complexity is seen by using terms corresponding to the augmented block size in the expressions of Table 2. The latency is equal to the time it takes to receive the complete frame and decode it. Look-ahead increases the time to receive the augmented frame linearly with $\nu$ and the decoding time according to the increase in complexity.

The required look-ahead $\nu$ depends on the channel conditions and the code construction. In general, a larger value is required as the channel error rate increases. We show how an appropriate value for $\nu$ can be chosen for a given code under specific channel conditions in Section 8.1. Typical values for $\nu$ are small ($\nu < 10$) for good to moderate channels ($P_\mathrm{i}$, $P_\mathrm{d} < 10^{-2}$). The required look-ahead increases significantly for poor channels: the example in Section 8.1 requires $\nu = 1000$ at $P_\mathrm{i} = P_\mathrm{d} = 2 \times 10^{-1}$.

### 7.3 Comparison with Davey–MacKay decoder

A key feature of the Davey–MacKay construction is the presence of a known distributed marker sequence that is independent of the encoded message. This allows the decoder, in principle, to compute the forward and backward passes over the complete stream. However, to reduce decoding delay, the decoder of [4] performs frame-by-frame decoding using a 'sliding window' mechanism. The 'sliding window' mechanism seems intended to approximate the computation of the forward and backward passes over all received data at once. This approach is similar in principle to ours when stream look-ahead is used; however, there are some critical differences which we discuss below.

**Table 3** Construction parameters of codes used in simulations

| Label[a] | Inner code | Marker | Outer code | Comment |
|---|---|---|---|---|
| P1 | (5, 16) sparse | random, distributed | LDPC (999, 888) $\mathbb{F}_{16}$ | published in [4, Fig. 8, Code D] |
| N1a | (5, 16) sparse | random, distributed | LDPC (999, 888) $\mathbb{F}_{16}$ | identical construction to P1, symbol-level MAP decoder |
| N1b | (10, 256, 3) TVB | none | LDPC (499, 444) $\mathbb{F}_{256}$ | same overall rate and block size as P1 |
| P2 | (6, 8) sparse | random, distributed | LDPC (1000, 100) $\mathbb{F}_8$ | published in [4, Fig. 8, Code I] |
| N2a | (6, 8) sparse | random, distributed | LDPC (1000, 100) $\mathbb{F}_8$ | identical construction to P2, symbol-level MAP decoder |
| N2b | (6, 8, 12) TVB | none | LDPC (1000, 100) $\mathbb{F}_8$ | same overall rate, block size and outer code as P2 |
| P3 | 9 bits, uncoded | 001/110, appended | LDPC (3001, 2000) $\mathbb{F}_2$ | published in [5, Fig. 7, Code D] |
| N3 | 9 bits, uncoded | 001/110, appended | LDPC (2997, 1998) $\mathbb{F}_2$[a] | identical inner code to P3, marginally smaller outer code |
| P4 | not applicable | not applicable | rate-3/14 turbo code $\mathbb{F}_2$ | published in [11, Fig. 4, Code T2] |
| N4 | (7, 8, 8) TVB | none | LDPC (666, 333) $\mathbb{F}_8$ | same overall rate as P4 |
| P5 | not applicable | not applicable | rate-1/10 turbo code $\mathbb{F}_2$ | published in [11, Fig. 4, Code T4] |
| N5 | (7, 4, 32) TVB | none | LDPC (855, 300) $\mathbb{F}_4$ | same overall rate as P5 |

[a]Labels starting with P indicate previously published results, whereas labels starting with N indicate new simulation results.
[b]Obtained by truncating the LDPC (3001, 2000) $\mathbb{F}_2$ of P3. This truncation is necessary so that the outer-encoded sequence can be expressed by an integral number of inner codewords.

In [4], the starting index for a given frame is taken to be the most likely end position of the previous frame, as determined by the Markov model posteriors. This is the same as the approach we use in Section 7.1. However, in [4], the initial conditions of the forward pass are simply copied from the final values of the forward pass for the previous frame. This is consistent with the view that the 'sliding window' mechanism approximates the computation over all received data at once, but contrasts with our method. In Section 7.1, the initial conditions of the forward pass are determined from the posterior probabilities of the drift at the end of the previous frame. These drift posteriors include information from the look-ahead region and from the priors at the end of the augmented frame, which were determined analytically from the channel parameters.

Observe that in the 'sliding window' mechanism of [4], the backward pass values cannot be computed exactly as for the complete stream. Instead, the decoder of [4] computes the forward pass for some distance beyond the expected end of frame position, and initialises the backward pass from that point. The suggested distance by which to exceed the expected end of frame position is 'several (e.g. five) multiples of $x_{\max}$', where $x_{\max}$ is the largest drift considered. The concept is the same as the stream look-ahead of Section 7.2. However, we recommend choosing the look-ahead quantity $v$ based on empirical evidence (see Section 8.1).

It is claimed in [4] that the backward pass is initialised from the final forward pass values; the reasoning behind this is unclear, and does not seem to have a theoretical justification. We initialise the backward pass with the prior probabilities for the drift at the end-of-frame, as explained in Section 7.1.

### 7.4 Initial synchronisation

The only remaining problem is to determine start-of-frame synchronisation at the onset of decoding a stream. This can be obtained by choosing state space limits $M_\tau$ large enough to encompass the initial desynchronisation and by setting equiprobable initial conditions: $\alpha_0(m) = \beta_N(m) = 1/M_\tau \ \forall m$. Previous experimental results [4] have assumed a known start for the first frame, with the decoder responsible for maintaining synchronisation from that point onwards. We adopt the same strategy in the following.

## 8 Results

Practical results are given in this section. We show how an appropriate choice of decoder parameters allows stream decoding to perform as well as when frame boundaries are known. Results are also given for existing constructions which can be expressed as TVB codes, showing how the symbol-level MAP decoder improves on the original decoder (in the case of [4]) or is equivalent (in the case of [5]). We also demonstrate some improved constructions allowed by the flexibility of TVB codes. These are achieved by using simulated annealing to find TVB codes of a required order with a good Levenshtein distance spectrum. Specifically, we seek to find constituent codes with the highest possible minimum Levenshtein distance and the lowest multiplicity at small distances. For all codes so designed, $M < N$; we construct our TVB codes using a random sampling with replacement of the unique constituent codes, and use this as our inner code. Construction parameters for all codes used in this section are given in Table 3. To facilitate reproduction of these results, the TVB codebooks used are available for download from the first author's web site [http://jabriffa.wordpress.com/publications/data-sets/].

### 8.1 Stream decoding with MAP decoder

The results of [13] assumed known frame boundaries; under these conditions, the decoder is arguably at an advantage when comparing with the results of [4, 5]. It is also not clear whether the MAP decoder can keep track of frame boundaries with the non-sparse constructions of [13, 15] and the TVB codes introduced here, especially in the absence of a known marker sequence. In the following, we investigate the performance of the MAP decoder under stream decoding conditions, and consider the choice of look-ahead required. As in [4], we assume that the start of the first frame is known, while the decoder is responsible for keeping synchronisation from that point onwards. We use the limits specified in Section 4.

We start by investigating the effect of stream decoding on the ability of the MAP decoder to track codeword boundaries. We consider a (6, 8, 12) TVB code, which is the inner code for the concatenated system N2b of Table 3. We simulate this inner code with a block size $N = 2000$ under the channel conditions at the onset of convergence for the concatenated system, that is at $P_i = P_d = 0.22$, assuming only the start position of the first frame is known. At each codeword boundary, we plot the fraction of correctly determined drifts (fidelity) in Fig. 4. As expected, the fidelity drops at the end of the frame, where the actual drift is unknown to the decoder. However, it can be observed that the fidelity reaches a steady high value within about 1000 codewords from the end of frame. It could therefore be supposed that a look-ahead of $v = 1000$ would be sufficient for this code under these channel

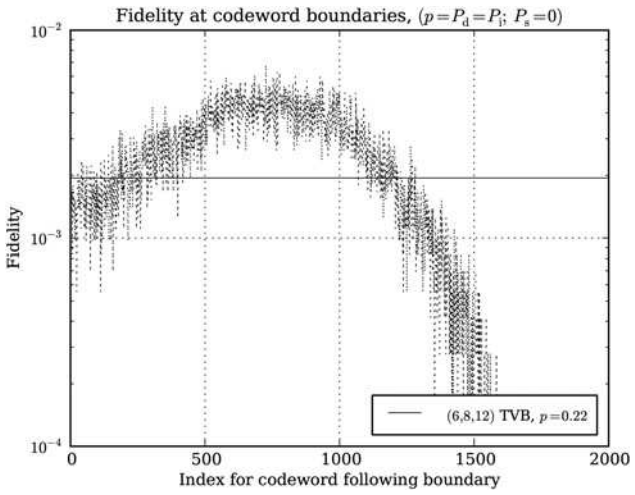


**Fig. 4** *Fraction of correctly resynchronised codeword boundaries (fidelity) as a function of codeword index, for code N2b of Table 3*

conditions. The dip at the start of the frame is caused by the uncertainty in the frame start position, because of the very low fidelity at the end of the previous frame.

To test this hypothesis, we concatenate this inner code with the (1000, 100) LDPC code over $\mathbb{F}_8$ of [4, Code I]. We simulate this system under the following conditions: (i) known frame start and end (frame decoding); (ii) known start for the first frame, unknown frame ends (stream decoding), no look-ahead; and (iii) stream decoding with look-ahead $\nu = 1000$ codewords. Results are shown in Fig. 5. We give results after the first and fifth iterations. As anticipated, performance under stream decoding conditions is poorer than frame decoding if there is no look-ahead. However, an appropriate look-ahead quantity allows the decoder to perform as well under stream decoding as under frame decoding.

It is important to highlight that this result is dependent on the inner code structure, and that therefore the generalisation to other constructions is not obvious. However, we have repeated the same test with other constructions, including those of [4, 5, 13–15] and the new constructions in this paper, and under different channel conditions. In all cases we have found that the result is repeatable, in that it is possible to approach the performance of frame decoding with stream decoding, as long as an appropriate look-ahead quantity is chosen. The only cost of stream decoding is therefore the need for a fidelity analysis to determine a suitable look-ahead value and the increased decoding latency and complexity caused by the augmented block size. Since the code performance is undiminished, to simplify our analysis from this point onwards we assume known frame start and end positions.

### 8.2 *Comparison with prior art*

We have already shown in [13] that the (symbol-level) MAP decoder allows us to obtain better performance from the codes of [4]. Further improvement can be obtained with iterative decoding, as we show here. Additionally, the flexibility of TVB codes allows us to obtain codes that perform better at the same size and/or rate. In the following, we simulate channel conditions $P_i = P_d$; $P_s = 0$ in order to compare with published results.

For low channel error rates, consider [4, Code D], listed as P1 in Table 3. We compare the previously published result with a MAP decoding of the same code (N1a) in Fig. 6. As shown in [13], the MAP decoder improves the performance of this code even after the first iteration; additional iterations improve the result further. At the same overall code rate and block size, we can improve the performance further by designing an inner TVB code with a better Levenshtein distance spectrum (N1b). We repeat the process at higher channel error rates for [4, Code I], listed as P2 in Table 3. Again, compared with the published result, a MAP decoding of the same code (N2a) improves performance even after the first iteration, also in Fig. 6. Additional iterations improve the result, but the difference in this case is less pronounced. Replacing the inner code with one of the same size, but a better Levenshtein distance spectrum (N2b) improves performance further.

As we have already discussed in Section 2.1, the marker codes given by Ratzer [5] can also be cast as TVB codes. In [5], binary outer LDPC codes were used. To use a binary outer code with our MAP decoder, the bitwise APPs can be obtained from the $q$-ary symbol APPs by marginalising over the other bits [21, p. 326]; these are then passed to the decoder for the binary outer code. In this case, for a binary outer code we expect the performance of the concatenated code to be identical, whether the inner code is decoded with the bit-level (MAP) decoder of [5] or with our symbol-level decoder. We show this in Fig. 7 for [5, Code

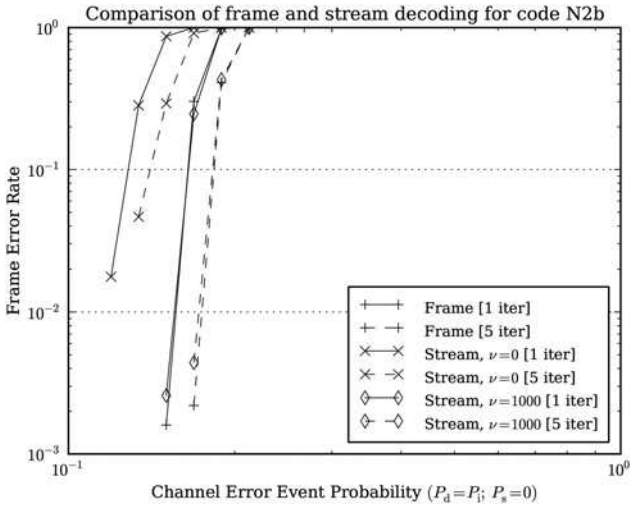


**Fig. 5** *Demonstration of the effect of look-ahead on the MAP decoder's performance under stream decoding conditions and comparison with frame decoding*

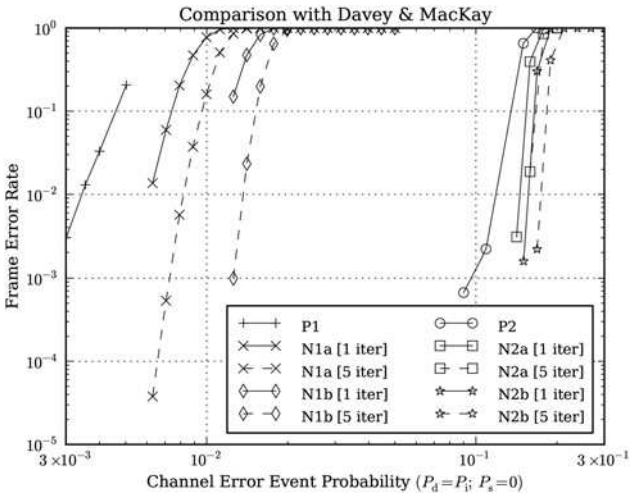


**Fig. 6** *Comparison with Davey–MacKay: improving the performance of [4, Code D] (left) and [4, Code I] (right) using our MAP decoder, iterative decoding and an inner code with better Levenshtein distance spectrum*

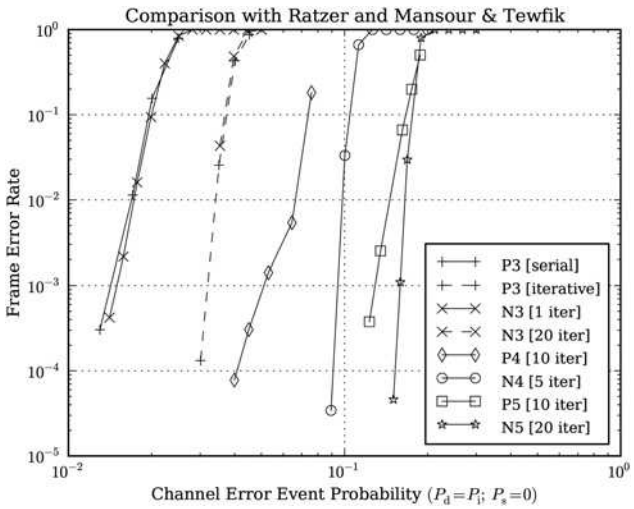


**Fig. 7** *Comparison with Ratzer and with Mansour and Tewfik: demonstrating the equivalence of our MAP decoder on [5, Code D], with and without iterative decoding and improving performance on [11, Codes T2, T4] at the same overall code rate, using inner codes with improved Levenshtein distance in concatenation with LDPC outer codes*

D], listed as P3 in Table 3, in comparison with an almost-identical code (N3) using our MAP decoder. It is important to highlight that decoding marker codes as TVB codes provides no material advantage; in fact, a cost is paid in complexity for doing so. We do not propose or expect that marker codes will be decoded as TVB codes. However, there is value in showing that marker codes can be decoded as TVB codes with no loss in performance. Specifically, this allows us to compare the structure of marker codes with other constructions, within the same context.

Finally, in Fig. 7 we compare with the more recent turbo codes of [11, Codes T2, T4], respectively listed as P4 and P5 in Table 3, with concatenated systems of the same overall rate, using a TVB inner code and an LDPC outer code. It can be seen that our concatenated systems outperform the codes of [11], significantly at lower channel error rates and somewhat less so at higher channel error rates.

## 9 Conclusions

In this paper we have considered TVB codes, which generalise a number of previous codes for synchronisation errors. We discussed the applicable design criteria for TVB codes, expressing some previously published codes as TVB codes and showing that the greater flexibility of TVB codes allows improved constructions. For example, our (7, 8, 4) TVB code achieves a symbol error rate of $10^{-4}$ at a $P_i$, $P_d$ that is almost two orders of magnitude higher than a marker or distributed marker code of the same size, and slightly better than our earlier SEC codes.

We also considered a number of important issues related to a practical implementation of the corresponding MAP decoder. Specifically, we have given an expression for the expected distribution of drift between transmitter and receiver due to synchronisation errors, with consideration for practical concerns when evaluating this expression. We have shown how to determine an appropriate choice for state space limits based on the drift probability distribution. The decoder complexity under given channel conditions is then expressed as a function of the state space limits used. For a given state space, we have also given a number of optimisations that reduce the algorithm complexity with no further loss of decoder performance. The proposed speedups, which are independent of the TVB code construction, result in a considerable reduction in complexity of almost two orders of magnitude for typical code sizes and channel conditions. For code constructions with appropriate mathematical structure, we expect to be able to replace the receiver metric, which considers each possible transmitted codeword, with a faster soft-output algorithm. Next, we have considered the practical problem of stream decoding, where there is no prior knowledge of the received frame boundary positions. In doing so, we have also shown how an appropriate choice of decoder parameters allows stream decoding to approach the performance when frame boundaries are known, at the expense of some increase in complexity.

Finally, practical comparisons of TVB codes with earlier constructions were given, showing that TVB code designs can in fact achieve improved performance. Even compared with the state-of-the-art codes of [11], the TVB codes presented here achieve a frame error rate of $10^{-3}$ at 24% higher $P_i$, $P_d$ for a rate-1/10 code and at 84% higher $P_i$, $P_d$ for a rate-3/14 code. We expect further improvements to the codes shown here to be possible, particularly by co-designing optimised outer codes. However, a detailed treatment of the design process is beyond the scope of this paper, and will be the subject of further work.

## 10 Acknowledgment

This research has been carried out using computational facilities procured through the European Regional Development Fund, Project ERDF-080.

## 11 References

[1] Sellers F.F.: 'Bit loss and gain correction code', *IRE Trans. Inf. Theory*, 1962, **IT-8**, pp. 35–38
[2] Levenshtein V.I.: 'Binary codes capable of correcting deletions, insertions and reversals', *Sov. Phys.–Dokl.*, 1966, **10**, (8), pp. 707–710
[3] Gallager R.G.: 'Sequential decoding for binary channels with noise and synchronization errors'. Technical Report, 2502, Massachusetts Institute of Technology Lexington Lincoln Laboratory, 27 October 1961
[4] Davey M.C., MacKay D.J.C.: 'Reliable communication over channels with insertions, deletions, and substitutions', *IEEE Trans. Inf. Theory*, 2001, **47**, (2), pp. 687–698
[5] Ratzer E.A.: 'Marker codes for channels with insertions and deletions', *Ann. Telecommun.*, 2005, **60**, pp. 29–44
[6] Wang F., Fertonani D., Duman T.M.: 'Symbol-level synchronization and LDPC code design for insertion/deletion channels', *IEEE Trans. Commun.*, 2011, **59**, (5), pp. 1287–1297
[7] Swart T., Ferreira H., dos Santos M.: 'Using parallel-interconnected Viterbi decoders to correct insertion/deletion errors'. Seventh AFRICON Conf. Africa, September 2004, vol. 1, pp. 341–344
[8] Schlauweg M., Profrock D., Muller E.: 'Correction of insertions and deletions in selective watermarking'. IEEE Int. Conf. Signal Image Technology and Internet Based Systems (SITIS), 2008, 30 November–3 December 2008, pp. 277–284
[9] Mansour M., Tewfik A.: 'Convolutional decoding in the presence of synchronization errors', *IEEE J. Sel. Areas Commun.*, 2010, **28**, (2), pp. 218–227
[10] Mansour M.F., Tewfik A.H.: 'Convolutional decoding in the presence of synchronization errors', *IEEE J. Sel. Areas Commun.*, 2010, **28**, (2), pp. 218–227
[11] Mansour M.F., Ahmed H.T.: 'A turbo coding scheme for channels with synchronization errors', *IEEE Trans. Commun.*, 2012, **60**, (8), pp. 2091–2100
[12] Mercier H., Bhargava V., Tarokh V.: 'A survey of error-correcting codes for channels with symbol synchronization errors', *IEEE Commun. Surv. Tutor.*, 2010, **12**, (1), pp. 87–96
[13] Briffa J.A., Schaathun H.G., Wesemeyer S.: 'An improved decoding algorithm for the Davey–MacKay construction'. Proc. IEEE Int. Conf. Communications, Cape Town, South Africa, 23–27 May 2010
[14] Briffa J.A., Schaathun H.G.: 'Improvement of the Davey–MacKay construction'. Proc. IEEE Int. Symp. Information Theory and its Applications, Auckland, New Zealand, December 7–10, 2008, pp. 235–238

[15] Buttigieg V., Briffa J.A.: 'Codebook and marker sequence design for synchronization-correcting codes'. Proc. IEEE Int. Symp. Information Theory, St. Petersburg, Russia, 31 July–5 August 2011
[16] Briffa J.A.: 'A GPU implementation of a MAP decoder for synchronization error correcting codes', *IEEE Commun. Lett.*, 2013, **17**, (5), pp. 996–999
[17] Bahl L.R., Jelinek F.: 'Decoding for channels with insertions, deletions, and substitutions with applications to speech recognition', *IEEE Trans. Inf. Theory*, 1975, **21**, (4), pp. 404–411
[18] Perez L.C., Seghers J., Costello Jr., D.J.: 'A distance spectrum interpretation of turbo codes', *IEEE Trans. Inf. Theory*, 1996, **42**, (6), pp. 1698–1709
[19] Ferrari G., Chugg K.M.: 'Linear programming-based optimization of the distance spectrum of linear block codes', *IEEE Trans. Inf. Theory*, 2003, **49**, (7), pp. 1794–1800
[20] Davey M.C.: 'Error-correction using low-density parity-check codes'. PhD dissertation, University of Cambridge, 1999
[21] MacKay D.J.C.: 'Information theory, inference, and learning algorithms' (Cambridge University Press, 2003)